Abstract

One of the most used algorithms to generate hexahedral meshes for extrusion volumes is the multi-sweeping method. The algorithm decomposes the geometry into many-to-one sub-volumes and then meshes each sub-volume separately.

However, the quality of the final mesh depends on the decomposition process. First, the location of inner nodes created during the decomposition process may induce bad quality elements. To avoid this drawback, we propose a three-stage decomposition process to locate those nodes. Second, the imprinting process is not robust when dealing with non-planar surfaces. For this reason, we introduce the new concept of the computational domain. The computational domain is a planar representation of the levels of the geometry. In this way, we improve the operations needed to perform imprints.

Keywords: mesh generation, hexahedral mesh, multi-sweeping, computational domain, geometry decomposition, inner node location.

1 Introduction

The generation of unstructured hexahedral meshes is still an open problem. Several methods have appeared in the literature to generate such meshes. However, an unstructured hexahedral mesher that can discretize any arbitrary geometry does not exist. Therefore, specific algorithms have been developed for specific types of geometries that often appear in industrial applications. For instance, sweep algorithms [1, 2, 3, 4] have been successfully applied for extrusion geometries defined by a single source surface and a single target surface (one-to-one sweeping algorithms). This method evolved into the many-to-one algorithm [5] that allows meshing extrusion volumes with several source surfaces but a single target surface. The method decomposes the

initial geometry into one-to-one barrels that can be meshed using a one-to-one sweeping scheme. In the last years, several algorithms have appeared to mesh many-to-many extrusion volumes (multi-sweeping algorithms) [6, 7]. This algorithm decomposes the volume into many-to-one sub-volumes. Then, each sub-volume is further decomposed into one-to-one barrels. The decomposition is performed by means of projecting nodes along the sweep direction and imprinting surfaces.

However, the quality of the final mesh is directly related to the robustness of the imprinting process and to the location of inner nodes created during the decomposition process, especially for geometries with curved surfaces. To overcome these drawbacks, we propose two original contributions. On the one hand, we introduce the new concept of the computational domain for sweep geometries. The computational domain is a planar representation of the sweep levels and it allows improving several geometric calculations performed during the imprinting process. This results in a more robust imprinting process.

On the other hand, we propose to improve the location of inner nodes by performing a three-stage decomposition process. In the first stage, we project the nodes on target surfaces to source surfaces. At this stage, the decomposition of the geometry is determined, but it is desirable to improve the location of inner nodes. To this end, in the second stage, we project back the nodes from source surfaces to target surfaces. In the third stage, the final location of inner nodes is computed as a weighted average of the nodes projected from target surfaces and the nodes projected from source surfaces.

Finally, we present several examples that illustrate the properties and applicability of the proposed strategies.

2 Outline of the multi-sweeping algorithm

The multi-sweeping method can be decomposed into four steps that are detailed in the following sections:

(i) Surface classification and linking face discretization.

(ii) Loop face projection and imprinting.

(iii) Loop edge meshing and volume decomposition into many-to-one sub-volumes.

(iv) Many-to-one sub-volumes discretization.

3 Surface Classification and linking sides discretization

The first step of the algorithm is the classification of the geometry surfaces. Each surface is classified as *source*, *target* or *linking side*. Source surfaces are the ones

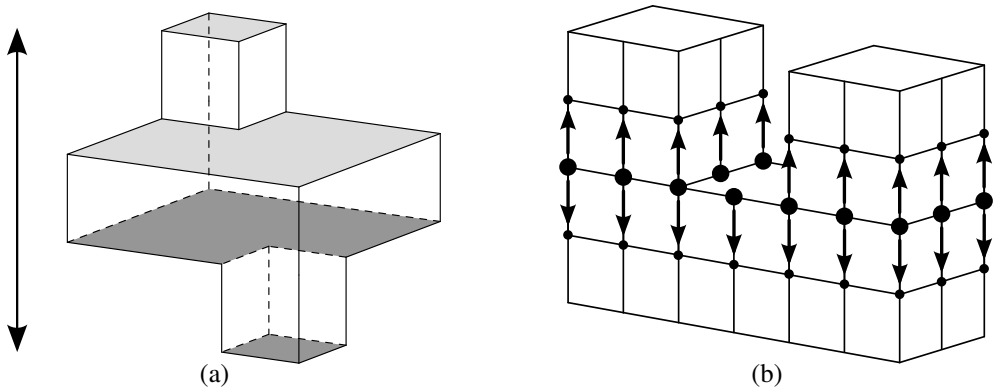


Figure 1: (a) Surface classification for a simple geometry: source surfaces (light grey), target surfaces (dark grey) and linking sides (transparent). (b) Linking sides mesh and detail of a level of sweep nodes.

that are meshed using quadrilateral elements. The mesh on source surfaces is then projected to the target surfaces. Finally, linking sides are the surfaces that connect source and target surfaces. The classification is accomplished using the procedure presented in [8]. This procedure finds a non-submappable surface and classifies it as target surface. Then, adjacent surfaces are classified as source, target or linking sides depending on the angle between them and the original one. The algorithm is iterated in an advancing front manner until all the surfaces are classified. Figure 1(a) presents a simple many-to-many extrusion volume with its surfaces properly classified as source, target and linking sides. When the surfaces are classified, we mesh the linking sides using the submapping method. Figure 1(b) shows the mesh generated on the linking sides using the submapping method. Note that the structured quadrilateral mesh for the linking sides determines a set of node levels between source and target surfaces.

4 Data Structures

This section defines three basic data structures that will be used through this work. Although they were previously introduced in [7], we include them for completeness.

