Generalized Adaptive Refinement for Grid-based Hexahedral Meshing

LUCA PITZALIS, University of Cagliari and CRS4, Italy
MARCO LIVESU, IMATI-CNR, Italy
GIANMARCO CHERCHI, University of Cagliari, Italy
ENRICO GOBBETTI, CRS4, Italy
RICCARDO SCATENI, University of Cagliari, Italy

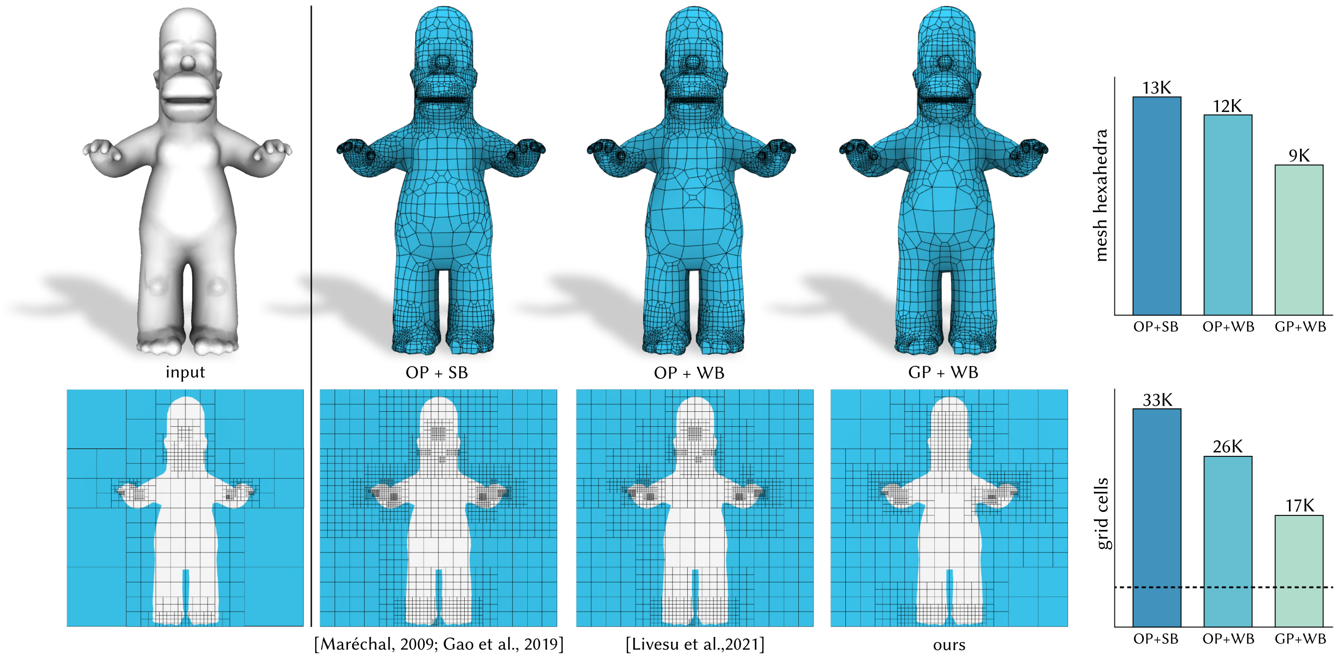


Fig. 1. Converting an adaptively refined grid (left) into a conforming hexahedral mesh requires additional refinement to fulfill the balancing and pairing criteria. Enforcing pairing through an octree (OP) yields dense grids, both with strong (SB) and weak (WB) balancing. Our generalized pairing criterion (GP) allows us to significantly reduce both grid and mesh size, while still guaranteeing the generation of a pure hexahedral mesh with any of the known dual schemes [Gao et al. 2019; Livesu et al. 2021; Maréchal 2009]. The dashed line at the bottom right indicates the size of the input grid.

Due to their nice numerical properties, conforming hexahedral meshes are considered a prominent computational domain for simulation tasks. However, the automatic decomposition of a general 3D volume into a small number of hexahedral elements is very challenging. Methods that create an adaptive Cartesian grid and convert it into a conforming mesh offer superior robustness and are the only ones concretely used in the industry. Topological schemes that permit this conversion can be applied only if precise compatibility conditions among grid elements are observed. Some of these conditions are local, hence easy to formulate; others are not and are much harder to satisfy. State-of-the-art approaches fulfill these conditions by prescribing additional refinement based on special building rules for octrees. These methods operate in a restricted space of solutions and are prone to severely over-refine the input grids, creating a bottleneck in the simulation pipeline. In this article, we introduce a novel approach to transform a general adaptive grid into a new grid meeting hexmeshing criteria, without resorting to tree rules. Our key insight is that we can formulate all compatibility conditions as linear constraints in an integer programming problem by choosing the proper set of unknowns. Since we operate in a broader solution space, we are able to meet topological hexmeshing criteria at a much coarser scale than methods using octrees, also supporting generalized grids of any shape or topology. We demonstrate the superiority of our approach for both traditional grid-based hexmeshing and adaptive polycube-based hexmeshing. In all our experiments, our method never prescribed more refinement than the prior art and, in the average case, it introduced close to half the number of extra cells.

CCS Concepts: • **Computing methodologies → Mesh geometry models**; **Mesh models**.

Authors' addresses: Luca Pitzalis, University of Cagliari and CRS4, Italy, luca.pitzalis94@unica.it; Marco Livesu, IMATI-CNR, Italy, marco.livesu@gmail.com; Gianmarco Cherchi, University of Cagliari, Italy, g.cherchi@unica.it; Enrico Gobbetti, CRS4, Italy, gobbetti@crs4.it; Riccardo Scateni, University of Cagliari, Italy, riccardo@unica.it.

0730-0301/2021/12-ART257 $15.00
https://doi.org/10.1145/3478513.3480508

Additional Key Words and Phrases: Hexahedral meshing, all-hex grids, adaptive refinement, integer linear programming

ACM Reference Format:
Luca Pitzalis, Marco Livesu, Gianmarco Cherchi, Enrico Gobbetti, and Riccardo Scateni. 2021. Generalized Adaptive Refinement for Grid-based Hexahedral Meshing. *ACM Trans. Graph.* 40, 6, Article 257 (December 2021), 13 pages. https://doi.org/10.1145/3478513.3480508

1 INTRODUCTION

Solving a partial differential equation (PDE) on a discrete mesh requires finding a good balance among geometric fidelity, numerical accuracy, and efficiency. Extremely dense meshes ensure good geometric approximation and tight error bounds, but are computationally expensive and require significant storage and communication bandwidth. Coarse meshes ensure fast computation and reduce memory pressure, but approximation error grows significantly. When generating meshes for simulations, it is therefore essential to introduce the least number of elements that provide the wanted accuracy, maximizing the efficiency of storage and computation.

Multiple studies in Computation Fluid Dynamics (CFD) and linear Finite Element Methods (FEM) have shown that hexahedral meshes perform better than tetrahedral meshes since they can yield tighter approximation bounds at a lower cost [Benzley et al. 1995; Cifuentes and Kalbag 1992; Erke Wang and Rauch 2004; Schneider et al. 2019; Wang et al. 2021]. Despite its importance and decades of academic and industrial research, the generation of hexahedral meshes conforming to given 3D shapes is still an open problem [Armstrong et al. 2015; Blacker 2000; Owen 1998; Schneiders 2000a; Shepherd and Johnson 2008; Tautges 2001].

