





# HexBox: Interactive Box Modeling of Hexahedral Meshes

F. Zoccheddu<sup>1</sup>, E. Gobbetti<sup>2</sup> , M. Livesu<sup>3</sup> , N. Pietroni<sup>4</sup>  and G. Cherchi<sup>1</sup> 

<sup>1</sup>University of Cagliari, Italy

<sup>2</sup>CRS4, Italy

<sup>3</sup>CNR IMATI, Italy

<sup>4</sup>University of Technology Sydney, Australia



**Figure 1:** HexBox is an easy to use box modeling tool for hexahedral meshing. It can be used to create and project a conforming hexmesh onto a target geometry (as in this example), or to model a novel hexmesh from scratch, like in a pure modeling environment.

## Abstract

We introduce HexBox, an intuitive modeling method and interactive tool for creating and editing hexahedral meshes. Hexbox brings the major and widely validated surface modeling paradigm of surface box modeling into the world of hex meshing. The main idea is to allow the user to box-model a volumetric mesh by primarily modifying its surface through a set of topological and geometric operations. We support, in particular, local and global subdivision, various instantiations of extrusion, removal, and cloning of elements, the creation of non-conformal or conformal grids, as well as shape modifications through vertex positioning, including manual editing, automatic smoothing, or, eventually, projection on an externally-provided target surface. At the core of the efficient implementation of the method is the coherent maintenance, at all steps, of two parallel data structures: a hexahedral mesh representing the topology and geometry of the currently modeled shape, and a directed acyclic graph that connects operation nodes to the affected mesh hexahedra. Operations are realized by exploiting recent advancements in grid-based meshing, such as mixing of 3-refinement, 2-refinement, and face-refinement, and using templated topological bridges to enforce on-the-fly mesh conformity across pairs of adjacent elements. A direct manipulation user interface lets users control all operations. The effectiveness of our tool, released as open source to the community, is demonstrated by modeling several complex shapes hard to realize with competing tools and techniques.

## CCS Concepts

• *Computing methodologies* → *Mesh geometry models; Mesh models;*

## 1. Introduction

Hexahedral meshes are a prominent volumetric mesh representation. They are largely used as computational domains for the resolution of partial differential equations for physically based simu-

lation [SHG\*22], making them a core ingredient of many software tools used by the automobile, naval, aerospace, medical, and geological industries, as well as for many applications in computer graphics and animation.

Despite extensive research, the lack of reliable and effective automatic methods for hexahedral meshing is a well established and well documented open scientific problem [PCS\*22], up to the point that manually crafting hexahedral meshes can still be considered the industry standard [LQS17]. At the same time, interactive and semi-interactive modeling of hexahedral meshes is notoriously a difficult problem, and modeling sessions are known to be tedious and time consuming among practitioners. Prominent interactive industrial software tools [CUB23, Cor23, ANS23, Alt23] combine scripting facilities and a user interface. They are powerful and robust, but the learning curve to master these tools is overly shallow, and counter-intuitive behavior is openly reported even in the official material (Sec. 2.2). Academic research has recently investigated novel solutions that aim to sidestep the well established ones. Still, the current attempts are not mature enough or rely on intrinsically-limited modeling paradigms and solution spaces (Sec. 2.3).

In this paper, we introduce *HexBox*, a novel interactive tool for generating hexahedral meshes that brings *box modeling*, the widely validated surface modeling paradigm, into the world of hexmeshing. The choice of box modeling for hexmesh creation is the distinctive trait that differentiates HexBox from alternative systems. Existing tools implement customized interface paradigms and require specific training (Sec. 2.3). Instead, we take intrinsic advantage of the fact that box modeling is arguably one of the most common, intuitive, and best-understood 3D modeling paradigms, and is already implemented in all major 3D software packages [Vau12]. Bringing it to a novel platform (hexmeshing), we ensure that any modeler, professional or hobbyist, is already familiar with it.

Employing the box modeling paradigm for creating and editing hexahedral meshes poses, however, important challenges. First of all, tools must provide ways to control also the inner structure of the meshes and not only their surface. Moreover, and most importantly, all modifications must obey the topological constraints intrinsic of hexahedral meshing. A direct implementation of the box modeling concept for volumes is not applicable, because the so-generated meshes would contain arbitrary polyhedra that are not cuboids.

Our tool realizes fundamental box modeling tools, such as subdivision and extrusion, ensuring that they meet all required constraints, and complements them with a broad set of additional facilities that are specifically designed for hexahedral meshing (Sec. 3). This includes local and global volumetric grid refinement, element removal (useful to control mesh topology), sub-mesh cloning (to copy/paste pre-baked portions of a mesh) plus a variety of alternative tools to relocate mesh vertices, e.g., via alignment, rotation, group scaling, projection to circles, lines, or target surfaces. HexBox takes care of all the topological constraints that must be fulfilled, ensuring that at any time during modeling, the mesh contains only hexahedral elements. This also includes enforcing, on demand, mesh conformity around areas with different refinements, which we address by exploiting recent advancements in adaptive grid-based meshing [PCS\*22](§4.6). A direct manipulation user interface lets users perform modeling by applying operations to selected areas of the mesh, provide immediate feedback, and exploiting the data structures maintained during modeling to also implement undoing, redoing, and parameterized replication of actions, as well as preservation of history across editing sessions (Sec. 4).

This set of features make it possible to use HexBox as a pure *modeling* tool to create hexahedral meshes from scratch, as well as a *remeshing* tool, importing a reference surface and projecting the modeled mesh onto it (Fig. 1). In this article, and in the accompanying videos, we showcase a variety of diverse meshes created with our tool, both CAD and free-form. We strongly believe that HexBox significantly advances state of the art in interactive hexmesh modeling, and, to maximize its impact and foster more research on interactive techniques, we released its code as Open Source ([github.com/cg3hci/HexBox](https://github.com/cg3hci/HexBox)).

## 2. Related Works

Our work introduces significant advances in hexahedral mesh generation and processing. A full review of the field is out of the scope of this paper. In the following, we only focus on the methods most closely related to ours, organizing them into methods that are fully automatic and methods that enjoy some form of user interaction, both in the industry and in academia. We refer the reader to a recent survey [PCS\*22] and associated SIGGRAPH course [PCS\*23] for a broader perspective.

### 2.1. Automatic methods

The most widely-studied fully automatic approaches explored by the computer graphics community in recent years employ adaptive grids [GSP19, PLC\*21, LPC22, Mar09], polycubes [XGDC17, GSZ11, LVS\*13, FXB16, FBL16, CLS16, MCBC22, DPM\*22, GLYL20, HJS\*14], or frame fields [NRP11, LLX\*12, JHW\*13, LZC\*18, KLF16, SVB17, CC19, PBS20, BBC22]. Grid-based approaches are versatile and unconditionally robust, but lack fine control on mesh quality and are prone to creating meshes with poor geometry and topology [LPC22]. Frame-field approaches produce higher quality meshes that follow geometric features, but existing methods are brittle and easily fail even on toy-like input shapes [PCS\*22](§8.1). Polycube methods are positioned in between but are not robust enough to ensure high quality blind processing of all shapes in a *black-box* setting. Moreover, to be efficiently used as a computational domain for simulations, hexmeshes must often be locally refined according to domain specific criteria [Bla00], which automatic techniques can only indirectly control. The lack of combined robustness, mesh quality, and fine control prevents a smooth transition to the industry and ultimately motivates the still prominent use of interactive techniques in industrial practice, even though these are notoriously tedious, costly, and time consuming [LQS17].

### 2.2. Interactive tools in industrial practice

Given the lack of satisfactory automated solutions, interactive methods have been investigated since the early days of the discipline. Sandia CUBIT [CUB23] (and its industrial twin, CoreForm [Cor23]) are prominent tools that combine user interaction with a scripting language. Alternative industrial tools that rely on similar approaches are also released by other companies [Alt23, ANS23]. The principal paradigm is domain decomposition, in which the user is provided with tools to split the initial volume into smaller chunks (e.g., via cutting planes) and to mesh

each component separately. This local meshing step is highly critical, because of both topological [Mit96, Thu93, Eri13] and algorithmic constraints that often prevent the meshing of a sub-volume, even with simple geometry. In particular, mesh conformity is typically enforced by first producing a quad tessellation of the shared interfaces and then filling the interior of each cavity with hexahedra using a direct meshing strategy [LW08, MH99, TBM96, VPR19, Mit99, CSS06, KBLK14]. However, direct methods are often heuristic and/or computationally expensive, making it hard for the software to assist the user even in basic tasks, such as recognizing whether a given decomposition is already meshable or guiding early decisions during the decomposition. In other words, it is entirely up to the experience of the user to decide how to split and when to stop. Even worse, confusing situations in which the software declares its inability to mesh a geometry even though a correct mesh can actually be computed if the user forces the software to proceed anyways indeed occur, and are even reported in the official CoreForm education material (see the video [Cor22] starting at minute 14:39). For all these reasons, this modeling paradigm offers a poor interactive experience and the learning curve is extremely shallow.

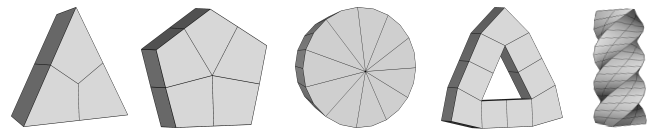
### 2.3. Research trends in interactive hex modeling

Researchers have devoted most of their efforts to the improvement of automatic techniques, exploring novel paradigms for interactive hexmesh modeling only recently [PCS\*22](§8.2).

A number of frameworks offer interactive visualization facilities to inspect the geometry and topological structure of a given hexahedral mesh [BTP\*19, XC18, NWW21]. However, they only support visualization and not editing. A limited amount of user interaction is present in methods that are predominantly automatic but either permit or require user intervention to solve small (often critical) tasks, such as manually selecting the best cutting membranes for block decomposition [LPP\*20, CZL21] or carefully positioning singular vertices inside the domain [YLZ22, YWL\*22]. These methods, however, provide limited control to the user. Skeleton-driven meshing tools such as [ZBG\*07, LMPS16, PBS22, LZLW15, ULP\*15, LAPS17, LLD12] could potentially be endowed with a user interface and employed for interactive hexmeshing, using existing skeleton modeling facilities [JLW10, BJD\*12, BMW12, BAS14, BMU\*15]. However, this class of solutions has a narrow scope because it only applies to tubular shapes.