4.1 Sweep Nodes

Sweep nodes are a computational structure used in the algorithm to store the connectivity of the linking sides mesh along the sweep direction. Each sweep node stores two pointers: a pointer to the next sweep node in the sweep direction and a pointer to the previous sweep node in the counter-sweep direction. It is worth to notice that some of these pointers may be null. Figure 1(b) shows an extrusion geometry with a row of sweep nodes. Note that there are sweep nodes that do not point to a sweep node in the next or the previous level.

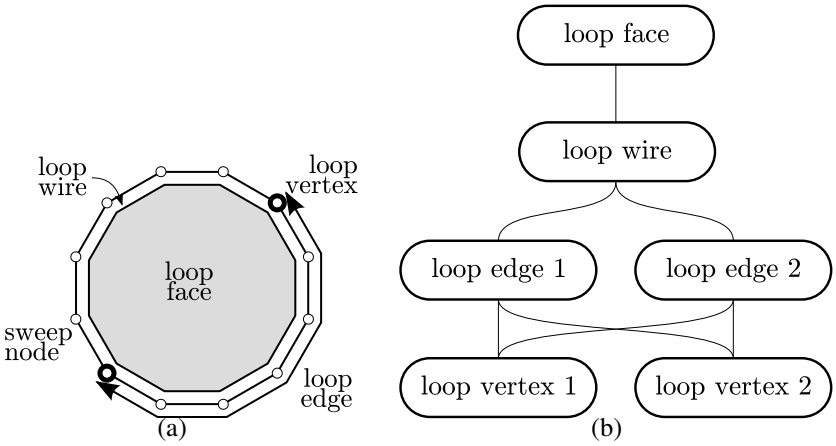


Figure 2: Description of: (a) a loop face and (b) its topology graph.

4.2 The Loop Geometry Engine

In order to decompose the geometry, we represent the source, target and mid-level faces as loops of sweep nodes. To this end, we use the *loop geometry* engine proposed in [7]. For instance Figure 2(a) shows a circular face represented by: *1)* a *loop face*; *2)* a *loop wire* that describes the boundary of the loop face; *3)* an ordered list of two *loop edges* that defines the loop wire and *4)* two *loop vertices* that define the initial and final points of each loop edge.

Loop vertices are represented by a sweep node. Loop edges are defined by an ordered list of sweep nodes. This list 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. In addition, we use the topology graph to describe the topology of the loop faces. Figure 2(b) shows the topology graph of the loop face presented in Figure 2(a). For instance, the loop wire uses loop edge 1 and, conversely, loop edge 1 is included in the loop wire.

Loop geometry engine structure is similar to a geometry engine, except that not all the elements in the loop geometry engine 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 the loop geometry and maintaining the topology graph during the whole decomposition process.

4.3 Control Loops

Control loops are defined as the loops of sweep nodes that bound each sweep level, large dots in Figure 3. In a many-to-many geometry the number of control loops may differ from level to level. For instance, in Figure 3 we show two levels of a many-to-one geometry. Note that the lower level is described by a single control loop whereas the upper level is defined by two control loops. Each control loop is composed by

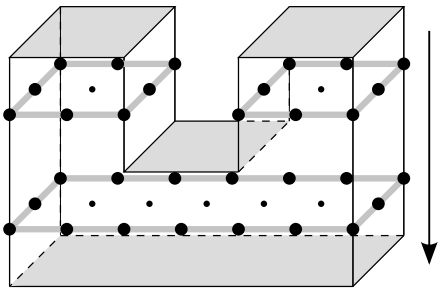


Figure 3: A simple geometry showing three control loops (thick grey).

a loop of sweep nodes that define the outer boundary of the sweep level and several loops of sweep nodes that define inner boundaries of the sweep level.

Control loops are used to project sweep nodes between two consecutive sweep levels. To this end, they store a list of nodes to project to the next level, small dots in Figure 3. To construct the control loops we propose the following procedure. Given two sets of target and source loop faces of the same level, $(\tau_i)_{i=1,\dots,n}$ and $(\sigma_i)_{i=1,\dots,m}$, respectively, the control loops are calculated as:

(i) Compute provisional target control loops, CL_t, as follows:

- The loops of nodes of control loops are defined as the boolean union $CL_t = \bigcup_{i=1,\dots,n} \tau_i$, see [9] for more details about the boolean union.
- The nodes to project are those that belong to more than one loop face and are not contained in the loops of CL_t.

(ii) Compute provisional source control loops, CL_s, using the same procedure detailed in step (i).

(iii) Collect the nodes of control loops in CL_s that are not included in CL_t. Then, we insert those nodes in the corresponding control loop in CL_t.

Note that in the previous procedure we can skip the second and the third steps if there are not source loop faces in the given level.

5 The Computational Domain

In this section we introduce the new concept of computational domain. It is used to improve the the robustness of the imprinting process. The *computational domain* of a given control loop is a planar representation of it. For instance, in Figure 4 we present a control loop in the physical domain (white squares) and its representation in the computational space (black circles). The computational domain is used to: *1.*- project the sweep nodes through the volume; *2.*- perform the imprinting between loop faces; and *3.*- project sweep nodes onto a geometrical source surface, see Section 6.

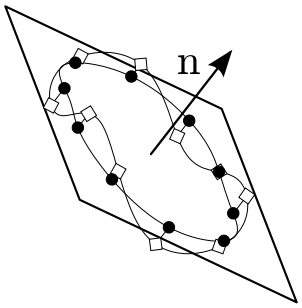


Figure 4: Definition of the pseudo-normal vector from the physical sweep nodes (white squares) and computational sweep nodes (black circles).