As of today, only the solutions based on adaptive grids have proven to meet stringent scalability and robustness requirements for general shapes, and are the only automatic methods implemented in commercial software [Distene SAS 2020]. These methods build an all-hex mesh by intersecting the input model with a Cartesian grid defining the mesh interior, and connecting it to the surface. Adaptive grids are employed instead of regular grids to reduce the mesh size. However, a locally-refined grid is not a pure hexahedral mesh, because spurious vertices (*hanging nodes*) arise at the interface between cuboids at different refinement levels. Local topological schemes permit to convert an adaptive grid to a conforming hexmesh via dualization, connecting pairs of adjacent hanging nodes to regularize per-vertex valences (Fig. 3). For this process to be possible, the two following criteria must be fulfilled:

- **balancing** – the difference in the amount of refinement of adjacent grid elements must not be greater than one;
- **pairing** – clusters of elements with equal size must have an even number of items along all their sides.

While the balancing condition can be easily encoded as a local constraint between pairs of adjacent grid cells, pairing is non-local, hence much harder to deal with.

Prior methods meet these constraints by transforming the non-local pairing condition into a more stringent local rule in a hierarchical space partitioning. Specifically, they impose that refined areas correspond to the internal (i.e, non leaf) nodes of a restricted octree, and guarantee the pairing property by imposing that if the child of an octant has been split, then its siblings having the same octant as their parent must be split as well [Gao et al. 2019; Hu et al. 2013; Livesu et al. 2021; Maréchal 2009]. As shown in Fig. 1, this process has a major impact on grid size. In our experiments, we found cases where the final grid grew more than 9 times its original size (Section 6).

Our main observation is that a considerable portion of this refinement is unnecessary and could be avoided if we were able to explore a larger space of solutions than the one considered by methods based on octrees. A concise 2D example is given in Fig. 2: a dense 4×4 sub-grid always satisfies both balancing and pairing, regardless of its relative position within the coarser grid. Nevertheless, if we position the subgrid *across* multiple quadrants in the hierarchy, the tree building rules will require to further split many nodes, producing a valid grid that is almost three times bigger than the original (already valid) one.

In this article, we introduce a novel method that significantly enlarges the space of hexmeshable adaptive grids, drastically reducing the amount of refinement necessary for balancing and pairing. Our key insight is that we can efficiently handle the non-local pairing condition by controlling the refinement of grid vertices instead of cells. This change of variable allows us to formulate the whole pairing problem as an integer optimization problem subject to linear constraints, without forcing fixed refinement patterns dictated by octree rules. We ultimately transfer refinement information from vertices to cells, isolating the least number of elements that must be split to fulfill all hexmeshing topological criteria. Moreover, we integrate this solution into an algorithm that combines pairing and balancing without any resort to tree rules, leading to a general approach applicable to any adaptive grid, regardless of its shape or topology.

As demonstrated in Section 6, our formulation scales well with grid size, and can be readily adapted to a variety of situations, including the local refinement of a polycube mapping [Gregson et al. 2011] to improve geometric or numeric accuracy, and the direct replacement of standard octree-based refiners in state-of-the-art hex-meshing pipelines [Gao et al. 2019; Hu et al. 2013; Livesu et al. 2021; Maréchal 2009]. In all these settings, our method ultimately produces conforming hexmeshes much coarser than prior grid-based solutions. In particular, our extensive experimentation shows that, on common benchmarks, hexmeshability is ensured by introducing, on average, about half the number of extra cells with respect to competing octree-based solutions. To grant full reproducibility and maximum adoption of our technique, we also release our reference implementation at the following GitHub repository: `github.com/cg3hci/Gen-Adapt-Ref-for-Hexmeshing`.

2 RELATED WORK

Hexmeshing is a very broad field, and a full review is out of the scope of this article. In the following, we only focus on the methods most closely related to ours. We refer the reader to a well-established survey [Armstrong et al. 2015] for a broader perspective.

Grid-based methods. Pioneering work on hexahedral meshing [Schneiders 1996; Schneiders and Bünten 1995] used a regular grid to position a set of hexahedra fully inside the object, exploiting

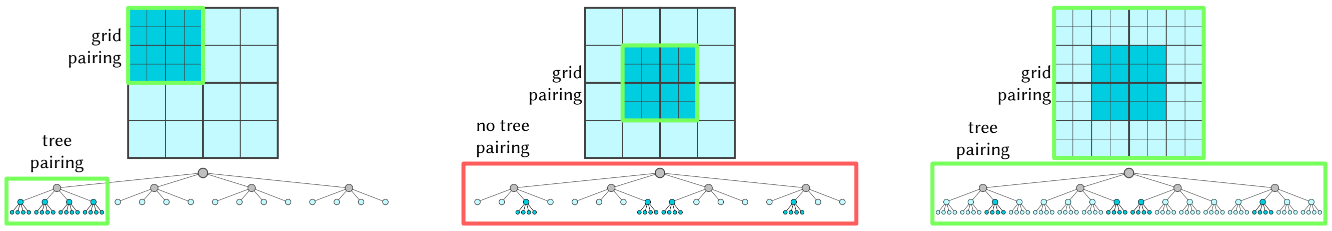


Fig. 2. Refining a regular 4×4 grid at a 2×2 minor (blue) always satisfies balancing and pairing, regardless of subgrid positioning. Methods based on rigid trees achieve pairing by asking refined areas to correspond to inner nodes in the tree, and also ensuring that both such nodes and their siblings are fully split [Maréchal 2009]. Left: if the dark minor aligns with the hierarchical structure, pairing is observed both in the grid and the tree. Middle: if the minor does not align with the hierarchy, there is grid pairing but not tree pairing. Right: enforcing pairing *through the tree* unnecessarily refines the whole grid.

vertex projection and padding to complete the mesh with boundary elements conforming to surface features. Adaptive grids based on octree refinement, together with topological schemes to incorporate the hanging nodes, were later introduced to reduce the number of elements [Schneiders 1997, 2000a,b; Schneiders et al. 1996]. Most of the later research in the field tackled the problem of projecting the grid surface to the target object, a critical operation that should secure well-shaped elements (i.e., valid for simulation) and also preserve sharp features [Gao et al. 2019; Ito et al. 2009; Lin et al. 2015; Schneiders 1996; Schneiders and Bünten 1995; Zhang and Bajaj 2006]. Our contribution focuses on the combinatorial problem of defining valid all-hex connectivity and is therefore orthogonal to this body of literature. Any existing projection techniques can be coupled with our method to produce a complete, end-to-end, hexmeshing pipeline.

Hexmeshing from adaptive grids. Just a few authors focused on the problem of defining the hexmesh connectivity from a locally-refined grid containing hanging nodes. To the best of our knowledge, all adaptive methods use octrees, which are locally refined according to topological [Mitchell and Vavasis 1992] or geometric criteria, such as local thickness [Livesu et al. 2021; Maréchal 2009], surface approximation [Gao et al. 2019], normal similarity [Ito et al. 2009], or a combination of those [Bawin et al. 2020]. A first wave of methods was based on the early refinement schemes proposed by Schneiders [1997]. However, these schemes are not exhaustive, as they allow to produce a valid hexahedralization for only 4 out of the 20 possible transitions. Maréchal [2009] showed that if the

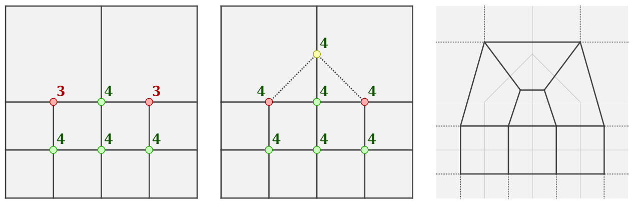


Fig. 3. Pairing and connecting hanging nodes with valence 3 arising at the interface between adjacent cells with different refinement, yields a new mesh where all vertices have valence 4. The dual of this mesh is a quadrilateral mesh. The balancing and pairing criteria jointly ensure that this transition scheme can be applied everywhere, producing a conforming mesh. Similar schemes also exist for 3D adaptive grids [Gao et al. 2019; Livesu et al. 2021; Maréchal 2009].