Methods inspired by the principles of Iso Geometric Analysis [NABR15] aim to unify the representations used for design and simulation. To this end, a few attempts to higher order (spline) volume modeling have been published in recent years [ABM19, ME16, ABC\*19], but these contributions are mainly concerned with topics that do not apply to the application context that we target (linear meshing), such as smart positioning of quadrature points, blending of basis functions and solid trimming.

Takayama [Tak19] introduced a fully interactive method to manually design the dual structure of a hexahedral mesh. Users interactively create cutting membranes using implicit functions sampled at the vertices of an underlying tetrahedral mesh. Cut positioning and local bending are controlled through point handles. Profile sketching can also be used to synthesize geometrically or topologically



**Figure 2:** HexBox supports the creation of hexahedral meshes that are outside of the scope of previous fully interactive modeling approaches working in polycube space [LZS\*21], either because they contain internal singularities (three leftmost prisms with valence 3, 5, and 11), or because fitting cuboids in the user interface is problematic due to the presence of non axis-aligned topological features (triangular ring) or twisting (rightmost bar).

complex membranes. The tool operates smoothly, but has intrinsic limitations that make the approach hardly able to scale on complex shapes. Since the cutting membranes are Hermite RBF functions [MGV11], they are shape-unaware. It is therefore hard to ensure that they remain inside the object without traversing its surface tangentially, especially around narrow or highly curved regions. Perhaps most importantly, operating in the dual setting makes modeling hardly intuitive even for experts in the field, because the user is never able to see how the mesh looks like during modeling. Switching back and forth between primal and dual would alleviate this problem, but is practically unfeasible because dualization is time consuming (minutes) and introduces permanent changes to the underlying tetrahedral mesh due to the mesh refinement necessary to incorporate cuts in the connectivity. The creation of 9 models of moderate complexity took the author about 8 hours [Tak23]. Moreover, meshes are uniform, and local adaptivity is theoretically possible but hardly realizable considering the impact that transition schemes have on the dual structure.

Li et al. [LZS\*21] introduced a fully interactive polycube-based meshing pipeline in which users can create, edit, and perform Boolean operations between sets of axis aligned cuboids whose union defines a polycube hexmesh [GSZ11]. The system interactively optimizes cuboid shapes in order to closely approximate the input model, also taking care of the volume mapping that is necessary to transfer the hexmesh from polycube to object space. The system is well engineered and operates flawlessly on commodity hardware. Reported modeling times span between 6 and 21 minutes for models with moderate complexity. On the negative side, the user is restricted to operate in a parametric space (polycube space) that does not fully cover the space of all possible hexahedral meshes. For example, internal singularities are intrinsically impossible to realize with this tool, which can, at best, push external singular lines (polycube edges) inwards by means of padding [CAS\*19] or orthogonal cuts [GLYL20]. In Fig. 2 we show a few representative examples of meshes that are not replicable with this tool, either for topological or geometric reasons. Adaptive meshing is also not supported, but could in principle be incorporated in the system (e.g., using generalized adaptive refinement [PLC\*21]).

It should be noted that both these fully interactive approaches are not intended to model a shape from scratch, but rather act much more like *remeshing* tools, taking in input a tetrahedralization of the target shape and processing it to turn it into a hexmesh. Con-

versely, the tool we propose has a broader scope and can be used either for pure modeling, designing a hexmesh from scratch, or it can be given as input a reference (surface) geometry and act as a remeshing tool, using a projection operator to conform to the target domain. In both cases, it natively supports adaptivity, automatically providing the necessary topological bridges to grant mesh conformity. Mesh adaptivity is highly important in real applications, as it permits to balance complexity with simulation accuracy [WGM11, WDL\*12, SHG\*22].

### 3. Anatomy of HexBox

HexBox realizes hexmesh creation and editing through a modeling engine driven by a graphical user interface that provides control as well as immediate feedback. Since our goal is to support the familiar box-modeling paradigm for hexmeshes, we concentrate on a basic set of operations required for that purpose, i.e., refining an initial model to work at different levels of details, adding or removing parts to modify shape and topology, as well as copying regions to reuse modeling efforts. All these operations have to be adapted to maintain consistent hex meshes, and be implemented in a form suitable for integration within a direct manipulation interface that has to offer low-latency results.

At the core of the efficient implementation of our method is the coherent maintenance, at all steps, of two parallel data structures: a hexahedral mesh representing the connectivity and geometry of the currently modeled shape, and a directed acyclic graph (DAG) that links hexahedra (*element nodes*) to topological changes (*operation nodes*) such as subdivisions, face extrusions, and elements' removal. In addition to permitting the undoing of operations, the DAG makes it possible to realize all the advanced modeling features efficiently.

In the following, we will first describe topological (Sec. 3.1) and geometric (Sec. 3.2) operations that act on the mesh structure and its vertices. We will then describe how these operations relate to updates of the DAG and how the DAG can be used to implement advanced modeling actions (Sec. 3.3).

#### 3.1. Box modeling for hexahedral meshes

Constructing hexahedral meshes using a box modeling paradigm requires the implementation of features that modify the mesh connectivity, by adding, removing, or replacing elements, while maintaining strict topological requirements. To this end, we recast refinement (Sec. 3.1.1) and extrusion (Sec. 3.1.2) to the realm of hexahedral meshes, and define a new copy-paste operation to copy sequences of operations from one region of the mesh and apply them to another (Sec. 3.1.3). We also introduce element removal as a way to control the mesh topology (Sec. 3.1.4).

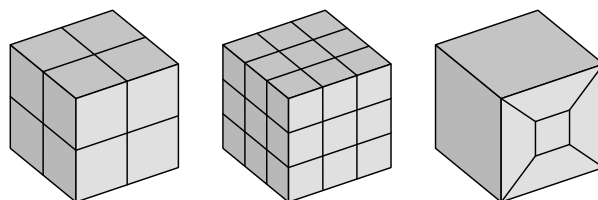
##### 3.1.1. Mesh refinement

Subdividing a hexahedron into smaller elements that occupy the same volume is a fundamental operation in box modeling, as it is required to define the resolution at which other geometric and topological operations will be performed. We grant maximum flexibility to the user, providing both global and local refinement operators

(Fig. 3). The former allows to quickly reach the base level of detail on top of which finer modeling operations will be performed, as well as to define the inner spatial resolution of the volumetric grid. The latter is useful to adapt the local mesh size to secondary smaller scale features, for example when the user desires to approximate a target shape, or to increase the resolution in a region of interest to precisely catch a physical phenomenon during simulation [WGM11, WDL\*12].

Global refinement is supported through two alternative schemes: *2-refinement*, where each edge of each hexahedron is bisected at its midpoint, creating 8 sub-elements (Fig. 3, left); and *3-refinement*, where each edge of each hexahedron is trisected, creating 27 sub-elements (Fig. 3, middle). Having two alternative schemes with different densities allows the user to control the *speed* at which mesh resolution grows more precisely, avoiding excessive refinement that is not strictly needed for modeling, especially when multiple global refinement steps are cascaded.

Local refinement is also supported through two alternative schemes: the *3-refinement* introduced above and a second local subdivision option that we call *face refinement*, which allows to locally edit the singular structure of the mesh, splitting a hexahedron into six sub-elements so as to decompose the selected face into five quads, one central and four lateral (Fig. 3, right). This type of split, introduced in [ISS09], is particularly useful to prepare the mesh for a subsequent extrusion of the central element because the lateral elements protect it, acting as a padding layer. Practical occurrences of face refinement are shown in our results (Sec. 5). We specifically chose, instead, to not support the *2-refinement* for local subdivision because of its peculiar effects on the handling of mesh conformity during editing operations, as we will detail in the following paragraphs.



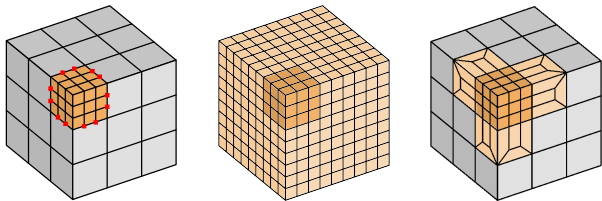
**Figure 3:** The three refinement operations we provide in our tool to manage the resolution of the modeled shape. On the left, a single hexahedron is split into 8 new hexahedra, while in the middle into 27 new hexahedra. On the right, we split a single face of a hexahedron, introducing 6 new elements.

The application of local subdivision operators introduces *hanging nodes*, i.e., new vertices that lie on edges or faces of the mesh, creating a non-conforming topology (e.g., the red vertices in Fig. 4, left). The treatment of hanging nodes is a widely studied topic in the literature, both from a mesh generation standpoint and from the point of view of the numerical solvers. HexBox strives for maximum flexibility, permitting the user to create both non-conforming and conforming meshes. Non-conforming meshes are, in fact, directly supported by numerical techniques such as Discontinuous Galerkin [CWM\*16], albeit at the cost of a more complex mathematical formulation and implementation with respect to conformal

solvers. On the other hand, hanging nodes could also be suppressed by substituting the hexahedra adjacent to refined cells with generic polytopal cells, but also this choice would produce meshes that require a specialized polytopal solver, such as VEM [BdVBM14]. Suppressing hanging nodes while retaining an all-hex topology is by far the solution that maximizes the compatibility of the mesh with the plethora of existing numerical solvers.

Suppression of hanging nodes was mostly studied in the context of adaptive grid-based hexmeshing [PCS\*23](§4.6), where conformity must be restored locally, without affecting the whole mesh (Fig. 4, middle). This is achieved using dedicated *transition schemes* that allow hanging nodes to be incorporated within the mesh structure (Fig. 4, right).

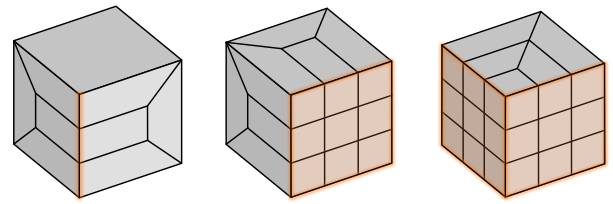
In general, methods to suppress hanging nodes in 2-refined grids are considered superior to the ones that suppress nodes generated by 3-refinement, because all possible cases can be solved with closed-form solutions and the mesh growth is lower [LPC22]. However, these methods impose non-local pairing constraints on the refined patterns [PLC\*21] and, most importantly, require mesh dualization. While this is not an issue for automatic methods, modeling a hexmesh interactively by operating in the dual space is extremely complex and costly, as already demonstrated by previous attempts [Tak19]. For this reason, in HexBox we decided to prioritize speed and ease of modeling over a slight advantage in terms of mesh resolution, forbidding the use of local 2-refinement operations.