To obtain a planar representation of control loops we use the *pseudo-area* and the *pseudo-normal* vectors considered in [4]. Given a loop of sweep nodes $X = \{\mathbf{x}_i\}_{i=1,\dots,n}$, we define the pseudo-area vector as

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

where $\mathbf{x}_{n+1} = \mathbf{x}_1$. If the control loop is defined by p loops of sweep nodes, $X_1, \dots, X_p$ the pseudo area vector is defined as

$$\mathbf{a} := \mathbf{a}_1 + \cdots + \mathbf{a}_p, \tag{2}$$

where $\mathbf{a}_1, \dots, \mathbf{a}_p$ are the pseudo-areas of loops $X_1, \dots X_p$, respectively. The pseudo-normal vector is defined as

$$\mathbf{n} := \frac{\mathbf{a}}{||\mathbf{a}||}. \tag{3}$$

The construction of the computational domain of a given control loop is performed in the following manner. First, we compute the pseudo-normal vector (3) of the loops of nodes that define the control loop. Second, we define the computational position of $\mathbf{x}_i$, for $i = 1, \dots, n$ as

$$\overline{\mathbf{x}}_i = \mathbf{x}_i - \langle \mathbf{x}_i, \mathbf{n} \rangle \mathbf{n}, \tag{4}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product. Note that we project all points $\mathbf{x}_i$ to the plane defined by the pseudo-normal vector. In [4], it is proved that among all the possible planar projections of the loop of points, this one maximizes the area of the projected loop. From the 3D representation of the computational domain given by (4), it is straightforward to compute a 2D representation, in (ξ, η) coordinates, of the computational domain, see details in Section 6.3.

6 Loop Face Projection and Imprinting

To perform the loop face projection and imprinting process we propose to the following four steps:

(i) Loop face projection.

(ii) Loop face imprinting pre-process.

(iii) Loop face imprinting.

(iv) Final location of inner sweep nodes.

6.1 Loop Face Projection

We start the projection process at the first sweep level. We collect all the target surfaces and compute the associated loop faces as detailed in Section 4.2. Then, the corresponding control loops are calculated according to Section 4.3. Note that there are no source surfaces on the first level. Once the control loops are constructed, the algorithm proceeds to calculate their associated computational domain, see Section 5. The next step is to project the inner sweep nodes to create the next level loop faces. In order to project the sweep nodes between levels we use the sweeping scheme presented in [10, 4]. It is based on a least-squares approximation of an affine mapping defined between two consecutive control loops. Note that the projection of sweep nodes is performed both in the physical domain and the 3D representation of computational domain. The physical and the 3D computational locations are stored for each sweep node. When sweep nodes are projected, the new sweep nodes are connected to the ones of the previous level. The loop faces of the next level are created using the nodes of the next level pointed from the nodes of the current level. In addition, we also create the loop vertices, loop edges and loop wires in the same manner. Finally, the loop geometry engine is updated in order to reflect the new changes of the loop geometry. During the loop face projection process, the new loop faces created in the next sweep level are considered as target loop faces. At the next level, we collect all the target and source loop faces and update the corresponding control loops. Finally, if it is required an imprinting process is performed for each resulting control loop.

6.2 Loop Face Imprinting Pre-Process

The loop face imprinting pre-process is performed in the computational domain and then, the physical domain is updated accordingly. Figure 5 presents the updated physical domain after each one of the imprinting pre-process steps. Figure 5(a) shows two target loop faces (dark grey) and one source loop face (light grey). First, the loop edges of target loop faces are matched each other in the 3D representation of the computational domain to ensure a conformal loop geometry, see Figure 5(b). Second, we project the sweep nodes of target loop faces on the geometrical surface of source loop faces, see Figure 5(c). This step is performed using the 2D representation of the computational domain as detailed in Section 6.3. Third, we search for a sweep node in a target loop face and a sweep node in a source loop face that are closer than a given tolerance in the 3D representation of the computational domain. Then, we move the

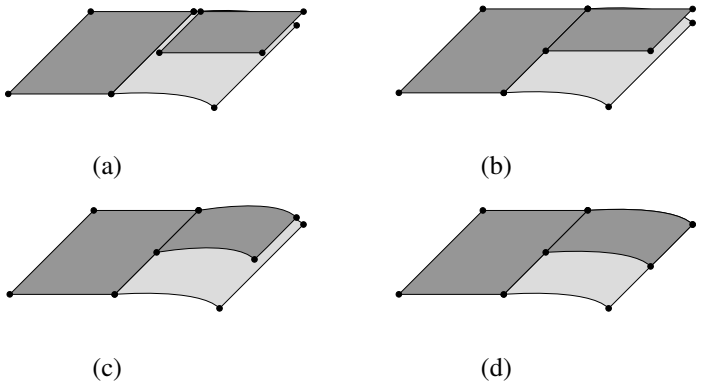


Figure 5: (a) Two target loop faces (dark) and a source loop face (light). (b) Target loop edges intersection. (c) Projection of target sweep nodes on source faces. (d) Collapse of target sweep nodes and source sweep nodes.

target sweep node to the location of the source sweep node in order to collapse them, see Figure 5(d).

6.3 Mapping of Sweep Nodes from the Computational Domain to Source Surfaces

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 [11]. This algorithm is robust when dealing with planar loops of nodes. However, when the loops of nodes are not planar, the algorithm may fail to give a correct answer.

Given a list of sweep nodes inside a source loop face, we have to project them on the surface that defines the source loop face. Several algorithms have been developed to project nodes onto a given surface. Most of them involve an orthogonal projection of nodes onto the target surface. Note that these projections are expensive from a computational point of view since it is necessary to solve a root finding problem for each node to project. To overcome this drawback, we propose to use a method previously developed to map meshes between two surfaces [3]. Let (ξ, η) and (x, y, z) be the coordinate systems of the 2D representation of the computational domain and of the physical domain, respectively. Note that (ξ, η) coordinates are not univocally defined. For instance, we can apply a planar rotation to obtain another valid coordinate system. Note that the procedure to project the nodes to source surfaces does not depend on which coordinate system is selected.