input grid satisfies both the balancing and the pairing conditions, a full hexahedral mesh can be obtained via dualization. The resulting algorithm has been exploited in a commercial software called MeshGems [Distene SAS 2020]. Similar results can also be obtained by dualizing the grid first, and then substituting clusters of non-hexahedral elements with a finite set of templates [Gao et al. 2019]. Very recently, Livesu and colleagues [2021] introduced a novel set of schemes that supports a weaker definition of balancing, producing meshes with simpler singular structure and lower element count. Following their terminology, in the remainder of the paper we will refer to the balancing criterion used in [Gao et al. 2019; Maréchal 2009] as *strong* balancing, and the one used in [Livesu et al. 2021] as *weak* balancing. The two definitions differ by the notion of neighborhood they adopt. For the strong case, all grid cells incident at a face, vertex, or edge must be balanced. For the weak case, only face-adjacent cells must be balanced. Our method is compatible with both definitions, but since the latter performs better than the former, we used the weak balancing version in all our experiments. To the best of our knowledge, our work is the first to observe that the space of adaptive grids that admit a hexmeshing is much wider than the space of grids that can be created with octrees. By substituting a rigid hierarchical structure with a more generic integer linear problem, we are able to obtain meshes with much lower (and in the worst case equal) element count, while still relying on the same set of topological schemes.

Voxelization and polycubes. Besides adaptive methods based on octrees, grids are also employed by hexmeshing methods that use voxelizations [Lin et al. 2015] and polycubes [Cherchi et al. 2016; Fang et al. 2016; Fu et al. 2016; Gregson et al. 2011; Huang et al. 2014; Livesu et al. 2013]. These methods typically do not involve local refinement, although initial attempts to combine adaptive meshing and polycubes have been proposed in recent literature. Specifically, Hu et al. [2016] fit an octree in polycube space, and use ideas from Maréchal [2009] to restore the all-hex connectivity. Conversely, other approaches like [Cherchi et al. 2019; Xu et al. 2017] keep the sampling grid fixed, and enlarge or shrink portions of the polycube to obtain the wanted adaptivity. Our method can be seamlessly incorporated in the first approach, granting a lower element count, and is superior to the latter, which imposes that the singular structure of the mesh does not change, thus limiting the ability to adapt the mesh to features that are not present in polycube space (Section 6.2).

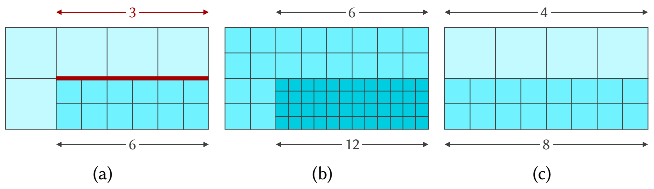


Fig. 4. The grid in (a) does not satisfy pairing because the interface between coarse and refined elements (in red) has an odd size. Pairing could be trivially obtained with a global step of refinement, which doubles the size of all sides and makes them even (b). However, this solution is highly expensive because it quadruplicates the number of cells (in 3D, the factor is 8×). Our method restores pairing by splitting one single cell, which is the optimal solution (c).

Other methods. A variety of alternative methods for hexahedral meshing have been proposed over the years, including sweepings along a given direction or guiding curve [Livesu et al. 2016; Staten et al. 2010; Wu et al. 2018], advancing front methods [Kremer et al. 2014], dual methods [Ledoux and Weill 2008], and methods guided by various forms of direction fields [Corman and Crane 2019; Jiang et al. 2013; Li et al. 2012; Liu et al. 2018]. Moreover, several authors have also proposed solutions for creating hex-dominant rather than all-hex meshes (e.g., [Livesu et al. 2020; Ray et al. 2018]). These approaches are orthogonal to ours, and have no methodological overlap.

3 OVERVIEW

Our algorithm transforms an adaptive Cartesian G into a modified grid G' that is suitable for conforming hexahedral meshing. We do not make any assumption on the motivation of the already performed refinement on G, which is related to the specific application and may come from prior knowledge (e.g., from a domain expert). The refinement transforming G into G' has, instead, the only goal to ensure hexmeshability through the satisfaction of both the balancing and pairing conditions introduced in Section 1. This allows us to hexmesh G' by first applying known topological schemes to obtain the all-hexahedra topology [Livesu et al. 2021], and then projecting the mesh onto the target geometry, as detailed in Section 6.

While this problem admits infinitely many solutions, our goal is to find the grid G' that minimizes its distance from the input grid G, according to metric

$$d(G, G') = \sum_{c \in G} |r(c) - r(c')| \,, \tag{1}$$

subject to per-cell constraints $r(c') \geq r(c)$, where $r(c)$ and $r(c')$ represent the amount of input and output refinement for cell c, respectively. In other words, we are looking for the minimum amount of *extra* refinement that fulfills the topological conditions for hexahedral meshing. We want the refinement to be minimal because, to be efficient, the mesh must not be unnecessarily dense. At the same time, we cannot under-refine any cell because the input density was the outcome of a process on which we have no control.

Our solution improves over current tree-based approaches, addressing the hexmeshability conditions summarized in Section 1 with a mixed algorithmic and numerical approach.

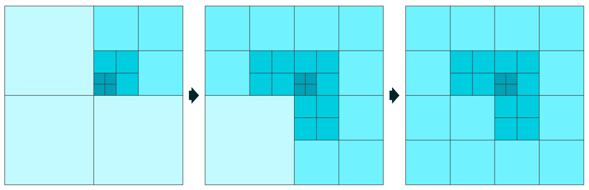


Fig. 5. The input grid (left) does not satisfy balancing, because its top-left and bottom-right cells have refinement one and are adjacent to cells with refinement four. Iteratively splitting each cell adjacent to a cell smaller than half of its size converges to a balanced grid (right). In this example, the final grid is weakly balanced. Splitting the cell with refinement two that is vertex-adjacent to one the cells with refinement four would produce a strongly balanced grid.

3.1 Balancing

The balancing condition imposes that the difference in the amount of refinement for pairs of adjacent cells must be either zero or one. As this condition is purely local, we can easily address it algorithmically with an iterative approach. We progressively split grid cells adjacent to another cell with more than one step of further refinement, repeating the process until all grid cells are balanced (Fig. 5). The local criterion can be either the strong balancing used in [Gao et al. 2019; Maréchal 2009] or the weak balancing used in [Livesu et al. 2021]. These two criteria only differ in the notion of per-cell neighborhood they use, and are both compatible with our pipeline, as already discussed in Section 2.

3.2 Pairing

A grid satisfies pairing if every cluster of face-adjacent cells with the same size has an even number of items along all its sides (Fig. 4). Differently from balancing, this condition necessitates to compute span lengths and cannot be simply expressed as a set of pairwise constraints among adjacent cells.

Our main intuition is that pairing can be more easily formulated by changing the variables of the problem, operating on small clusters of cells sharing a grid vertex instead of single cells. Grid vertices have a less local view on the grid, which consists in a 2×2 sub-grid in 2D, and a $2 \times 2 \times 2$ sub-grid in 3D. In both cases, the cluster of cells incident to a vertex has even size across all dimensions. Therefore, if we manage to distribute refinement through grid vertices and guarantee no overlap between the clusters associated with vertices that carry refinement, we can ensure that all clusters of refined cells will have even sides, thus achieving pairing.

Even if it may not seem obvious, this idea is deeply connected with the tree rules expressed by Maréchal [2009] and used by all octree methods. These rules essentially impose that clusters of cells with same refinement correspond to leaf parents in the tree. But the parent of 8 leaves can be thought of as the only grid vertex shared by all its children. Therefore, one could interpret this operation as the grid vertex (the parent) controlling the refinement of all its children (the leaves). Octree-based refinement makes this operation safe, in the sense that, by construction, there cannot exist conflicts in the assignment, since each leaf parent controls a unique set of octants. For the same reason, this approach is very limiting, because not all grid nodes can be associated to parents of leaves in the tree, hence the number of possible solutions is restricted by the rigid octree structure.