**Figure 4:** Local refinement of a single hexahedron (left) yields a non conformal mesh containing hanging nodes (in red). Conformity can be easily restored by globally propagating the refinement to all mesh elements (middle). Alternatively, the transition schemes listed in Fig. 5 allow to obtain a much coarser conforming mesh (right). Pale orange areas denote the extent of the transition region used to restore mesh conformity.

Fig. 5 illustrates the three schemes that we use to suppress hanging nodes around 3-refined areas, introduced in [ISS09]. Substituting hexahedral cells that are adjacent to the refined area along a convex edge (left), a face (middle) or a concave edge (right), the associated hanging nodes are incorporated into a conforming mesh. A closed form transition scheme for elements that are adjacent to a refined area through a concave corner is, however, impossible to devise. This is a known limitation of 3-refinement approaches, and is faced by iteratively applying the base scheme in Fig. 3 (middle) and those in Fig. 5, until reaching a configuration that can be treated with the available schemes [PCS\*22](§4.6).

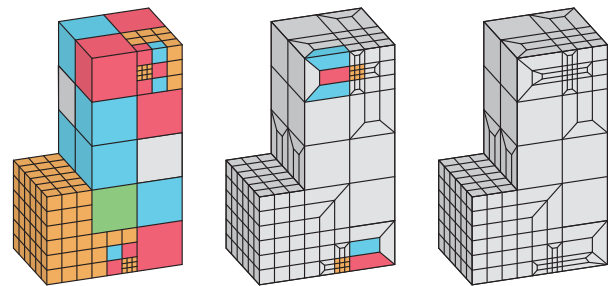
The overall process that iteratively applies the transition schemes and removes the hanging nodes, bringing the mesh into a state of



**Figure 5:** The three transition schemes we use to restore mesh conformity around refined areas. These schemes remove hanging nodes from hexahedra with: a single subdivided edge (left), a single subdivided face (middle), and two subdivided adjacent faces (right).

conformity is called *makeConforming*. A graphical illustration of the *makeConforming* operation is described in Fig. 6.

While it would be possible to automatically apply it after each local refinement operation, this could prevent the user to obtain the desired result because *makeConforming* would likely install transition schemes into elements that the user might desire to further split with 3-refinement. For this reason, we support the editing of not conforming meshes and let the user decide when to apply *makeConforming* by calling it from the interface.

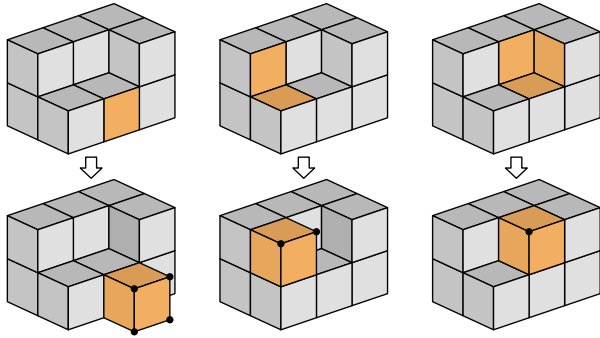


**Figure 6:** The *makeConforming* operation incorporates the hanging nodes into the structures, restoring the conformity of the mesh. Starting from the hexahedra refined by the user (orange), mesh conformity is obtained by installing the transition schemes listed in Fig. 5 in the adjacent elements. In this example, the blue elements are replaced with the schemes in Fig. 5 left, the red elements with the one in Fig. 5 middle, and the green elements with the schema in Fig. 5 right. More than one iteration of the *makeConforming* can be required in order to reach the conformity of the mesh (mesh on the right).

### 3.1.2. Extrusion

While subdivision creates new elements to occupy the same volume, the user can cover new volume by extruding one or more surface faces, producing a new element adjacent to them. The extrusion action must combine a topological operation, which simply consists in adding the new element to the grid, linking it to the correct face, edges, and vertices, with geometric operations that need to assign plausible positions to the new vertices. In the simplest case, i.e., when the extrusion starts from a single face (Fig. 7, left), our tool creates the vertices of the new hexahedron in the direction given by the normal of the selected face at a distance equal to

the average length of its edges. If the extrusion originates from two faces sharing the same edge (Fig. 7, middle) or three faces sharing the same vertex (Fig. 7, right), both forming concave configurations, the vertices of the new hexahedron are generated in the direction of the average of the normals of the selected faces. Clearly, the newly created elements, as well as all the other elements of the mesh, can be subsequently moved in the modeling space as described in Sec. 3.2.



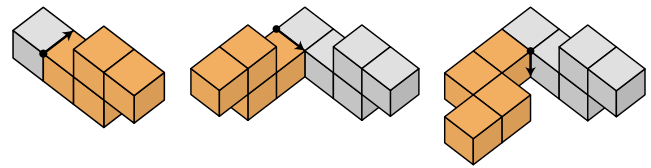
**Figure 7:** We implement the addition of new elements via face extrusion, handling three possible configurations. On the left, a new hexahedron is created by extruding a single face; in the middle, a new hexahedron is created starting from two adjacent faces, while on the right, the new element is created through the extrusion of three adjacent faces. New elements are highlighted in orange, while the new vertices are depicted as black circles.

### 3.1.3. Copy and paste

Since many real-world objects include multiple approximately similar parts, it is important to support the creation of mesh portions by replication in other areas of an already modeled shape (Fig. 8). Think, for example, of gear teeth, branches, protuberances, arms or legs, and other protrusions. For this reason, we directly support the copy and paste of mesh portions that have been created by repetitive extrusion (Sec. 3.1.2). The user can thus select an extruded mesh portion (Fig. 8, left), create a cloned version and connect it to other faces of the modeled mesh (Fig. 8, middle). The user can choose one, two or three connected faces on which to paste the copied branch, as if the new portion had been initially generated in place by face extrusion. Once the new portion has been pasted on the mesh, the user can change the orientation, as shown in Fig. 8, right. Clearly, the position of the vertices of the elements composing the cloned portion can then be varied subsequently, as for all the other elements of the mesh, as described in Sec. 3.2.

### 3.1.4. Hexahedra removal

While the previous operations increase the complexity of the modeled mesh, hexahedra removal reduces the element count and makes it possible to obtain cavities or grooves by digging from the surface toward the inside of the model. It also permits the creation of models with any genus by creating holes. The removal operation simply requires the selection of the elements to be removed and the update of the mesh structure.



**Figure 8:** HexBox allows the user to copy and paste portions of the modeled mesh. A branch is selected (left), cloned, and pasted on another mesh area (middle). The user can then rotate the cloned branch as desired (right).

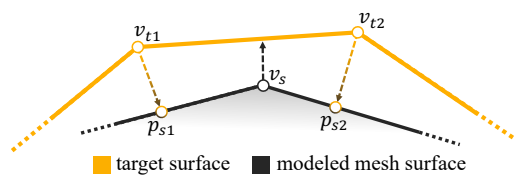
In principle, objects with a genus greater than zero may also be created by connecting elements coming from different portions of the surface (e.g., creating a torus by connecting the two ends of a deformed cylinder). The connection of non-adjacent portions of a locally-refined hexmesh, however, is very complex except in the simplest cases, since the system would need to handle the connection of parts with differing refinement levels and different geometric shapes. For this reason, we do not support this operation, and we offer the users tools to create holes by deleting elements from the mesh structure. The idea is that the user adopts a coarse-to-fine approach during the modeling, first creating the basic structure of the shape and then enriching it with the desired details. Once the expected connectivity has been obtained through combinations of extrusion and subdivision, as explained in the previous sections, one or more elements can be removed in order to change the genus of the modeled shape. The effectiveness of this restricted approach for creating complex shapes is illustrated in our results (Sec. 5).

## 3.2. Purely-geometric operations

In addition to creating the mesh connectivity and topology, while immediately obtaining a starting 3D shape through the assignment of default geometric properties, HexBox allows the user to modify the geometry by moving vertices in 3D, individually or in groups. At any stage of the modeling pipeline, the user can perform these operations through an (assisted) direct manipulation or by applying automatic methods. We support a broad variety of basic geometric techniques, based on rotations, scalings and projections to circles, spheres, planes and lines. Despite trivial from a geometry processing standpoint, these tools heavily contribute making HexBox an easy-to-use modeling tool, and we encourage the reader to watch the accompanying videos to assess the impact that these facilities have on the modeling experience. In the following, we concentrate, instead, on the features that allow users to employ the tool for the approximation of a target surface, i.e., projection and sharp feature handling.

### 3.2.1. Projection to a target surface

When HexBox is used as a remeshing tool, projection is used to displace the vertices of the hexmesh so as to match the input target geometry. Our projection algorithm takes inspiration from the quad reprojection method of QuadWild [PNA\*21], in the sense that it uses an *inverse* projection operator to approximate the target geometry well. In short, rather than projecting the hexmesh vertices directly to the target geometry, we do the opposite, projecting the



**Figure 9:** An example of the projection system implemented in HexBox. The vertex  $v_s$  is attracted to the target mesh in the direction computed as the weighted sum of the vectors  $p_{s1} - v_{t1}$  and  $p_{s2} - v_{t2}$ , where the  $v_t$  are the vertices of the target mesh,  $p_s$  their projection points on the hexmesh surface and  $v_s$  is the hexmesh vertex we want to move forward towards the target shape.

target geometry to the modelled shape and using the so generated vectors to devise per-vertex displacements for the hexmesh surface points. The main intuition behind this approach is that, in a direct projection setup, portions of the target geometry that are behind narrow passages are seldom reached by projecting rays. With our inverse approach, ray-inaccessible regions still project to the hexmesh geometry, providing the necessary attraction to permit the hexmesh to *sneak in*, ensuring higher geometric fidelity.

To convert projections from the target geometry to the hexmesh into vertex displacements we proceed as follows: for each surface vertex of the hexmesh, we compute a displacement vector that is defined as a weighted sum of all vectors that project a vertex of the target mesh onto a quad face incident to it (Fig. 9). For this step, we take advantage of the HexBox interface (Sec. 4), permitting the user to select the weighting scheme that produces the best result. Various ways to combine such vectors are provided in the system, such as the length of projection vectors, the inverse distance between the base point of each projection vector and the vertex to relocate, quadrilateral barycentric coordinates [Flo15], or a combination of those. In case no vertex of the target mesh projects onto the quads incident to a hexmesh surface vertex, we move it by means of Laplacian smoothing, thus accommodating the motion of its neighbors.