Once the 2D coordinates of the sweep nodes are obtained, we have to calculate a mapping, $\boldsymbol{X}(\xi, \eta)$, to project the computational nodes to the physical domain, see Figure 6. Note that source loop faces represent a geometric surface. Thus, instead of projecting the nodes directly to the physical surface, we propose to compute a mapping, $\boldsymbol{\Pi}(\xi, \eta)$, to map the nodes to the parametric domain of the surface. Then,

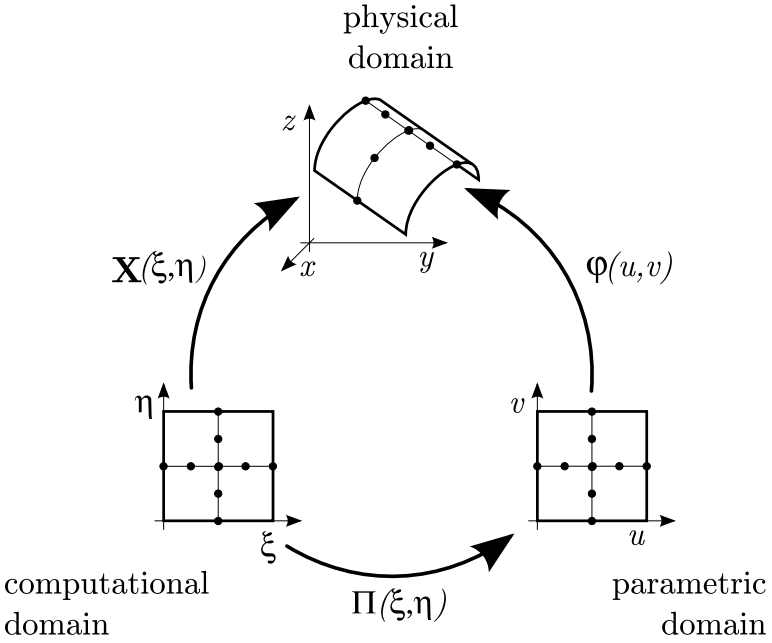


Figure 6: Illustration of the mapping of sweep nodes from the computational domain to source surfaces

using the parametrization of the surface, $\boldsymbol{\varphi}(u, v)$, we map the corresponding nodes to the physical domain. That is,

$$\boldsymbol{X} = \boldsymbol{\varphi} \circ \boldsymbol{\Pi}.$$

Note that the mapping $\boldsymbol{\Pi}(\xi, \eta)$ is unknown. In this work we propose to approximate $\boldsymbol{\Pi}(\xi, \eta)$ by means of a least-squares approximation of an affine mapping, $\widetilde{\boldsymbol{\Pi}}(\xi, \eta)$, defined between the 2D representation of the computational domain and the parametric domain of the surface, see [3]. That is, we approximate $\boldsymbol{X}(\xi, \eta)$ by

$$\boldsymbol{X} \approx \boldsymbol{\varphi} \circ \widetilde{\boldsymbol{\Pi}}.$$

6.4 Loop Face Imprinting

The result of the imprinting process between target loop faces, $(\tau_i)_{i=1...n}$, and source loop faces, $(\sigma_i)_{i=1...m}$, 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, $(\tau_i)_{i=1...n}$. 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, $(\sigma_i)_{i=1...m}$. The loop face imprinting operation is based on segment intersections, see [7, 9] for more details. The main difference of the presented method is that all of

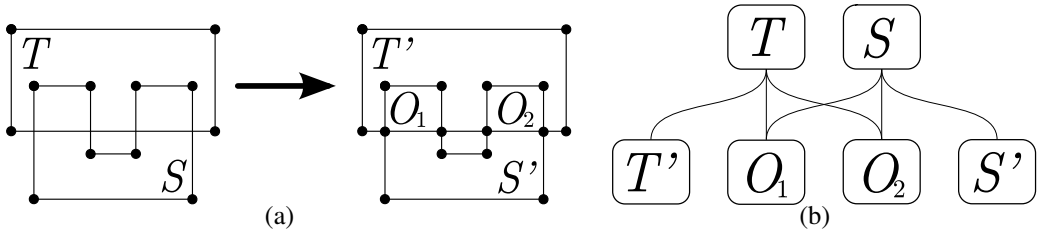


Figure 7: (a) Imprinting process of a target loop face (T) and a source loop face (S). (b) Loop face partitioning graph.

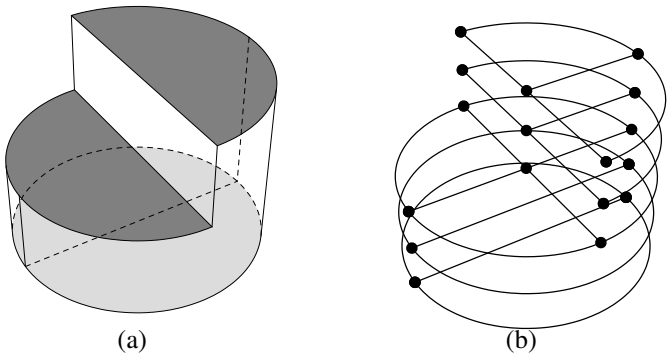


Figure 8: A many-to-many extrusion geometry and its corresponding loop faces. (a) Wire frame model showing two target surfaces (dark grey) and two source surfaces (light grey). (b) Loop faces.

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.