Our novel formulation allows us to implement the same mechanism of distributing refinement through grid vertices, but lets *any* grid vertex distribute refinement to its incident cells. In our approach, conflicts among adjacent vertices with overlapping minors are not handled by grid refinement rules but with a set of carefully crafted linear constraints. This allows us to parameterize a much wider space of valid solutions and formulate and solve pairing as an integer linear programming (ILP) problem.

Considering that under-refinement is not allowed, the only way to fulfill pairing is to enlarge clusters of refined elements, splitting their neighbors. If the input grid is assumed to be balanced, this can never result in splitting a grid cell more than once, meaning that the ILP we solve is actually a binary problem.

In the following, we first formalize our pairing approach as an ILP with binary unknowns for regular grids (Section 4). Then, in Section 5, we illustrate how this basic strategy can be extended to adaptive grids.

4 REGULAR BINARY GRIDS

We assume a regular grid G with prescribed binary refinement as input, meaning that each grid cell has a Boolean flag associated with it, which indicates whether it should be split once or kept as it is. Our goal is to split the least amount of cells, such that the input refinement plan is observed and the output grid satisfies the pairing criterion. Since only two levels of refinement are possible, balancing is satisfied by construction.

The reader may observe that the pairing problem admits a trivial solution: since the double of any odd number is even, applying one extra step of refinement to each cell would trivially ensure pairing everywhere (Fig. 4). This solution is impractical, though, because it would increase the grid size by a factor of 4 in 2D and 8 in 3D, thus violating our minimality criterion in most situations. For instance, we expect that, when the prescribed input refinement already yields a grid satisfying the pairing condition, the pairing algorithm does not prescribe any additional refinement. Notice that previous octree-based methods fail to satisfy this basic requirement, unnecessarily increasing the grid size.

As briefly mentioned in Section 3, we formalize the pairing problem as an ILP with binary unknowns, assigning refinement to grid vertices v, and letting grid cells c accumulate refinement indirectly from their incident vertices, according to formulation

$$r(c') = \sum_{v \in c} r(v) \quad . \tag{2}$$

where $v \in c$ are the 8 vertices of the cell c, $r(v)$ are the binary unknowns representing the per-vertex refinement, c are the grid cells, and $r(c')$ is the output per-cell refinement computed in post-processing after solving the ILP. If unconstrained, this formulation would permit each cell to receive up to eight steps of refinement. To ensure that each cell is split at most once, we must ensure that it receives positive refinement only from one of its incident vertices, which in turn means that the $2 \times 2 \times 2$ minors centered at two grid vertices that carry positive refinement can never overlap. Furthermore, to fulfill pairing, also partial tangencies between vertex

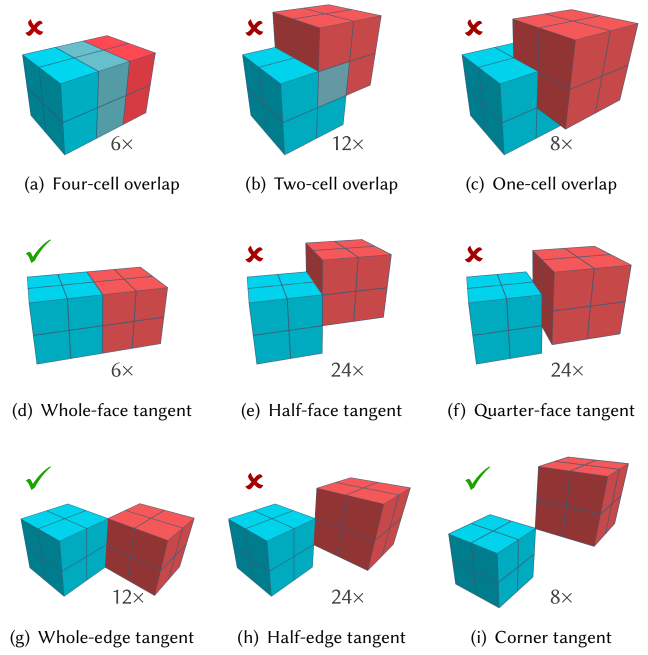


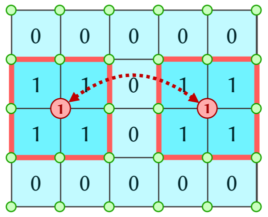
Fig. 6. Exhaustive taxonomy of all possible conflicts between the $2 \times 2 \times 2$ minors associated to the vertices in a regular 3D grid. The configurations in (a), (b), (c), (e), (f), and (h) are illegal, in (d), (g), and (i) are legal. The number of combinations for each configuration, taking into account symmetry, is indicated in each subfigure.

minors must be avoided because they introduce odd (unit length) sides. Fig. 6 shows all the possible interactions between overlapping or tangent vertex minors, indicating the illegal ones. In terms of ILP, avoiding all the illegal configurations in a 3D grid translates to 98 linear constraints of the type

$$r(v_i) + r(v_j) \le 1 \tag{3}$$

where v_i, v_j represent the vertices corresponding to centers of the offending minors.

These constraints alone, however, are still not sufficient to ensure a valid output grid. To understand this, one must recall that existing dual schemes for conforming hexahedral meshing are always positioned at the coarse side of a transition (Fig. 3), meaning that each cluster of refined elements must be surrounded by a buffer layer of unrefined cells that host the necessary transitions. If two refined areas are separated by a coarser narrow bridge of width one, it would be impossible to install the two schemes necessary for both clusters. We handle this situation by introducing additional linear constraints between the centers of potentially offending clusters, ensuring that only one of them has positive refinement (see inset). In 3D, this translates to 210 pairwise linear constraints per vertex, of the same type of those in Eq. 3.



Summarizing, the complete formulation for binary grids becomes

$$\min_{r(v)} E = \sum_{c \in G} \left(\sum_{v \in c} r(v) - r(c) \right) \tag{4}$$

$$s.t.$$

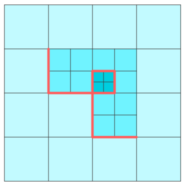
$$\forall c \in G, \quad \sum_{v \in c} r(v) \geq r(c)$$

$$\forall ij \in N_P \quad r(v_i) + r(v_j) \leq 1$$

where $r(v)$ are the unknown per-vertex binary refinements, $r(c)$ is the known input per-cell refinement, and N_P is the set of all vertices that are adjacent according to the conflicting masks shown in Fig. 6 and in the inset above.

5 ADAPTIVE GRIDS

In adaptive grids, clusters of cells with equal size and an odd number of items along one of their sides may occur at any level of the refinement hierarchy (red edges in the inset below), and not only at the boundary between cells with refinement 0 and 1, as it happens for regular binary grids. This makes it difficult to encode the pairing



problem in a single ILP formulation, as we did for the regular binary formulation. We face this issue by splitting global pairing into a sequence of simpler local problems. By restricting our analysis to the portion of the grid involving pairs of adjacent refinement levels in the hierarchy, we can operate on a temporary regular binary grid, and solve the pairing there. Considering an adaptive grid G with refinement levels $l_{\max}, \ldots, l_{\min}$, we create a sequence of binary sub-grids involving cells with refinement

$$(l_{\max}, l_{\max-1}), (l_{\max-1}, l_{\max-2}), \ldots, (l_{\min+1}, l_{\min}),$$

and apply the formulation described in Section 4 to each of them, updating the global grid after each solve. Note that, depending on the refinement pattern, the sub-grid extracted at each level may contain holes or may be composed of multiple connected components. Our approach correctly handles, without modification, all such configurations. For any pair of refinement levels (l_{i+1}, l_i), the local solve achieves pairing by promoting some of the cells with refinement l_i to level l_{i+1}. Proceeding from the finer to the coarser levels ensures the convergence of the process. Since no new cells with refinement greater than l_{i+1} are created, the boundaries involving level l_{i+1} are fully solved for in the current iteration. A simple example of our iterative pairing procedure is shown in Fig. 8.