Left alone, the surface projection described above can easily generate flipped elements. Surface projection is therefore coupled with a displacement of interior vertices, which are required to follow and accommodate the evolution of the surface. Internal vertices are relocated by means of a one or more steps of Laplacian smoothing, the amount of smoothing being controllable by the user. In order to forbid the inversion of mesh elements, all vertex relocations, both on the surface and internal, are executed with a line search algorithm, iteratively halving the displacement vector if the Jacobian of any of the incident hexahedra becomes negative. We also provide the possibility of applying a *padding* step [MT95] near the surface of the modeled shape, in order to increase the degrees of freedom with which the projection algorithm can work.

While this approach is fully robust, in the sense that it strictly guarantees the generation of a valid simulation mesh with all positive Jacobians, it may generate poor approximations of the target shape because of the line search stop criterion. In some cases, the user may disable the line search filtering and proceed by a free projection, obtaining higher geometric fidelity at the cost of looser

guarantees on mesh quality (see the attached video of the modeling of the Cow in Fig. 1).

We emphasize that our simplified projection strategy is not aimed to overcome more robust and convoluted methods such as [GSP19, LZS\*21], which could also be incorporated in our system thanks to its modular design. In particular, when used without explicit checks on the Jacobians, our tool does not provide guarantees of correctness and may likely introduce degenerate or inverted elements during projection. When this happens, we rely on an off-the-shelf untangling algorithm to repair the broken elements [LSVT15].

### 3.2.2. Handling sharp features

CAD geometries are rich of sharp features that must be precisely encoded in the mesh for a faithful simulation (e.g., in fluid dynamics). HexBox fully supports sharp feature preservation by allowing users to select edge chains on the target surface and corresponding edge chains on the surface of the modeled mesh, as detailed in Sec. 4. These correspondences are then used as hard constraints in the projection and smoothing processes. Note that this interactive selection of sharp linear features allows the user to select which characteristics are required to be preserved on a case-by-case basis, permitting to handle also non geometric features that may be relevant for simulation, such as separation between materials with different physical properties. For the pure geometric part, we also support automatic feature detection via dihedral angle thresholding. Users can of course combine the two strategies, starting from an imperfect feature network computed automatically and improving it interactively. In Fig. 13, we included several models (Block, Cylinders2, CAD1, CAD5, CAD7, CAD8, Knob) in which sharp features preservation was a key ingredient.

### 3.3. Modeling with a DAG

In HexBox, all the topological and mesh connectivity operations that the user performs during the modeling process are stored in a Directed Acyclic Graph (DAG). A similar approach was taken by Face Extrusion Quad Meshes [PBS22] for supporting interactive editing of quad meshes. Our DAG comprises *element nodes* corresponding to the mesh hexahedra, and *operation nodes* encoding the topological operations of hexahedra subdivision, face extrusion, and element removal. In addition, to also cover geometric modifications, the position of the vertices is stored in an auxiliary vector, which is updated and stored at each modeling step together with the DAG. The union of the DAG and the vertex position vector allows us to reconstruct the whole history of the modeling session, starting from the initial cube and keeping track of every operation.

Our DAG representation has a single root, which is, in all cases, an element node (i.e., the initial cube). Going deeper into the hierarchy, element nodes are children of an operation node if they have been generated as a result of that operation. Conversely, operation nodes are children of an element node if the operations have been applied to that element. Logically, no arcs are allowed between two element nodes or two operation nodes.

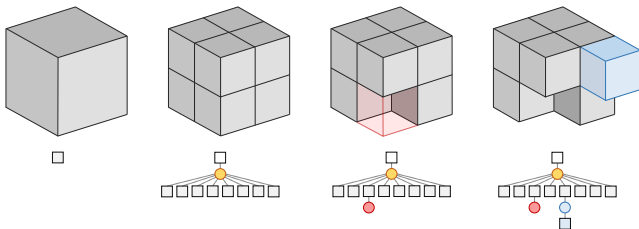
It is essential to underline that the operation nodes do not have a one-to-one correspondence with the primitive modeling actions

described in Sec. 3.1 and Sec. 3.2. Indeed, a single mesh structure modification action performed by the user can generate multiple element nodes and operation nodes, while a single geometric action, such as vertex moving, projection, or smoothing, does not generate nodes but only acts on the vertex position vector. Also, note that not every element node in the DAG necessarily corresponds to a hexahedron in the mesh at the current state of the modeling. In fact, all the elements that have been part of the mesh at some step of the modeling are stored in the DAG, including those that have been removed or subdivided and thus replaced by smaller hexahedra (which correspond to their descendants in the DAG).

Having the modeling history stored in the DAG simplifies the efficient implementation of several modeling features. For example, cloning a portion of the mesh generated by extrusion starting from a hexahedron  $h$  simply copies the sub-graph with  $h$  as a root and adds it as a child of the node representing the hexahedron in which we want to replicate the cloned shape. We can then traverse the sub-graph and apply all operations relative to the new root to update the current mesh. Actually, since the results of operations are cached with the nodes, we do not redo the operation from scratch, but directly reuse the cached mesh structure of the subgraph, relocating it by suitably transforming the vertices.

Another operation in which the DAG is crucial is undoing the last performed operations and restoring the modeling process to a previous state. With the DAG, we can go up the hierarchy towards the root, undoing all operations one by one (i.e., removing from the mesh the hex cells corresponding to the created element nodes or adding back the removed ones).

Finally, storing the modeling in the DAG also allows us to save the state of the modeling and resume it in a later session, always maintaining the possibility of moving forward and backward in the list of the performed operations (starting from the initial cube).



**Figure 10:** A modeling session with all the performed operations stored in a DAG. On the top line, the mesh at each step of the modeling pipeline. On the bottom, the DAG with different elements and operation nodes (yellow = subdivision, red = removal, blue = extrusion).

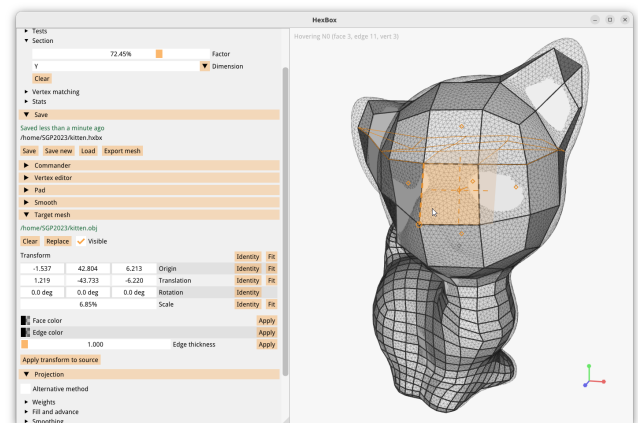
For clarity of exposition, Fig. 10 illustrates how a simple modeling session is stored in our DAG structure. We start from the single cube stored in the root of the DAG. We split the cube with a 2-refinement subdivision, and this action is stored in the DAG with an operation node of type "subdivision" (yellow circle) having the root as a parent and a child node for every generated new hexahedra (gray squares). Then a hexahedron is removed, and this action is encoded by adding an operation node of type "removal" as a child

of the node representing the removed hexahedron (red circle). Finally, we create a new hexahedron via extrusion of a face of an existing hexahedron, and this consists in adding an operation node of type "extrusion" as a child of the selected hexahedron (blue circle) and an element node as a child of the just added extrusion node (blue square).

#### 4. User interface

All the operations described in the previous section are made available to the user through a Graphical User Interface (GUI) that aims to cover modeling and remeshing features.

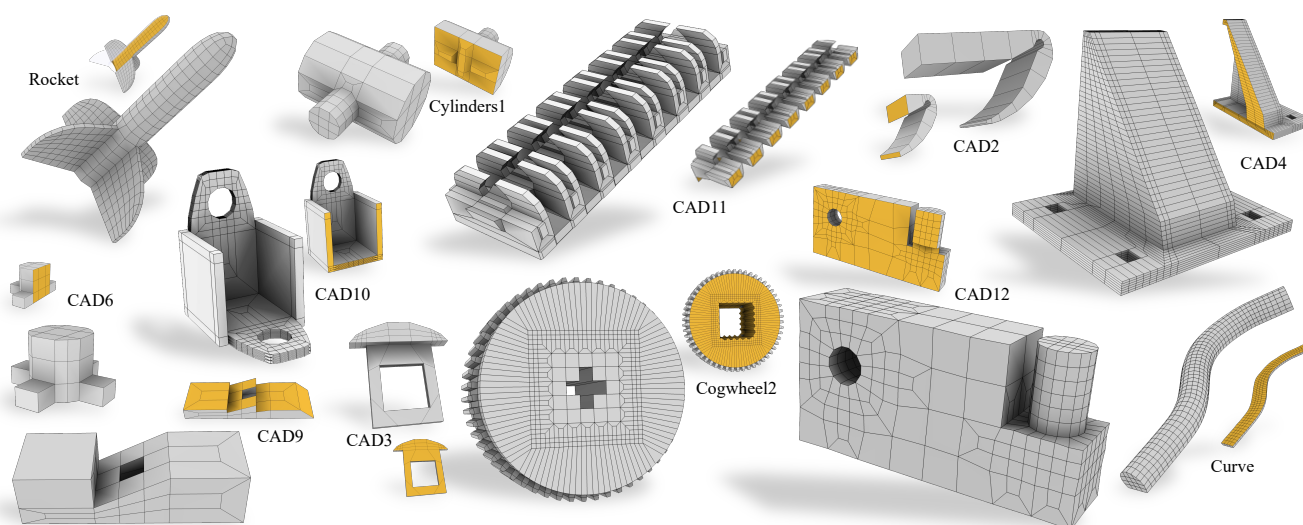
The HexBox GUI has a window divided into two parts: a main canvas, where a 3D view of the mesh is displayed throughout the modeling process, and a side panel summarizing the available modeling tools and their parameters (see Fig. 11). The side panel also shows the list of actions performed during the editing session. In this list, we distinguish the actions applied and those canceled with green and red colors, and we provide a brief informative human-readable description of the actions. For the user, restoring the modeling to a previous state is always possible. Both viewing and restoring modeling actions are possible thanks to the GUI access to the DAG, which maintains the modeling state. This possibility is available across editing sessions since we can store and restore the entire state on a file by serializing and deserializing the mesh and DAG structures.