We also create a new graph that relates each of the new loop faces with the old loop faces from which they came from. This graph is denoted as *loop face partitioning graph*. It allows recovering the set of loop faces in which a given loop face is decomposed during the imprinting process. In addition, it also allows recovering the loop faces in which a given loop face is included. Figure 7(a) shows the imprinting process of a target loop face, T, and a source loop face, S. In this example, the results of the imprinting are $\Theta_o = \{O_1, O_2\}$, $\Theta_t = \{T'\} = \{T - S\}$ and $\Theta_s = \{S'\} = \{S - T\}$. Figure 7(b) presents the associated partitioning graph. In this case, the old target loop face, T, is composed by the new loop faces T', O_1 and O_2. Moreover, the new overlap loop face O_1 is a common section of the old target loop face T and the old source loop face S. This information is required to perform the final decomposition process and the discretizacion of all many-to-one sub-volumes.

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. The projection and imprinting process is iterated until the last sweep level is reached. Figure 8(a) presents a multi-sweep geometry with two

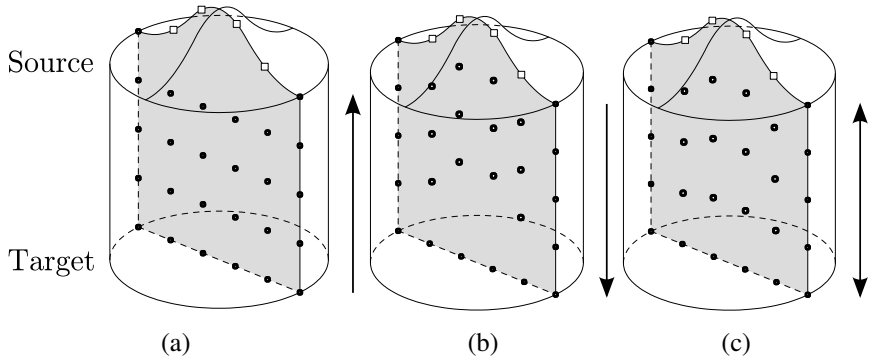


Figure 9: Projection of inner nodes in a many-to-many geometry. (a) Projecting from target surface. (b) Projecting from source surface. (c) Weighted average between projections from both cap surfaces.

target surfaces (light grey) and two source surfaces (dark grey). Figure 8(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.

6.5 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. Note that this idea has been previously introduced during the meshing process of one-to-one barrels [6, 10].

For each overlap loop face, ω, obtained during the imprinting process, the nodes that have to be projected to target surfaces are those in ω such that their previous node lies inside the volume, but not at the boundary. Figure 9 presents a many-to-many extrusion geometry with a planar bottom surface split in two patches (classified as target), and a curved top surface (classified as source). Sweep nodes on cap surfaces that have to be projected through the volume are marked using white squares. Figure 9(a) shows the position of inner nodes projected from target surfaces. These nodes do not capture the shape of the source surface. On the contrary, Figure 9(b) presents the location of inner nodes projected from source surfaces. The nodes do not reproduce the shape of the target surfaces. However, Figure 9(c) presents the node locations by averaging the position of nodes projected from target and source surfaces. Note that the nodes reproduce the shape of both cap surfaces.

For each overlap reference loop face, ω, and the list of nodes to be projected, $nodes$, Algorithm 1 presents a procedure to relocate inner nodes. First, at Line 4, the algorithm finds a source control loop, CL_s, and a target control loop, CL_t, 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 Algorithm 1, the node projec-

Algorithm 1 Inner nodes creation

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

tor is constructed as detailed in [10, 4]. 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 surfaces is recovered. Next, at Line 12 the final position of the previous node is obtained by a weighted average of the computed location projecting from the target and source surfaces. 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.

7 Loop Edge Meshing and Volume Decomposition

In order to generate an hexahedral mesh, we need 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 [12]. However, in order to reduce the computational cost of the integer linear problem, it is only necessary to impose an even number of intervals to the source loop faces obtained during the im-

Algorithm 2 Selection of target and source control loop

Return : ControlLoop CL_t, CL_s

function `ObtainTargetAndSourceControlLoops`(LoopFace ω)
    LoopFace $S \leftarrow$ `obtainContainingLoopFace`(ω)
    if `isNull`(S) **then**
        LoopFace $T \leftarrow$ `previousLoopFace`(ω)
    else
        LoopFace $T \leftarrow$ `previousLoopFace`(S)
    end if
    $CL_t \leftarrow$ `getControlLoop`(T)
    $CL_s \leftarrow$ `nextControlLoop`(CL_t)
    $\omega \leftarrow T$
end function

printing process. Then, the information is propagated in the sweep direction. Hence, the new integer linear problem is

$$\begin{aligned} &\min \sum_{e \in \mathcal{E}} n_e \\ &\text{subject to:} \\ &\sum_{e \in \sigma} n_e = 2n_\sigma \quad \text{for all source loop face } \sigma, \\ &n_e \geq N_e \qquad \text{for all source loop edges } e \in \mathcal{E}, \end{aligned} \tag{5}$$

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.

When the loop edges are meshed, we proceed to decompose the geometry into many-to-one sub-volumes. For each one of the target surfaces, we collect every loop face *stacked* over it in order to define a many-to-one sub-volume. Figure 10 presents a geometry decomposed into two sub-volumes. Note that each sub-volume only contains a single target loop face.

8 Mesh Examples

This section presents four examples of meshes generated using the proposed multi-sweeping method. In the first two examples the proposed method is applied to multi-sweep geometries defined by planar cap surfaces. The first example presents a geometry in which the number of control loops differ from level to level. Note that, the method automatically detects when it is necessary to create or delete control loops. Figure 11(a) presents the mesh generated of a wall support using the multi sweeping method. The second example shows a mesh generated on a multi-sweep volume. In this example, there are edges that have to be matched during the imprinting process.