5.1 Optimality and independence of local solves

The proposed iterative formulation is globally optimal if, after having solved each local ILP, no refined cell that is completely internal in the global grid has faces exposed at the boundaries of the local sub-grid. In fact, if this condition holds, each local grid becomes completely disjoint from the others. Note that all sub-grids undoubtedly observe this condition before solving the ILP. Otherwise, cells with refinement l_{i+1} would be face-adjacent to cells having refinement l_{i-1} in the global grid, violating the balancing condition and leading to a contradiction. However, there are cases where local pairing will demand splitting cells having boundary faces, creating an overlap with the local grid at the subsequent iteration, hence a connection between the associated ILPs. Besides, whenever this happens, the balancing criterion is violated. For this reason, we interleave global balancing and local pairing, restoring the former with the algorithmic approach described in Section 3.

5.2 Balancing policy and local grid boundary

As reported in Section 3, literature offers two alternative balancing criteria, which both support conforming hexahedral meshing and are compatible with our approach. Strong balancing adopts a broader definition of neighborhood, deeming any two cells sharing a face, edge, or vertex as adjacent. Weak balancing uses a stricter definition, for which only cells sharing a face are adjacent. These definitions impact our iterative pipeline because, depending on the chosen balancing policy, cells with distant levels of refinement may or not touch the boundary of a local grid.

Strong balancing. If the strong balancing criterion is used, any cell in the local binary grid containing a boundary element will have zero refinement. Indeed, this is always the case; otherwise, there will be a refined cell in the local grid that is edge/vertex-adjacent to some twice coarser cell not in the current grid, thus violating the strong balancing and leading to a contradiction. Consequently, any possible growth of refined clusters necessary to achieve pairing will be strictly contained in the current grid, meaning that only internal grid vertices will receive positive refinement.

Weak balancing. If weak balancing is observed, there may be cells in the local grid that have one vertex or edge exposed on the boundary, and may therefore be adjacent to coarser cells in the global grid that are not part of the current computational domain. In this case, the ILP may restore pairing by assigning refinement to some boundary vertex in the local grid, which in turn demands to

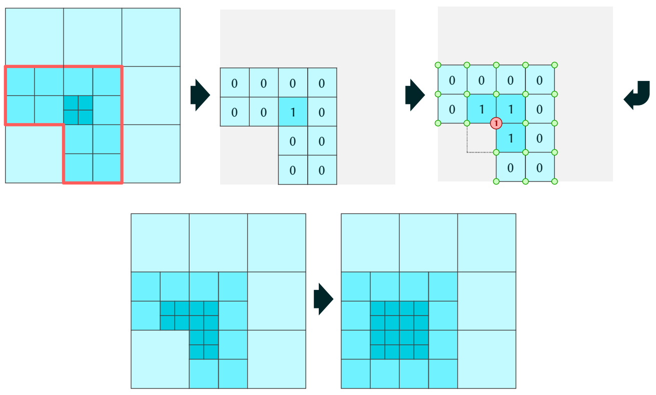


Fig. 7. Top: the ILP solver has assigned positive refinement to a boundary vertex in the local sub-grid. The minor associated to it must be enforced in the global grid, otherwise pairing is not guaranteed. Bottom: in 2D, a grid vertex may be incident to cells with up to three different levels of refinement, therefore two splits are enough to perform this operation. In 3D, a vertex may accommodate cells with up to four different levels of refinement, therefore the maximum number of necessary splits grows to three.

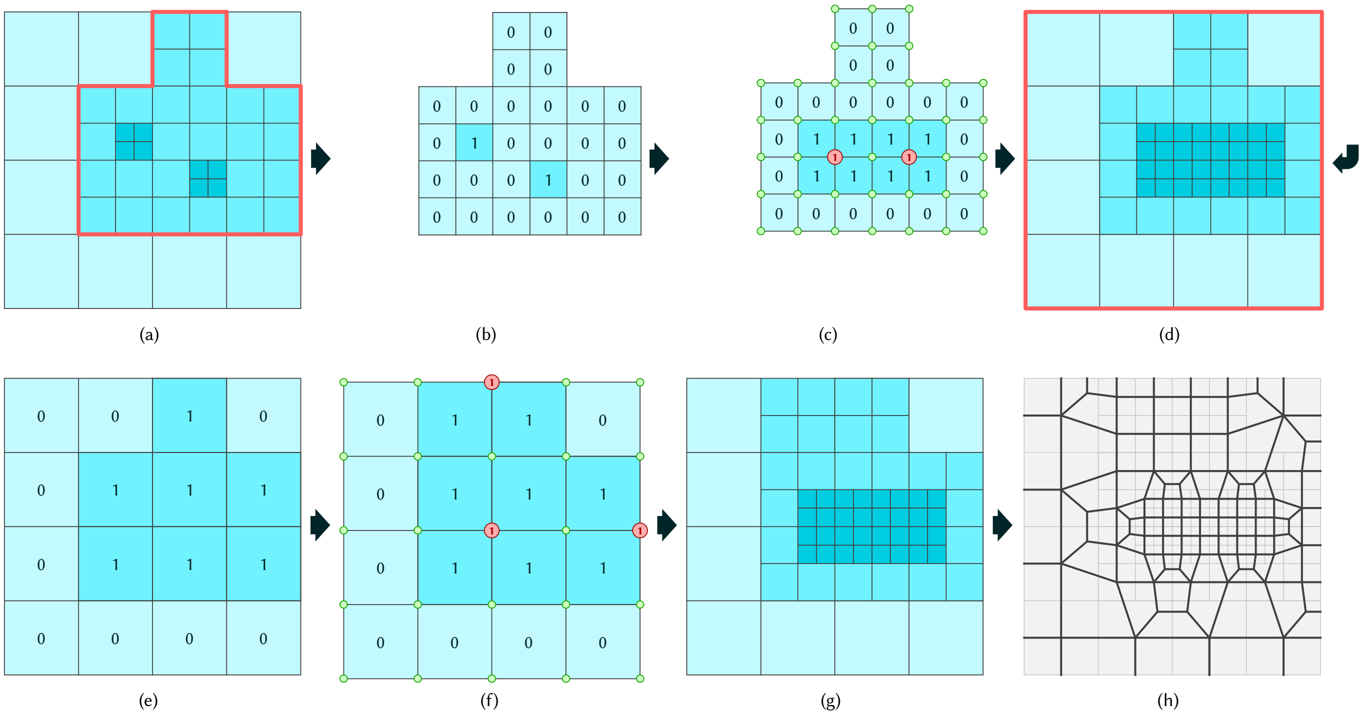


Fig. 8. Iterative pipeline for adaptive grid pairing. (a) input grid (pre-balanced), (b) regular sub-grid for levels of refinement 1, 2, (c) solution of the pairing in the regular sub-grid, (d) porting of the local solution in the global grid, (e) regular sub-grid for levels of refinement 0, 1, (f) solution of the pairing in the regular sub-grid, (g) porting of the local solution in the global grid, (h) output quadmesh.