**Figure 11:** The GUI of HexBox comprises a main canvas, in which the modeled shape is displayed, and a side panel with the modeling tools. This screenshot shows the interface in light mode, but according to user preferences, the whole application can be set in dark mode.

As illustrated in the accompanying videos, for modeling a new shape, the application opens with a single cube displayed on the main canvas. If desired, the user can load a reference mesh to be used as a guide during the modeling process and possibly as a target mesh for the projection operation. This mesh is displayed in semi-transparency, overlapped with the modeled mesh, and can be shown or hidden as desired. A trackball interface is used to manipulate the view and see the model from different angles.





**Figure 12:** Gallery of models created from scratch using HexBox.

Editing actions are activated in the side panel or through keyboard shortcuts. Global modifications are applied to the entire model (e.g., 2-refinement of the entire grid), while local topological or geometric modifications require the prior selection of the affected elements. HexBox streamlines selection by applying a context-dependent and mouse-position-dependent process that strives to understand which element the user wishes to select. When a single element is required for the operation, visual feedback while the mouse is hovering highlights the currently selected edge, vertex, and face. These are the face under the cursor, and its closest edge and vertex.

For the extrusion operation, the user can select whether to extrude from the current face, edge, or vertex. If a face is selected, the element grows from it. If an edge is selected, the extrusion occurs from the two adjacent faces. When selecting a vertex, the extrusion occurs from the three adjacent faces, as described in Sec. 3.1.2. For the 3-refinement of one or more hexahedra, selecting one or more elements to apply the refinement is sufficient. Face selection is obtained by clicking on the target face during element selection. Once the refinement has been applied, the *makeConforming* operation can be optionally executed to restore the conformity of the mesh structure. The copy-paste action involves selecting an element and marking it as copied, then paste it onto one, two, or three selected faces as described for the extrusion. The local configuration of the elements in the destination area determines the orientation of the cloned portion in its new location. The user can rotate it and move its vertices in space as desired. The removing operation simply consists of selecting one or more elements and then deleting them. In the case of vertex or edge selection, all adjacent elements are removed.

In addition to selecting single elements, HexBox allows multiple selections of entire mesh areas. In particular, it is possible to select or deselect all vertices in a viewed area by drawing a rectangle with the mouse. In this case, all the vertices that project within the user-bounded area are selected. More complex selections can

be obtained by adding or subtracting the set of vertices created by different rectangle selections. Note that also inner vertices may be selected with this operation, allowing the tool to apply operations also on inner elements.

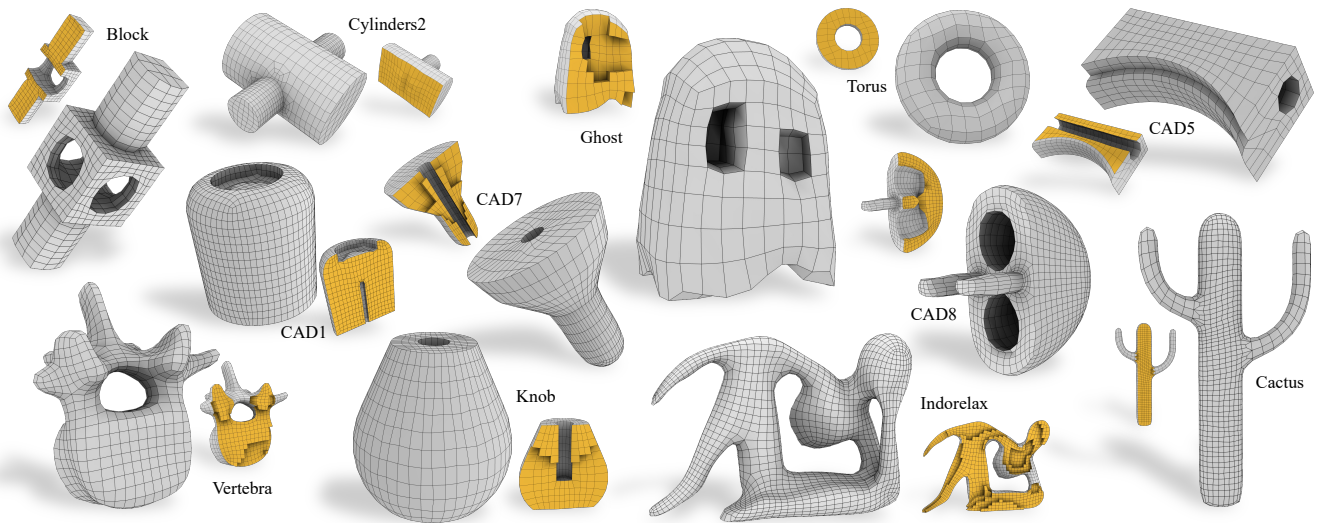
The group selection operation is essential when the user wants to move entire clusters of vertices in space. Once the vertices have been selected, with a combination of tools on the side panel of the GUI or keyboard shortcuts, it is possible to translate, rotate or scale the elements marked by the selected vertices.

Keyboard shortcuts allow users to lock transformations along one or more axes, where applicable. For example, it is possible to scale only on two axes out of three, rotate on a single axis, or translate along a single axis. The GUI provides a real-time preview of the applied transformation, which give the user immediate feedback. The real transformation is only applied when the user confirms it. Finally, for finer tweaks, the user can manually assign the transformation values by typing values or dragging sliders in the side panel.

As mentioned in Sec. 3.2.2, we offer the possibility of marking sharp features through the selection of mesh and target edge chains. The pairs of chains in the target mesh and in the modeled mesh are displayed with matching colors overlaid over the respective meshes. The user can decide whether to view all the pairs of chains together or view them individually. Once satisfied with the selection, the user can start the projection or smoothing processes and go back to edit the selected features if necessary.

## 5. Discussion

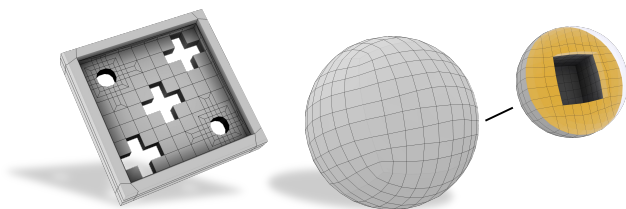
We implemented HexBox in C++, using Cinolib [Liv19] for volume mesh processing and Dear ImGui [Cor] for the user interface. To maximize the impact of our research, we publicly released the source code ([github.com/cg3hci/HexBox](https://github.com/cg3hci/HexBox)) and uploaded output models in the HexaLab online database [BTP\*19].



**Figure 13:** Gallery of models created using HexBox as a remeshing tool, i.e., modeling a hexmesh and projecting its skin onto a target surface mesh.

We also emphasize that HexBox is expected to be a *live* project. While this article describes the framework as it is today, we plan to introduce new features and ameliorate the current ones with future works, hopefully with the contribution of the hexmesh community (more on this in Sec. 6).

In the following, we briefly discuss the main characteristics and outcomes of the current design and implementation. A series of accompanying videos illustrate the behavior of the system through live captures of modeling sessions. A representative set of created models is presented in Fig. 12 and Fig. 13, and detailed in Tab. 1.



**Figure 14:** HexBox allows users to model a wide range of mesh topologies, regardless of the number of handles and inner cavities. The Tic-tac-toe model has genus 5. The Hollow-sphere model has one cavity.

**Shape Space.** HexBox is a quite versatile tool, which allows to fully cover the range of existing hexmesh topologies, both at a local and at a global level. Specifically, HexBox users can create hexmeshes with any genus (Fig. 14) and number of connected components (Fig. 15), and also have full control on the singular structure, creating singular edges of any valence, in the interior and on the surface (Fig. 2). Alternative techniques, both interactive and automatic, do not exhibit the same flexibility: frame field methods are

more versatile and permit internal singularities, but only with restricted valence [PBS20]. Grid-based methods are fully automatic and robust, but can hardly control mesh connectivity, which inevitably does not align with the object curvature and also contains poor shaped elements around high valence edges [LPC22]. Polycube methods do not permit internal singularities and, as the authors of [LZS\*21] confirmed, interactive polycube modeling does not also permit to create simple models like the ones shown in Fig. 2, either because they contain unsupported singularities or because of the presence of twisting and non axis aligned components for which fitting a cube is problematic with their user interface. We emphasize that even though HexBox covers all possible mesh topologies and edge valences, this does not necessarily mean that any mesh can be created *with ease*. In particular, the sequence of operations necessary to reproduce the wanted mesh structure may be non intuitive in a box-modeling workflow or may be hindered by the user interface.

**Singularity alignment.** Having full operative control on the mesh geometry and topology enables the generation of high quality hexmeshes that endow a coarse block decomposition. In particular, operating in a coarse to fine manner through a sequence of splitting and extrusions, the box modeling paradigm intrinsically ensures that singular vertices align, permitting even to non expert users to easily create high quality mesh connectivities (Fig. 16). Creating block-structured meshes with controlled singular structure is relevant both for shape compression [Tau04] and for higher order methods for the resolution of PDEs. These methods rely on a finite number of templated solutions to lay basis functions over local clusters of adjacent elements [WZT\*18]. They are therefore mostly incompatible with automatic methods, which do not offer direct control on the mesh connectivity.