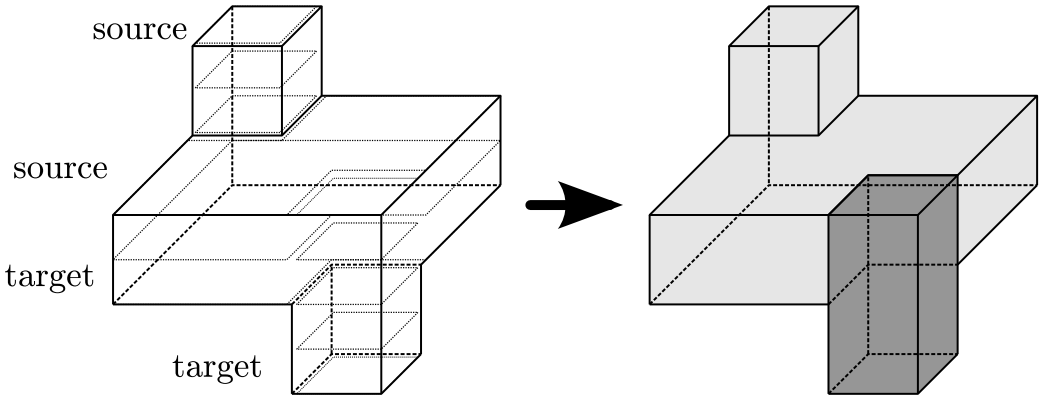
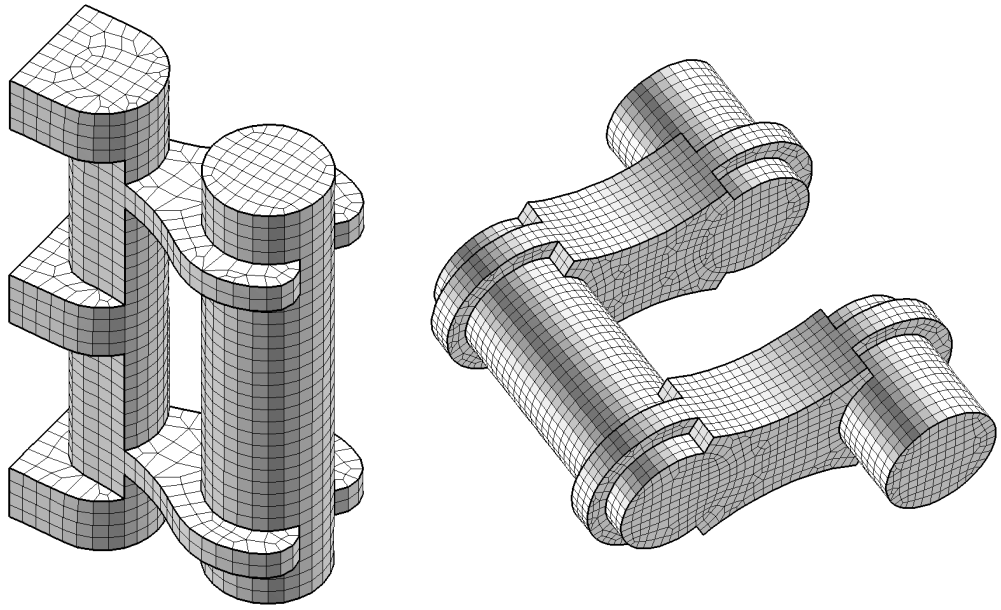


Figure 10: Decomposition of a many-to-many extrusion volume into many-to-one sub-volumes.



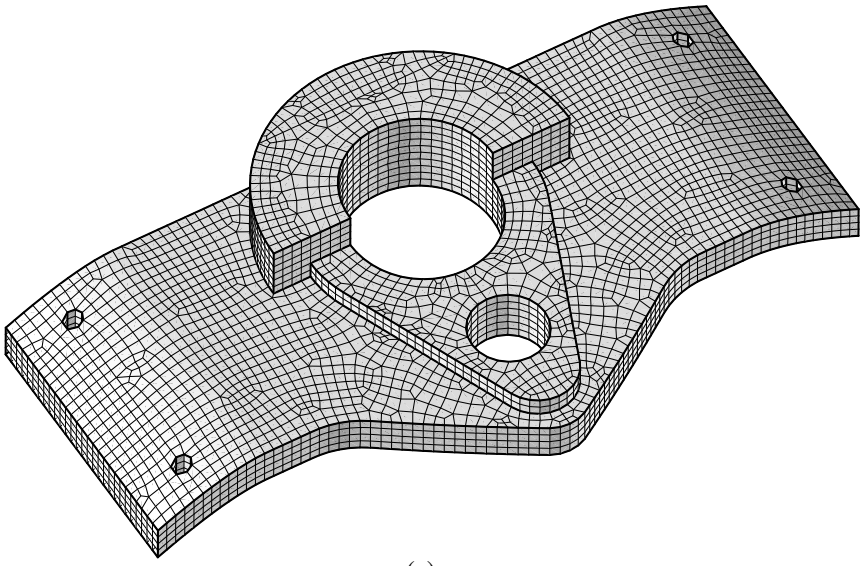
(a) (b)

Figure 11: Meshes generated using the multi-sweeping method. (a) Wall support. (b) Crankshaft.