ALGORITHM 1: Make an Adaptive Grid Hexmeshable

Input: an adaptively refined grid G, with $l_{\min}$, $l_{\max}$ being the minimum and maximum amount of cell refinement.

```
balancing(G);                                                   (Fig. 5)
for $i = l_{\max} - 1 \rightarrow l_{\min}$ do
    $G_i$ = binary regular subgrid($i$);                         (Fig. 8)
    pairing($G_i$);
    if some boundary cell gets refined then
        if boundary vertex in local solution then
            impose vertex minor in the global grid;             (Fig. 7)
        end
        balancing($G$);                                         (Fig. 5)
    end
end
return $G$;
```

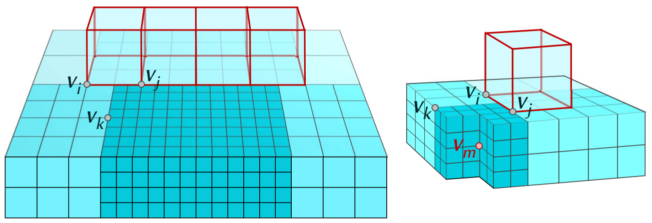


Fig. 9. Areas with refinement n and areas with refinement $n-2$ sharing the same edge do not align, making the installation of transition schemes in [Livesu et al. 2021] not applicable (left). We avoid this problem by imposing, only when this problem occurs, the refinement of the vertex V_m that controls the minor of refinement n (right) to be equal to zero. Vertices V_i, V_j and V_k highlight the correspondences between the two images.

resolve the overlap between the local solution and the global grid by splitting the coarser cells incident at such vertex (Fig. 7). This issue can only arise at the concavities of the local sub-grid. In our algorithm, we include an explicit check to correctly update the grid should this happen.

Sub-grid alignment. Weak balancing allows elements with three alternative levels of refinement to be adjacent to the same grid edge. Our local formulation considers only two levels of refinement per iteration. Because of this disparity, clusters of elements separated by two levels of refinement may be misaligned. Note that the resulting grid would still be perfectly balanced and paired, hence theoretically suitable for conforming hexahedral meshing, but the schemes in [Livesu et al. 2021] do not handle this case. This issue can be addressed either by extending the schemes in [Livesu et al. 2021], essentially shifting them by one position along the concave edge where the three levels of refinement meet, or by adding one extra constraint to preserve alignment across multiple iterations. We opted for the latter solution and we proceed as described in Fig. 9.

Complete pipeline. The pseudo-code in Algorithm 1 summarizes our entire pipeline. All in all, the choice of the strong balancing policy simplifies each iteration, as boundary vertices in the local solution do not arise. At the same time, strong balancing is known to introduce unnecessary refinement [Livesu et al. 2021], and is therefore sub-optimal in terms of output grid size. In our experience, we noticed that the additional checks required for the weak balancing policy produce a marginal overhead in the computation, while securing much coarser grids. Therefore, while remaining compatible with all methods in the prior art, we used the weak balancing policy in all our experiments.

6 RESULTS AND APPLICATIONS

To validate our approach, we implemented a C++ prototype of our grid processing algorithm, using CinoLib [Livesu 2019] for mesh processing and Gurobi [2020] to solve ILPs. To grant full reproducibility, upon acceptance, we will release both the source code and the data of all our tests to the public domain, in the Github repository `github.com/cg3hci/Gen-Adapt-Ref-for-Hexmeshing`. Output meshes (attached to this submission as additional material) will also be uploaded in the Hexalab online repository [Bracci et al. 2019].

To illustrate the flexibility of our method, in the following we evaluate it in two different settings. The first use case concerns its usage for guaranteeing the hexmeshability of an adaptive grid constructed using octree-based refinement in object-space (Section 6.1). Instead, the second use case targets the generation of hexmeshes from polycube mappings (Section 6.2).

6.1 Octree methods

These methods employ an octree to define an adaptively refined grid that well approximates the target geometry according to some geometric criterion, and eventually exploit the same hierarchical structure to enforce balancing and pairing (Section 1). They obtain a conforming hexahedral mesh by applying dual transition schemes first, and then projecting the boundary onto the target shape with various geometric approaches [Gao et al. 2019; Lin et al. 2015]. We compared our technique with prominent methods in this category, substituting the octree with our ILP formulation for the enforcement of the pairing criteria. In our comparison, we considered both well-established state-of-the-art methods using strongly balanced grids [Gao et al. 2019; Maréchal 2009] and their recently introduced extension to weakly balanced grids [Livesu et al. 2021].

Experimental setup. While commercial and academic software is available for each of the techniques mentioned above, each method uses different geometric criteria for octree splitting, and would ultimately produce a different starting grid. We implemented a standard octree to secure a fair comparison, initializing the root to the smallest cube containing the input object. We then iteratively split octants until the local grid size was lower than half the local shape thickness, measured with SDF [Shapira et al. 2008]. We used the so generated grid as input for all methods, and reproduced results of Maréchal [2009] and Gao et al. [2019] by applying strong balancing and pairing *through* the octree hierarchy, and results in Livesu et al. [2021] by applying weak balancing and the same octree pairing approach. For our method, we applied the iterative strategy detailed in Section 5, using a weak balancing policy. As a result, we obtained three adaptive grids, that we converted into conforming hexahedral meshes with the schemes proposed by Livesu et al. [2021], which are implemented in CinoLib [Livesu 2019].

	Input	**OP + SB**			**OP + WB**			**Ours**		
	size	Δ_{abs}	Δ_{rel}	$\times$	Δ_{abs}	Δ_{rel}	$\times$	Δ_{abs}	Δ_{rel}	$\times$
Avg	12K	24,4	231%	3.3×	19,9	186%	2.9×	13,6	116%	2.1×
Max	128K	213,4	802%	9.0×	176,2	589%	6.9×	151,5	451%	5.5×

Table 1. Cumulative statistics for all meshes in our benchmark. We report the initial grid cell count, and the absolute growth (Δ_{abs}), relative growth (Δ_{rel}), and increase factor (×) of methods based on Octree Pairing (OP), both with Strong Balancing (SB) and Weak Balancing (WB), and our method. The absolute growth is the number of extra cells added to the initial grid to fulfill hexmeshability criteria; the relative growth is the ratio of absolute growth to the initial grid size; the increase factor is the ratio of the final cell count to initial cell count.

Dataset and comparisons. We considered all the models in the dataset released by Gao et al. [2019], which comprises 202 organic and CAD models of varying geometric and topological complexity.

We compare output grids in terms of their number of cells. Note that, depending on the application, both the interior and the exterior of the input object may be relevant. For example, structural FEM analysis is mostly concerned with the interior, whereas CFD is often used to simulate air and liquid dynamics *around* the object (e.g. a plane or a boat). For this reason, in our comparison we always consider the total number of grid elements, and not the subset of elements strictly contained in the input shape.

In two cases, the input octree was already suitable for hexahedral meshing, and all methods introduced no further refinement. In six cases, the three methods introduced exactly the same amount of refinement, producing the same output grid. Note that all these cases correspond to overly simple shapes, which are either perfect cubes or strongly resemble a cube (Fig. 11). In all the remaining cases (194 out of 202), our method clearly outperforms prior art, producing grids with much lower element count (Fig. 10). Remarkably, our performances were consistent across all models in the dataset, and we never produced a hexmesh with more elements than the ones produced with competing solutions [Gao et al. 2019; Livesu et al. 2021; Maréchal 2009].