**Performance.** Supporting an interactive modeling workflow requires the application to provide high-frequency feedback for direct

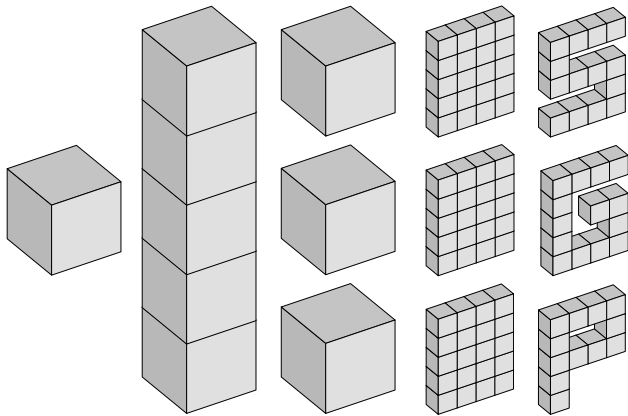
Model	#H	G	#B	#Sing V/E	Edge Val	SJ (min/max/avg)	Feat	Mod/Rem	Time
Block	1584	3	100	96 / 384	2 / 3 / 4 / 5	0.18 / 0.99 / 0.72	yes	rem	4:00
Cactus	2136	0	202	82 / 488	1 / 2 / 3 / 4 / 6	0.07 / 0.99 / 0.85	-	rem	2:25
CAD1	5564	0	196	52 / 372	2 / 3 / 4 / 5 / 6	0.10 / 0.99 / 0.92	yes	rem	4:00
CAD2	17	0	1	8 / 76	1 / 2	0.58 / 1.00 / 0.91	yes	mod	2:30
CAD3	186	1	18	24 / 122	1 / 2 / 3 / 4	0.02 / 1.00 / 0.77	yes	mod	1:30
CAD4	2704	4	99	68 / 708	1 / 2 / 3 / 4	0.07 / 1.00 / 0.80	yes	mod	4:00
CAD5	720	1	10	20 / 252	1 / 2 / 3 / 4	0.06 / 0.99 / 0.91	yes	rem	5:00
CAD6	31	0	22	40 / 76	1 / 2 / 3 / 4	0.30 / 1.00 / 0.73	yes	mod	1:00
CAD7	1536	1	8	16 / 224	1 / 2 / 3 / 4	0.01 / 0.99 / 0.77	yes	rem	3:15
CAD8	5120	1	164	92 / 572	2 / 3 / 4 / 5 / 6	0.04 / 0.99 / 0.74	yes	rem	6:00
CAD9	55	1	46	40 / 82	1 / 2 / 3 / 4 / 5	0.19 / 1.00 / 0.74	yes	mod	0:45
CAD10	1253	2	929	334 / 828	1 / 2 / 3 / 4 / 5 / 6 / 8	0.00 / 1.00 / 0.73	yes	mod	2:15
CAD11	274	0	218	568 / 1340	1 / 2 / 3 / 4	0.20 / 1.00 / 0.73	yes	mod	6:00
CAD12	750	1	598	226 / 541	1 / 2 / 3 / 4 / 5 / 6	0.01 / 0.99 / 0.76	yes	mod	2:00
Cogwheel1 (Fig. 17)	688	1	256	264 / 772	1 / 2 / 3 / 4	0.20 / 1.00 / 0.85	yes	mod	2:42
Cogwheel2	1320	1	1266	632 / 1460	1 / 2 / 3 / 4 / 5 / 6 / 7	0.00 / 1.00 / 0.91	yes	mod	4:30
Cow (Fig. 1)	3424	0	116	128 / 564	2 / 3 / 4 / 5 / 6	0.45 / 0.99 / 0.84	-	rem	4:56
Curve	832	0	1	8 / 240	1 / 2	0.30 / 0.99 / 0.89	yes	mod	4:00
Cylinders1	137	0	101	104 / 236	1 / 2 / 3 / 4 / 5	0.23 / 1.00 / 0.63	yes	mod	2:15
Cylinders2	1960	0	119	104 / 568	1 / 2 / 3 / 4 / 5	0.15 / 0.99 / 0.81	yes	rem	2:15
Ghost	490	0	55	25 / 148	1 / 2 / 3 / 4	0.43 / 0.99 / 0.88	-	rem	2:00
Hollow-sphere (Fig. 14)	604	-	26	16 / 168	1 / 2 / 3 / 4	0.13 / 0.99 / 0.80	-	rem	1:30
Indorelax	6176	2	133	92 / 706	2 / 3 / 4 / 5 / 6	0.43 / 0.99 / 0.84	-	rem	9:00
Knob	1600	0	17	16 / 208	1 / 2 / 3 / 4	0.08 / 0.99 / 0.89	yes	rem	4:20
Rocket	385	0	22	40 / 340	1 / 2 / 3 / 4	0.15 / 0.99 / 0.83	yes	mod	7:00
SGP (Fig. 15)	40	-	27	68 / 202	1 / 2 / 3	1.00 / 1.00 / 1.00	yes	mod	0:40
Tic-tac-toe (Fig. 14)	5688	5	5688	1466 / 4840	1 / 2 / 3 / 4 / 5 / 6 / 7 / 8	0.00 / 1.00 / 0.68	yes	mod	3:45
Torus	192	1	8	16 / 128	1 / 2 / 3 / 4	0.08 / 0.97 / 0.66	-	rem	2:15
Vertebra	2601	1	118	130 / 563	2 / 3 / 4 / 5 / 6	0.27 / 0.99 / 0.86	-	rem	5:00

**Table 1:** For each model we report: number of hexahedra (**#H**), genus (**G**), number of block domains (**#B**), number of singular vertices/edges (**#Sing V/E**), the list of edge valences (**Edge Val**), scaled Jacobians (**SJ**), presence of sharp features (**Feat**), modeling or remeshing strategy (**Mod/Rem**), and modeling time, in minutes (**Time**). Meshes created in a modeling session are shown in Fig. 12. Meshes created in a remeshing session are shown in Fig. 13. For meshes that do not appear in any of these mosaic images, we specify the image were they are shown.

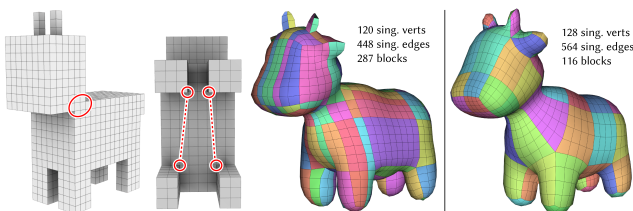
manipulation actions, as well as minimize the latency of processing operations. During normal operations (selection, viewing), the method renders a static model cached in graphical memory. On a standard PC (intel i5-13600k CPU, 32GB DDR4, integrated graphics), we verified that a model with over 7M faces can be rendered at over 130 Hz during viewing and selection. Actions that only modify geometry by moving vertices (Sec. 3.2) add little overhead, and are generally sufficiently fast to be done with direct manipulation (consistently over 30Hz). In the current implementation, the most costly actions are those that add/remove elements (Sec. 3.1), whose cost heavily depends not only on the number of primitives touched, but is dependent on the global mesh size, since we update the full graphical representation each time the mesh is modified. We can easily remove this limitation by implementing partial updates of the graphical representation. In any case, these operations are not performed inside a direct manipulation action, but are triggered by the user that can wait for their completion. In most practical cases, the latency is small enough to provide the illusion of immediate completion. The *MakeConforming* operation, for instance, takes 61ms on the benchmark PC for going from 11K elements to 16K elements to fix a mesh where a large number of faces have interfaces

with three levels of 3-refinement of difference (i.e., a face in an element connected to  $9^3$  faces in the neighbor element). Significantly growing the mesh size leads however to larger delays. For instance, it may take over one minute to ensure conformance of an 8M-elements mesh if we increase the refinement-level difference to four (one face matched with  $9^4$  faces). The latter case, however, is very extreme, and we may still avoid blocking the application by launching refinement in a separate thread. It should be noted, however, that the manually modeled hexahedral meshes are typically much smaller than our extreme case. For instance, the HexaLab database [BTP\*19] currently contains 908 unique meshes with an average element count of 30K (maximum 590K).

**Modeling with HexBox.** Fig. 12 shows a variety of hexahedral meshes that were generated from scratch with our tool. Screen captures of the modeling sessions in the additional material. Mesh features and modeling time are conveniently reported in Tab. 1. Overall, these models showcase a large variety of mesh genera, sharp features and singular types, practically confirming the breadth of our modeling space.

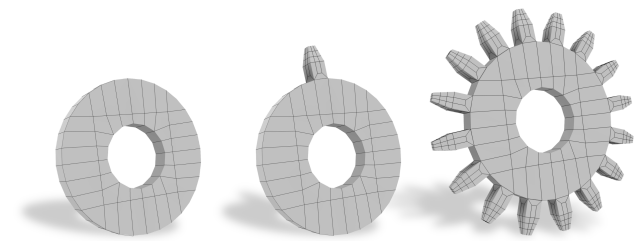


**Figure 15:** Modeling a hexmesh with multiple connected components (right) starting from a single cube (left) is possible with HexBox. The desired effect can be obtained by using hex removal to separate the various components (transition from second to third column).



**Figure 16:** Topological structure comparison between a hexmesh automatically computed with a polycube method [LVS\*13] (left) and a mesh created with HexBox (right). Although the polycube mesh contains less singular vertices and edges, these do not align well (red circles), generating a denser coarse block decomposition. Note that the polycube is quite coarse and does not even endow the ears, which would increase the geometric fidelity but also further increase the number of block domains. Manually crafting a hexmesh with box modeling intrinsically promotes the singularity alignment, yielding meshes with shape aligned edge flow and coarser layout.

**Copy/Pasting.** For shapes containing repetitive elements, such as dents in mechanical wheels or decorative patterns in artistic pieces, HexBox offers a copy/paste facility that allows to model a certain block once and install it in different places, also controlling its local orientation. This option greatly simplifies the modeling process. To validate this feature we executed a simple experiment, which consists in modeling a wheel gear with 16 dents in two different ways: by modeling each dent separately, and then by modeling a single dent once and then copy/pasting it. Using copy/paste allowed the modeler to save more than 35% of the time, producing exactly the same mesh (Fig. 17). We point the reader to the accompanying videos to observe the two modeling sessions.



**Figure 17:** Starting from a plain wheel (left), we modeled a single dent in HexBox (middle) and then copy/pasted it to insert all the others (right). The whole process took 2:42 min. Making the same model (Cogwheel1) creating each dent separately took 4:15 min.

**Remeshing with HexBox.** When a surface mesh representation of the target shape is known, HexBox can be used as a remeshing tool, exploiting a projection operator that positions the boundary vertices of the hexmesh onto the target surface (Sec. 3.2.1). Besides the projection step, these modeling sessions are similar to session where the hexmesh is created from scratch. Meshes created in this way are depicted in Fig. 1 and in Fig. 13. As for the pure modeling examples, also these meshes exhibit a good variety of topological and geometric features. Notably, remeshing of CAD shapes exhibit a good preservation of the sharp features, which are preserved by our projection operator.

## 6. Conclusion and future works

We have presented HexBox, an intuitive modeling method and interactive tool for creating and editing hexahedral meshes using a box modeling approach. The main idea behind the tool is to support volumetric modeling by modifying its surface through a set of operations that are familiar to a wide community of modelers, lowering the entrance barrier. To do so, we had to extend and combine many state-of-the-art solutions in grid-based meshing, taking into account the needs of interactive applications based on direct manipulations, with their constraints on low-latency and high-frequency feedback. As demonstrated by our results (Sec. 5), the current tool implementation is of immediate practical use, and surpasses current interactive modeling solutions (see Sec. 2) in terms of speed of modeling and complexity of the modeling space. We have also shown how we can cover both modeling from scratch, and creating and projecting a conforming hexmesh onto a target geometry.