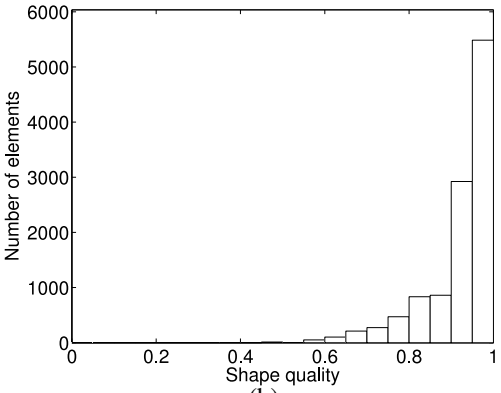
This is automatically performed by the algorithm and no user intervention is needed. Figure 11(b) presents the mesh generated on a crankshaft.

The last two example are devoted to multi-sweep geometries defined by non-planar cap surfaces. The third example presents a mesh generated for a mechanical piece in which the cap faces are almost parallel, see Figure 12(a). In this case, our method has been able to generate a high quality mesh. Figure 12(b) shows the distribution of the elements according the their shape quality [13]. Note that the minimum element quality is above 0.5.

The fourth example shows the inner nodes created during the decomposition pro-



(a)

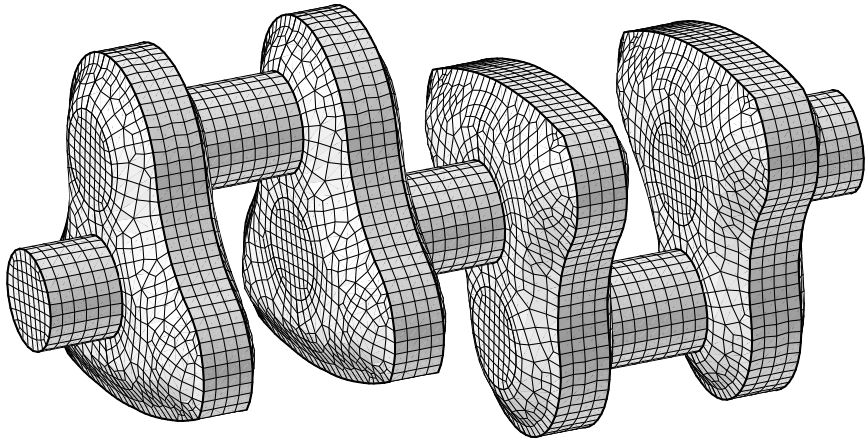


(b)

Figure 12: (a) Mesh generated for a mechanical piece. (b) Distribution of the elements according to their shape quality for the mechanical piece mesh.

cess for a crankshaft. In this example, several cap surfaces are non-parallel and non-planar. Figure 13(a) presents the mesh generated for the crankshaft. Figure 8 shows the position of inner nodes projecting only from target surfaces. Note that the levels of nodes do not follow the shape of the cap faces. Figure 8 presents the location of inner nodes calculated by the proposed method. In this case, the levels of nodes smoothly follow the shape of the cap faces.

Figure 14(a) presents the distribution of the elements according to their shape quality when inner nodes are projected only from target surfaces. Note that there is a large number of elements with quality less than 0.3. Figure 14(b) shows the distribution of the elements according to their shape quality when inner nodes are located using the presented method. In this case, we improved the minimum shape quality of the mesh.



(a)

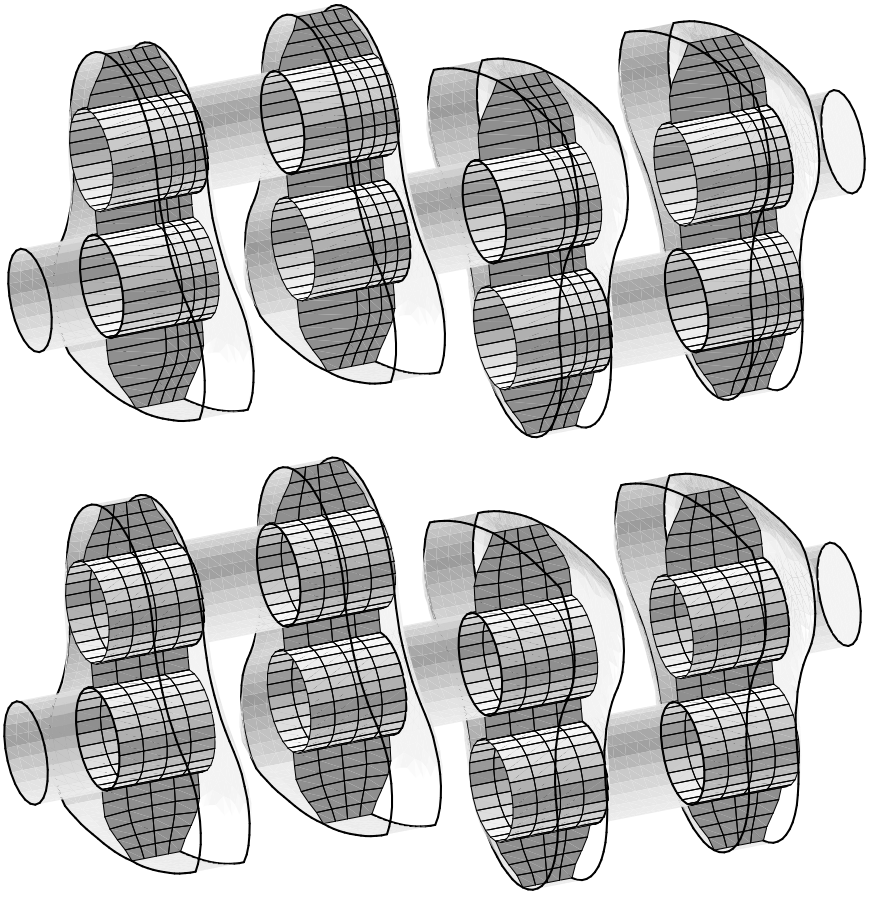


Figure 13: Location of the inner nodes computed during the decomposition process of the crankshaft. (a) General view of the final mesh. Inner nodes location when: (b) projecting nodes from target surfaces and (c) interpolating nodes projected from source and target surfaces.