Comparing input and output grid sizes, it becomes evident that, in terms of mesh resolution, the real bottleneck in the hexahedral meshing pipeline is not shape complexity, but rather the refinement necessary to secure the fulfillment of the conforming hexmeshing criteria (Table 1). For classical octree methods based on strong balancing [Gao et al. 2019; Maréchal 2009], in 88% of the cases the input grid size was more than doubled; in 58% of the cases the size was more than tripled; in 26% of the cases it was more than quadrupled, and in one case the output grid was more than 9 times bigger than the input! The recently proposed weak balancing [Livesu et al. 2021] alleviates mesh growth, but it is still based on rigid octrees: in 86% of the cases grid size was at least doubled; in 43% of the cases it was at least tripled, and in 1% of the cases it was more than quadrupled. By unlinking pairing from rigid octrees, our method performed much better, and it at least doubled grid size in 63% of the cases; tripled it in 6% of the cases, and only in one case it produced a mesh with more than triple size. In Fig. 12 we directly compare the three

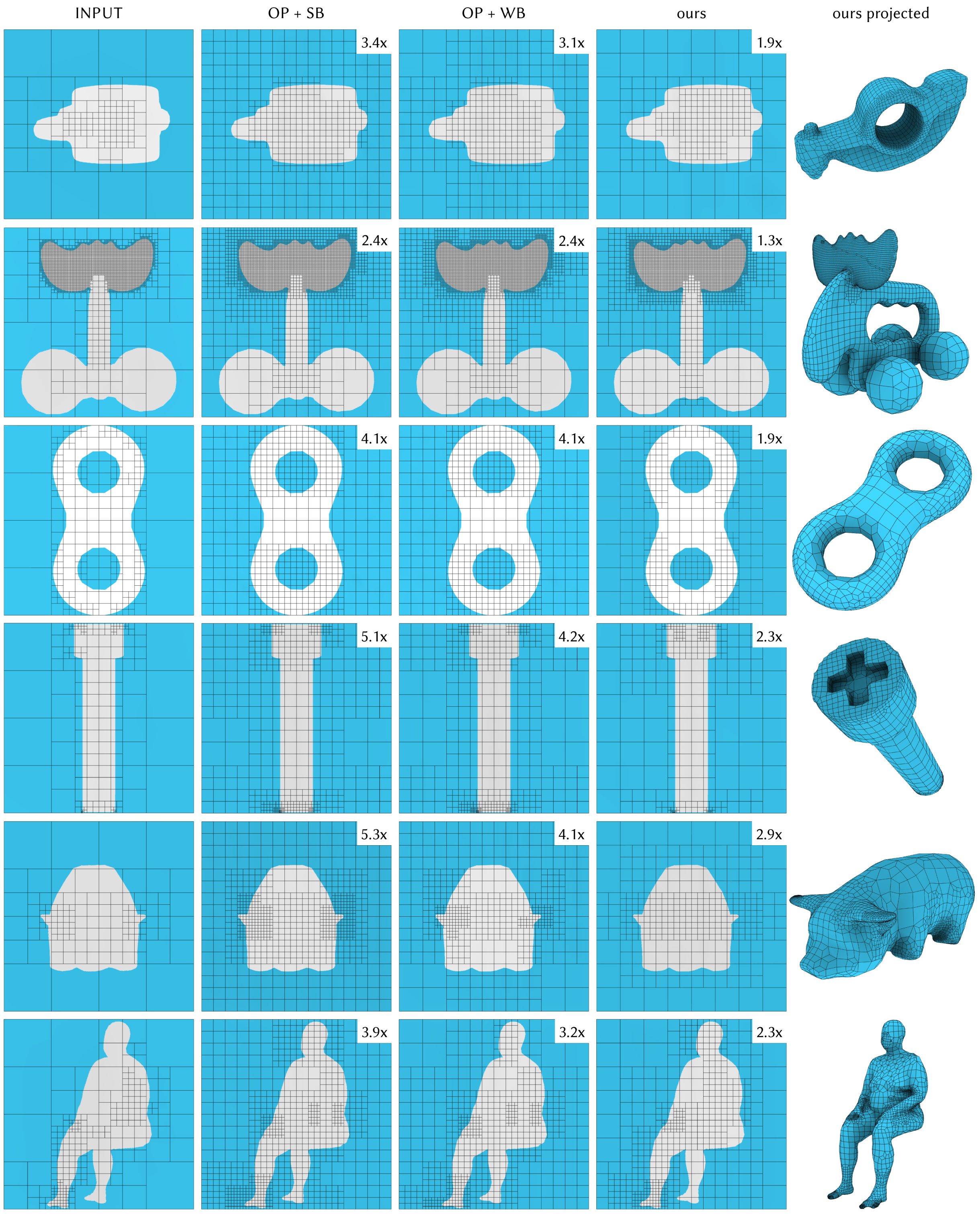


Fig. 10. A random subset of models in the benchmark released by Gao et al. [2019]. From left to right, for each model we show: the input adaptive grid; the grid obtained with octree pairing (OP) and strong balancing (SB) [Gao et al. 2019; Maréchal 2009]; the grid obtained with octree pairing and weak balancing [Livesu et al. 2021]; our grid, and the final mesh. For each grid we report the growth factor in the to upper-right corner, measured as the ratio between grid size and input grid size.

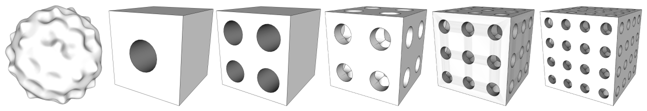
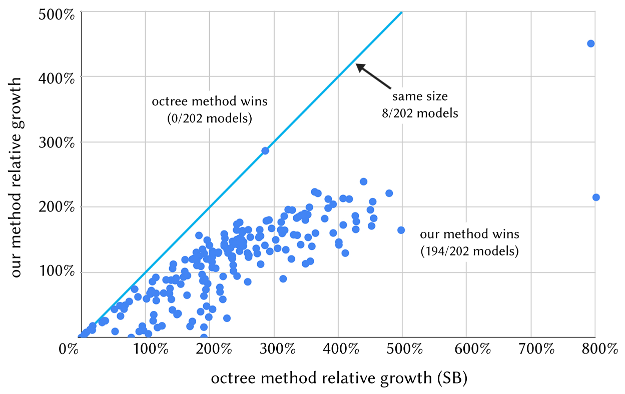


Fig. 11. Six out of the eight models from the dataset released with Gao et al. [2019], where our method and octree-based methods produced exactly the same result. We do not show the other two models because they are duplicates. Note that almost all meshes are perfect cubes, hence nicely fit the structure imposed by a regular octree. For the remaining 194 models, our method consistently produced much coarser meshes.

approaches in terms of relative growth w.r.t. input grid size. We can conclude that, on the testing dataset, our method almost halved the number of extra cells required by octree approaches. Considering that numerical methods that use these meshes have a complexity that may be quadratic in the number of degrees of freedom, halving grid size leads to significant benefits in terms of computational cost.

Geometric projection. While the presented comparisons are purely based on mesh size, to produce the meshes shown in Fig. 1 and Fig. 10 we had to project the boundary of the hexmesh to the target geometry. These aspects are orthogonal to the problem tackled in the paper, but, for completeness, we report here the simplified heuristics used for generating our results. We attempted to relocate each boundary vertex to its closest point on the input geometry, and used binary search to iteratively revert the move if some incident hexahedral element flipped its orientation. We repeated this process for 20 iterations and eventually applied one step of edge-cone rectification [Livesu et al. 2015] to smooth the geometry and maximize minimum per-element quality. Sharp creases in the input mesh were detected based on dihedral angle thresholding (60°), and mapped to the edges of the hexmesh as described in Gao et al. [2019]. Note that this projection system is overly simple and may fail to produce a satisfactory result in complex cases. Nevertheless, our contributions are purely on the combinatorial side of the hexmeshing pipeline, and our method could be coupled with any of the existing algorithms for robust hexmesh boundary projection, possibly increasing geometric fidelity [Gao et al. 2019; Lin et al. 2015].

Scalability. Fig. 13 reports the running times for our algorithm as a function of the size of the input grid in number of cells for each of the benchmark meshes. All experiments have been performed on a PC with an Intel Core i9 2.90GHz processor with 128GB RAM. The timing results demonstrate that the running time is growing approximately linearly with the number of elements in the input mesh, with a median speed of 710 input cells/s (1644 output cells/s). As expected, the running times are dominated by pairing, which takes over 90% of the total running time. Around half of the pairing time is spent setting up the problem and updating the grids, while the other half is spent inside the Gurobi solver (47% as median value on all benchmark meshes). Note that the size of each ILP depends on the number of elements with same refinement, hence it is only indirectly affected by the global size of the input grid. Our current implementation is not optimized, and we foresee interesting avenues to further reduce running times. In particular, at the moment, the solver is not warm started, and we conjecture that initializing it with a feasible solution close to the input data could reduce running times by a significant amount.