While HexBox already contains a significant set of modeling tools and facilities, we plan to further extend it to permit users to model even more complex shapes (such those in [BRK\*22]), and to further improve the modeling experience. In particular, most of our efforts for the near future will be devoted to: provide facilities to model with extrinsic symmetries, which would allow to approximately halve the modeling effort of roughly symmetric shapes; introduce tools for smooth subdivision volumes (akin [BWX02]); improve the projection system in (Sec. 3.2.1), which currently is not always able to obtain a satisfactory geometric fidelity while retaining the positiveness of all elements' Jacobians. We also plan to perform a user evaluation to identify the strengths and weaknesses of our interactive approach. Furthermore, to keep the project alive

and foster the creation of a community around our tool, in addition to releasing the full source code and a gallery of created meshes, we will produce documentation and video tutorials.

Last but not least, we are investigating methods to allow users to start the modeling session from a non-cube root, such as an existing mesh. While it is directly feasible with little implementation effort to allow our DAG's root to encode complex structures (i.e., an entire mesh) rather than a single cube, thus allowing users to alter the structure of existing meshes, the real challenge is to extend HexBox to load an existing mesh and reconstruct the DAG representing a potential modeling session leading it.

## Acknowledgments

The authors are particularly grateful to L. Pitzalis and R. Scateni for their valuable contributions in the preliminary phase of this work, and to F. Protais for helping us with the projection code. E. Gobetti acknowledges the contribution of Sardinian Regional Authorities to CRS4 Visual and Data Intensive Computing Activities. G. Cherchi gratefully acknowledges the support to his research by PRIN 2020 project, funded by the Italian Ministry of University and Research (MUR), CUP: F73C22000430001.

## References

- [ABC\*19] ANTOLIN P., BUFFA A., COHEN E., DANNENHOFFER J. F., ELBER G., ELGETI S., HAIMES R., RIESENFELD R.: Optimizing micro-tiles in micro-structures as a design paradigm. *Computer-Aided Design 115* (2019), 23–33. 3
- [ABM19] ANTOLIN P., BUFFA A., MARTINELLI M.: Isogeometric analysis on v-reps: first results. *Computer Methods in Applied Mechanics and Engineering 355* (2019), 976–1002. 3
- [Alt23] ALTAIR: <https://www.altair.com/hypermesh/>, 2023. [Accessed: 2023-03-21]. 2
- [ANS23] ANSYS: <https://www.ansys.com/products/meshing>, 2023. [Accessed: 2023-03-21]. 2
- [BAS14] BÆRENTZEN J. A., ABDRAHIMOV R., SINGH K.: Interactive shape modeling using a skeleton-mesh co-representation. *ACM Trans. Graph. 33*, 4 (2014), 1–10. 3
- [BBC22] BRÜCKLER H., BOMMES D., CAMPEN M.: Volume parametrization quantization for hexahedral meshing. *ACM Trans. Graph. 41*, 4 (2022), 1–19. 2
- [BdVBM14] BEIRÃO DA VEIGA L., BREZZI F., MARINI L. D., RUSSO A.: The hitchhiker's guide to the virtual element method. *Mathematical models and methods in applied sciences 24*, 08 (2014), 1541–1573. 5
- [BJD\*12] BOROSÁN P., JIN M., DECARLO D., GINGOLD Y., NEALEN A.: Rigmesh: automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph. 31*, 6 (2012), 1–9. 3
- [Bla00] BLACKER T. D.: Meeting the challenge for automated conformal hexahedral meshing. In *Proc. 9th International Meshing Roundtable* (2000), Springer, pp. 11–20. 2
- [BMU\*15] BARBIERI S., MELONI P., USAI F., SCATENI R., ET AL.: Skeleton lab: an interactive tool to create, edit, and repair curve-skeletons. In *STAG* (2015), pp. 121–128. 3
- [BMW12] BÆRENTZEN J. A., MISZTAL M. K., WELNICKA K.: Converting skeletal structures to quad dominant meshes. *Computers & Graphics 36*, 5 (2012), 555–561. 3
- [BRK\*22] BEAUFORT P.-A., REBEROL M., KALMYKOV D., LIU H., LEDOUX F., BOMMES D.: Hex me if you can. *Computer Graphics Forum 41*, 5 (2022), 125–134. 12
- [BTP\*19] BRACCI M., TARINI M., PIETRONI N., LIVESU M., CIGNONI P.: Hexalab. net: An online viewer for hexahedral meshes. *Computer-Aided Design 110* (2019), 24–36. 3, 9, 11
- [BWX02] BAJAJ C., WARREN J., XU G.: A smooth subdivision scheme for hexahedral meshes. *The visual computer 18*, 5/6 (2002), 344–356. 12
- [CAS\*19] CHERCHI G., ALLIEZ P., SCATENI R., LYON M., BOMMES D.: Selective padding for polycube-based hexahedral meshing. *Computer Graphics Forum 38*, 1 (2019), 580–591. 3
- [CC19] CORMAN E., CRANE K.: Symmetric moving frames. *ACM Trans. Graph. 38*, 4 (2019), 1–16. 2
- [CLS16] CHERCHI G., LIVESU M., SCATENI R.: Polycube simplification for coarse layouts of surfaces and volumes. *Computer Graphics Forum 35*, 5 (2016), 11–20. 2
- [Cor] CORNUT O.: Dear ImGui. <https://github.com/ocornut/imgui>. [Accessed: 2023-03-21]. 9
- [Cor22] COREFORM LLC: Coreform cubit basics: Hex meshing fundamentals. [https://youtu.be/TOFq-Pkn1\\_A?t=879](https://youtu.be/TOFq-Pkn1_A?t=879), 2022. [Accessed: 2023-04-02]. 3
- [Cor23] COREFORM: <https://coreform.com/products/coreform-cubit/government/>, 2023. [Accessed: 2023-03-21]. 2
- [CSS06] CARBONERA C. D., SHEPHERD J. F., SHEPHERD J. F.: A constructive approach to constrained hexahedral mesh generation. In *Proceedings of the 15th international meshing roundtable* (2006), Springer, pp. 435–452. 3
- [CUB23] CUBIT: <https://cubit.sandia.gov>, 2023. [Accessed: 2023-03-21]. 2
- [CWM\*16] CHAN J., WANG Z., MODAVE A., REMACLE J.-F., WARBURTON T.: Gpu-accelerated discontinuous galerkin methods on hybrid meshes. *Journal of Computational Physics 318* (2016), 142–168. 4
- [CZL21] CAI W.-H., ZHAN J.-M., LUO Y.-Y.: User-intervened structured meshing methods and applications for complex flow fields based on multiblock partitioning. *Journal of Computational Design and Engineering 8*, 1 (2021), 225–238. 3
- [DPM\*22] DUMERY C., PROTAIS F., MESTRALLET S., BOURCIER C., LEDOUX F.: Evocube: A genetic labelling framework for polycube-maps. *Computer Graphics Forum 41*, 6 (2022), 467–479. 2
- [Eri13] ERICKSON J.: Efficiently hex-meshing things with topology. In *Proceedings of the twenty-ninth annual symposium on Computational geometry* (2013), pp. 37–46. 3
- [FBL16] FU X.-M., BAI C.-Y., LIU Y.: Efficient volumetric polycube-map construction. *Computer graphics forum 35*, 7 (2016), 97–106. 2
- [Flo15] FLOATER M. S.: Generalized barycentric coordinates and applications. *Acta Numerica 24* (2015), 161–214. 7
- [FXB16] FANG X., XU W., BAO H., J.: All-hex meshing using closed-form induced polycube. *ACM Trans. Graph. 35*, 4 (2016), 1–9. 2
- [GLYL20] GUO H.-X., LIU X., YAN D.-M., LIU Y.: Cut-enhanced polycube-maps for feature-aware all-hex meshing. *ACM Trans. Graph. 39*, 4 (2020), 106–1. 2, 3
- [GSP19] GAO X., SHEN H., PANOZZO D.: Feature preserving octree-based hexahedral meshing. *Computer graphics forum 38*, 5 (2019), 135–149. 2, 7
- [GSZ11] GREGSON J., SHEFFER A., ZHANG E.: All-hex mesh generation via volumetric polycube deformation. *Computer graphics forum 30*, 5 (2011), 1407–1416. 2, 3
- [HJS\*14] HUANG J., JIANG T., SHI Z., TONG Y., BAO H., DESBRUN M.:  $\ell_1$ -based construction of polycube maps from complex shapes. *ACM Trans. Graph. 33*, 3 (2014), 1–11. 2
- [ISS09] ITO Y., SHIH A. M., SONI B. K.: Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *International Journal for Numerical Methods in Engineering 77*, 13 (2009), 1809–1833. 4, 5