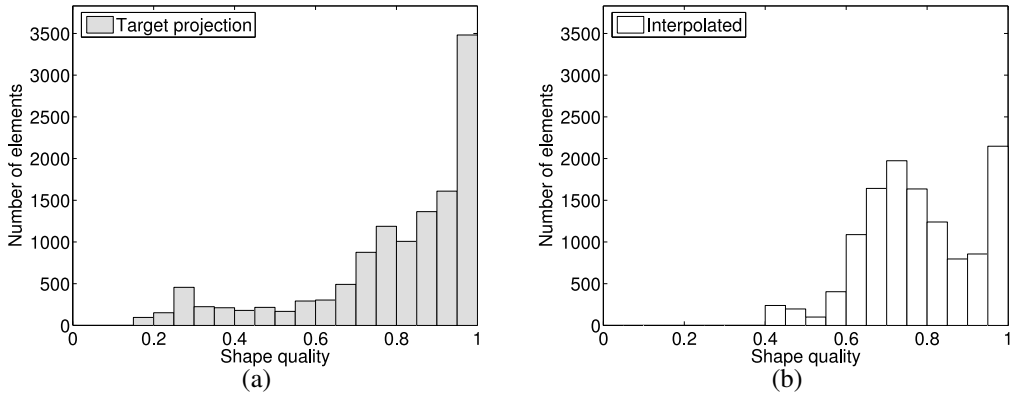


Figure 14: Distribution of the elements according to their shape quality for the crankshaft meshes. (a) Projecting nodes from target surfaces. (b) Using the proposed weighted average.

Table 1: Element quality statistics of the meshes generated for the crankshaft.

	Projecting from target surfaces	Interpolated between source and target surfaces
$\min(f_{shape})$	0.15	0.43
$\max(f_{shape})$	0.99	0.99
$\bar{f}_{shape}$	0.79	0.78
$\sigma(f_{shape})$	0.21	0.14

Table 1 presents a comparison of the statistical information of both meshes. When inner nodes are projected form target surfaces, the minimum element quality is 0.15. Instead, when using our method, the minimum element quality is 0.43. In this case, the mean value slightly decreases when our method is applied. Finally, the deviation also decreases when using the proposed method. Note that the accuracy of a simulation is determined by the minimum element quality, which is improved using our method.

9 Conclusions

In this work, we have presented the decomposition process of a multi-sweeping method. It is well known that the decomposition process influences the quality of the final mesh. On the one hand, inner nodes created during the decomposition process may induce bad quality elements or even inverted elements. On the other hand, quality of the final mesh is also affected by the robustness of the imprinting process. When the loops of nodes that define the loop faces are not planar, the imprinting process may become unstable.

In order to improve the location of inner nodes, we propose a three -stage decom-

position process. During the first stage, the nodes from target surfaces are projected to source surfaces. In the second stage, the nodes on source surfaces are projected back to target surfaces. The final position of inner nodes is computed as a weighted average between both projections.

In order to improve the robustness of the imprinting process, we introduce the new concept of computational domain. The computational domain is a planar representation of control loops. Since the computational domain is planar, the robustness of the imprinting process is enhanced. In addition, the computational domain is also used to project the nodes that come from target surfaces onto source surfaces.

It is worth to notice that the multi-sweeping algorithm is successfully implemented in the ez4u meshing environment [14].

Acknowledgements

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 and by Universitat Politècnica de Catalunya (UPC).

References

[1] P. Knupp, "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*, 1998.

[2] M.L. Staten, S.A. Canann, S.J. Owen, "BMSweep: Locating interior nodes during sweeping", *Engineering with Computers*, 15: 212–218, 1999.

[3] X. Roca, J. Sarrate, A. Huerta, "Mesh projection between parametric surfaces", *Communications in Numerical Methods in Engineering*, 22: 591–603, 2006.

[4] X. Roca, J. Sarrate, "An automatic and general least-squares projection procedure for sweep meshing", in *Proceedings of the 15th International Meshing Roundtable Birmingham*, 2006.

[5] M.A. Scott, S.E. Benzley, S.J. Owen, "Improved many-to-one sweeping", *International Journal for Numerical Methods in Engineering*, 63: 332–348, 2006.

[6] T. Blacker, "The Cooper tool", in *Proceedings of the 5th International meshing roundtable*, 1996.

[7] D.R. White, S. Saigal, S. Owen, "CCSweep: an automatic decomposition of multi-sweep volumes", *Engineering with Computers*, 20: 222–236, 2004.

[8] D.R. White, T. Tautges, "Automatic scheme selection for toolkit hex meshing", *International Journal for Numerical Methods in Engineering*, 49: 127–144, 2000.

[9] D. White, S. Saigal, "Improved imprint and merge for conformal meshing", in *Proceedings of the 11th International Meshing Roundtable*, 2002.

[10] X. Roca, J. Sarrate, "A new least-squares approximation of affine mappings for sweep algorithms", *To appear in Engineering with Computers*.

[11] J. O'Rourke, *Computational geometry in C*, Cambridge University Press, 1998.

[12] M. Berkelaar, "LP Solve", http://sourceforge.net/projects/lpsolve, 2009.

[13] P.M. Knupp, "Algebraic mesh quality metrics for unstructured initial meshes", *Finite Elements in Analysis and Design*, 39: 217–241, 2004.

[14] X. Roca, J. Sarrate, E. Ruiz-Gironés, "A graphical modeling and mesh generation environment for simulations based on boundary representation data", in *Congresso de Métodos Numéricos em Engenharia*, 2007.