- [JHW\*13] JIANG T., HUANG J., WANG Y., TONG Y., BAO H.: Frame field singularity correction for automatic hexahedralization. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (2013), 1189–1199. 2
- [JLW10] JI Z., LIU L., WANG Y.: B-mesh: a modeling system for base meshes of 3d articulated shapes. *Computer Graphics Forum* 29, 7 (2010), 2169–2177. 3
- [KBLK14] KREMER M., BOMMES D., LIM I., KOBELT L.: Advanced automatic hexahedral mesh generation from surface quad meshes. In *Proceedings of the 22nd International Meshing Roundtable* (2014), Springer, pp. 147–164. 3
- [KLF16] KOWALSKI N., LEDOUX F., FREY P.: Smoothness driven frame field generation for hexahedral meshing. *Computer-Aided Design* 72 (2016), 65–77. 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation. 2
- [LAPS17] LIVESU M., ATTENE M., PATANÉ G., SPAGNUOLO M.: Explicit cylindrical maps for general tubular shapes. *Computer-Aided Design* 90 (2017), 27–36. 3
- [Liv19] LIVESU M.: cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes. *Transactions on Computational Science XXXIV* (2019). <https://github.com/mlivesu/cinolib/>. 9
- [LLD12] LIN H., LIAO H., DENG C.: Filling triangular mesh model with all-hex mesh by volume subdivision fitting. *State Key Lab of CAD & CG, Zhejiang University Report No: TR ZJUCAD 2* (2012), 2012. 3
- [LLX\*12] LI Y., LIU Y., XU W., WANG W., GUO B.: All-hex meshing using singularity-restricted field. *ACM Trans. Graph.* 31, 6 (Nov. 2012). 2
- [LMPS16] LIVESU M., MUNTONI A., PUPPO E., SCATENI R.: Skeleton-driven adaptive hexahedral meshing of tubular shapes. *Computer Graphics Forum* 35, 7 (2016), 237–246. 3
- [LPC22] LIVESU M., PITZALIS L., CHERCHI G.: Optimal dual schemes for adaptive grid based hexmeshing. *ACM Trans. Graph.* 41, 2 (2022). 2, 5, 10
- [LPP\*20] LIVESU M., PIETRONI N., PUPPO E., SHEFFER A., CIGNONI P.: Loopycuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing. *ACM Trans. Graph. (SIGGRAPH)* 39, 4 (2020). 3
- [LQS17] LU J. H.-C., QUADROS W. R., SHIMADA K.: Evaluation of user-guided semi-automatic decomposition tool for hexahedral mesh generation. *Journal of Computational Design and Engineering* 4, 4 (2017), 330–338. 2
- [LSVT15] LIVESU M., SHEFFER A., VINING N., TARINI M.: Practical hex-mesh optimization via edge-cone rectification. *ACM Trans. Graph. (Proc. SIGGRAPH 2015)* 34, 4 (2015). 7
- [LVS\*13] LIVESU M., VINING N., SHEFFER A., GREGSON J., SCATENI R.: Polycut: Monotone graph-cuts for polycube base-complex construction. *ACM Trans. Graph. (Proc. SIGGRAPH ASIA 2013)* 32, 6 (2013). 2, 12
- [LW08] LEDOUX F., WEILL J.-C.: An extension of the reliable whisker weaving algorithm. In *Proceedings of the 16th international meshing roundtable* (2008), Springer Berlin Heidelberg, pp. 215–232. 3
- [LZC\*18] LIU H., ZHANG P., CHIEN E., SOLOMON J., BOMMES D.: Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.* 37, 4 (2018), 93–1. 2
- [LZLW15] LIU L., ZHANG Y., LIU Y., WANG W.: Feature-preserving t-mesh construction using skeleton-based polycubes. *Computer-Aided Design* 58 (2015), 162–172. 3
- [LZS\*21] LI L., ZHANG P., SMIRNOV D., ABULNAGA S. M., SOLOMON J.: Interactive all-hex meshing via cuboid decomposition. *ACM Trans. Graph.* 40, 6 (2021), 1–17. 3, 7, 10
- [Mar09] MARÉCHAL L.: Advances in octree-based all-hexahedral mesh generation: handling sharp features. In *Proceedings of the 18th international meshing roundtable*. Springer, 2009, pp. 65–84. 2
- [MCBC22] MANDAD M., CHEN R., BOMMES D., CAMPEN M.: Intrinsic mixed-integer polycubes for hexahedral meshing. *Computer Aided Geometric Design* 94 (2022), 102078. 2
- [ME16] MASSARWI F., ELBER G.: A b-spline based framework for volumetric object modeling. *Computer-Aided Design* 78 (2016), 36–47. 3
- [MGV11] MACÉDO I., GOIS J. P., VELHO L.: Hermite radial basis functions implicits. *Computer Graphics Forum* 30, 1 (2011), 27–42. 3
- [MH99] MÜLLER-HANNEMANN M.: Hexahedral mesh generation by successive dual cycle elimination. *Engineering with computers* 15, 3 (1999), 269–279. 3
- [Mit96] MITCHELL S. A.: A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. In *STACS 96: 13th Annual Symposium on Theoretical Aspects of Computer Science Grenoble, France, February 22–24, 1996 Proceedings* 13 (1996), Springer, pp. 465–476. 3
- [Mit99] MITCHELL S. A.: The all-hex geode-template for conforming a diced tetrahedral mesh to any diced hexahedral mesh. *Engineering with Computers* 15 (1999), 228–235. 3
- [MT95] MITCHELL S. A., TAUTGES T. J.: Pillowing doublets: Refining a mesh to ensure that faces share at most one edge. In *Proc. 4th International Meshing Roundtable* (1995), pp. 231–240. 7
- [NABR15] NGUYEN V. P., ANITESCU C., BORDAS S. P., RABCZUK T.: Isogeometric analysis: an overview and computer implementation aspects. *Mathematics and Computers in Simulation* 117 (2015), 89–116. 3
- [NRP11] NIESER M., REITEBUCH U., POLTHIER K.: Cubecover—parameterization of 3d volumes. *Computer Graphics Forum* 30, 5 (2011), 1397–1406. 2
- [NWW21] NEUHAUSER C., WANG J., WESTERMANN R.: Interactive focus+ context rendering for hexahedral mesh inspection. *IEEE Transactions on Visualization and Computer Graphics* 27, 8 (2021), 3505–3518. 3
- [PBS20] PALMER D., BOMMES D., SOLOMON J.: Algebraic representations for volumetric frame fields. *ACM Trans. Graph.* 39, 2 (Apr. 2020). 2, 10
- [PBS22] PANDEY K., BÆRENTZEN J. A., SINGH K.: Face extrusion quad meshes. In *ACM SIGGRAPH 2022 Conference Proceedings* (2022), pp. 1–9. 3, 7
- [PCS\*22] PIETRONI N., CAMPEN M., SHEFFER A., CHERCHI G., BOMMES D., GAO X., SCATENI R., LEDOUX F., REMACLE J.-F., LIVESU M.: Hex-mesh generation and processing: A survey. *ACM Trans. Graph.* 42, 2 (oct 2022). 2, 3, 5
- [PCS\*23] PIETRONI N., CAMPEN M., SHEFFER A., CHERCHI G., BOMMES D., GAO X., SCATENI R., LEDOUX F., REMACLE J.-F., LIVESU M.: A course on hex-mesh generation and processing. In *ACM SIGGRAPH Asia 2022 Courses* (New York, NY, USA, 2023), SIGGRAPH Asia '22, Association for Computing Machinery. 2, 5
- [PLC\*21] PITZALIS L., LIVESU M., CHERCHI G., GOBBETTI E., SCATENI R.: Generalized adaptive refinement for grid-based hexahedral meshing. *ACM Trans. Graph. (SIGGRAPH Asia)* 40, 6 (2021). 2, 3, 5
- [PNA\*21] PIETRONI N., NUVOLE S., ALDERIGHI T., CIGNONI P., TARINI M.: Reliable feature-line driven quad-remeshing. *ACM Trans. Graph.* 40, 4 (jul 2021). 6
- [SHG\*22] SCHNEIDER T., HU Y., GAO X., DUMAS J., ZORIN D., PANOZZO D.: A large-scale comparison of tetrahedral and hexahedral elements for solving elliptic pdes with the finite element method. *ACM Trans. Graph.* 41, 3 (mar 2022). 1, 4
- [SVB17] SOLOMON J., VAXMAN A., BOMMES D.: Boundary element octahedral fields in volumes. *ACM Trans. Graph.* 36, 4 (May 2017). 2
- [Tak19] TAKAYAMA K.: Dual sheet meshing: An interactive approach to robust hexahedralization. *Computer Graphics Forum* 38, 2 (2019), 37–48. 3, 5

- [Tak23] TAKAYAMA K.: Personal communication, 2023. 3
- [Tau04] TAUTGES T. J.: Moab-sd: integrated structured and unstructured mesh representation. *Engineering With Computers* 20 (2004), 286–293. 10
- [TBM96] TAUTGES T. J., BLACKER T., MITCHELL S. A.: The whisker weaving algorithm: a connectivity-based method for constructing all-hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering* 39, 19 (1996), 3327–3349. 3
- [Thu93] THURSTON W.: Hexahedral decomposition of polyhedra. *Posting to sci. math* 25 (1993). 3
- [ULP\*15] USAI F., LIVESU M., PUPPO E., TARINI M., SCATENI R.: Extraction of the quad layout of a triangle mesh guided by its curve skeleton. *ACM Trans. Graph.* 35, 1 (2015), 1–13. 3
- [Vau12] VAUGHAN W.: *Digital Modeling*. New Riders, 2012. 2
- [VPR19] VERHETSEL K., PELLERIN J., REMACLE J.-F.: Finding hexahedrizations for small quadrangulations of the sphere. *ACM Trans. Graph.* 38, 4 (2019), 1–13. 3
- [WDL\*12] WACKERS J., DENG G., LEROYER A., QUEUTEY P., VISONNEAU M.: Adaptive grid refinement for hydrodynamic flows. *Computers & Fluids* 55 (2012), 85–100. 4
- [WGM11] WYMAN N., GALPIN P., MIRSKY M.: A method for geometry-sensitive, cfd solver independent mesh adaptation. In *Eleventh International Conference on Computational Fluid Dynamics (ICCFD11)* (2011). 4
- [WZT\*18] WEI X., ZHANG Y. J., TOSHNIWAL D., SPELEERS H., LI X., MANNI C., EVANS J. A., HUGHES T. J.: Blended b-spline construction on unstructured quadrilateral and hexahedral meshes with optimal convergence rates in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering* 341 (2018), 609–639. 10
- [XC18] XU K., CHEN G.: Hexahedral mesh structure visualization and evaluation. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 1173–1182. 3
- [XGDC17] XU K., GAO X., DENG Z., CHEN G.: Hexahedral meshing with varying element sizes. In *Computer Graphics Forum* (2017), vol. 36, Wiley Online Library, pp. 540–553. 2
- [YLZ22] YU Y., LIU J. G., ZHANG Y. J.: Hexdom: polycube-based hexahedral-dominant mesh generation. In *Mesh Generation and Adaptation: Cutting-Edge Techniques*. Springer, 2022, pp. 137–155. 3
- [YWL\*22] YU Y., WEI X., LI A., LIU J. G., HE J., ZHANG Y. J.: Hexgen and hex2spline: polycube-based hexahedral mesh generation and spline modeling for isogeometric analysis applications in ls-dyna. In *Geometric Challenges in Isogeometric Analysis*. Springer, 2022, pp. 333–363. 3
- [ZBG\*07] ZHANG Y., BAZILEVS Y., GOSWAMI S., BAJAJ C. L., HUGHES T. J.: Patient-specific vascular nurbs modeling for isogeometric analysis of blood flow. *Computer methods in applied mechanics and engineering* 196, 29–30 (2007), 2943–2959. 3