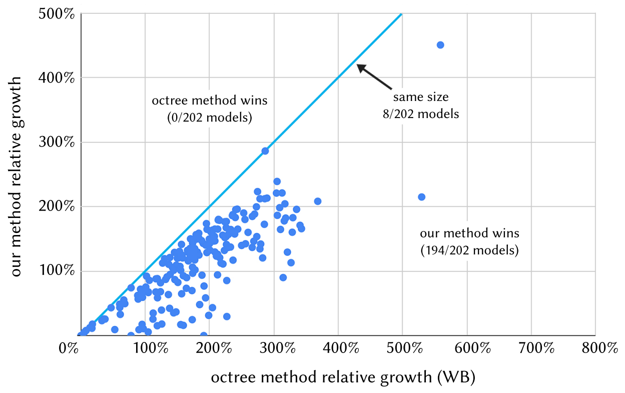


Fig. 12. Relative growth of our method compared to the relative growth of the octree method using strong balancing [Gao et al. 2019; Maréchal 2009] (top), and weak balancing [Livesu et al. 2021] (bottom). Each blue point is a mesh from the dataset released By Gao et al. [2019], where the horizontal axis is the growth obtained with octree refinement, and the vertical axis the growth achieved with our method. Except for 8 cases out of 202, where the three methods produced exactly the same grid (points on the diagonal lines), our growth is lower for all the remaining models. On average, we more than halve grid size with respect to state-of-the-art strong balancing methods [Gao et al. 2019; Maréchal 2009], and we almost halve grid size with respect to recently introduced weak balancing ones [Livesu et al. 2021].

6.2 Polycube methods

Besides octree-based methods that operate in object space, our algorithm can be directly installed into any pipeline that leverages Cartesian grids to generate the hexmesh connectivity, securing mesh adaptivity with minimal element count. We demonstrate this ability by combining polycube-based hexmeshing with our approach.

Polycube methods proceed by mapping a target shape onto an orthogonal polyhedron (or *polycube* [Tarini et al. 2004]), where they generate the mesh connectivity, typically with a regular grid sampling. The vertices of the so generated mesh are then mapped back to the object space through the inverse map, producing the output hexahedral mesh [Fang et al. 2016; Fu et al. 2016; Huang et al. 2014; Livesu et al. 2013]. In a sense, these methods can be seen as an extension of traditional grid-based techniques [Schneiders 1996], with

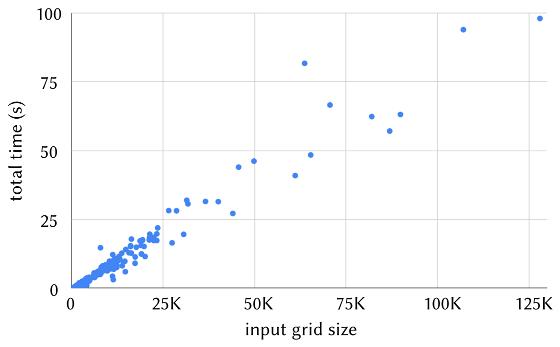


Fig. 13. Running time in seconds (vertical axis) of our algorithm as a function of the size of the input grid in number of cells (horizontal axis).

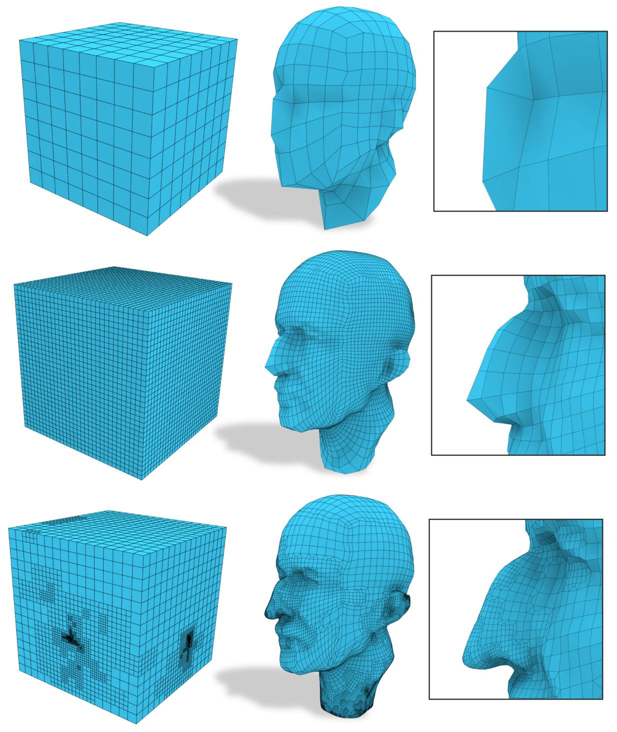


Fig. 14. Mapping a relatively simple shape to a coarse polycube inevitably introduces compression and stretching. Regardless of the sampling frequency, a regular sampling does not compensate the map distortion, and the resulting mesh lacks geometric fidelity (top, middle). By coupling our generalized adaptive sampling with polycube techniques, we can produce meshes with comparable size but much higher geometric fidelity (bottom). Input polycube map courtesy of Aigerman and Lipman [2013]

the only difference that they operate in a carefully constructed parametric space, and perform the final mapping through a volumetric parameterization, and not through a geometric approach.

Compared with octree-based approaches, polycube methods are often able to produce hexahedral meshes with a much cleaner structure and higher geometric quality [Gregson et al. 2011; Livesu et al. 2020]. Nevertheless, current polycube techniques are much less

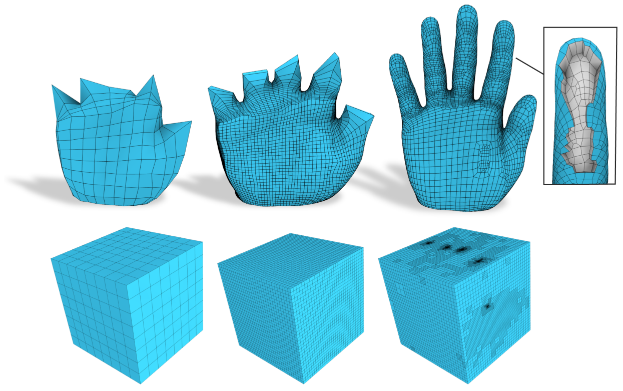


Fig. 15. Mapping a complex shape to a coarse polycube introduces extreme distortion, at the point that major features in the input shape cannot be recovered with a regular sampling (left, middle). We adaptively refined the polycube grid with our method, splitting hexahedra until the distance in the input shape of any two points that map inside the same hexahedron was smaller than 1/50 of the bounding box diagonal. With this approach, we were able to introduce all the necessary elements and singular vertices to reproduce all fingers, while bounding the global mesh size (right). The output mesh counts only 25K hexahedra, and despite the huge map distortion, the average and minimum scaled Jacobian are 0.65 and 0.1, respectively. The mesh is therefore perfectly suitable for analysis. The input polycube map was initialized with PolyCut [Livesu et al. 2013]; flipped tetrahedral elements were fixed with the approach of Garanzha et al. [2021]. The output hexmesh was eventually smoothed with Edge-Cone Rectification [Livesu et al. 2015].

robust and make strong assumptions on the underlying polycube structure and volumetric map. In fact: (i) polycubes are expected to explicitly encode all the relevant features of the object, ensuring that the mesh connectivity adapts to them; (ii) the map is assumed to be *ideal*, meaning that it does not contain significant distortion, so that a regular grid sampling in polycube space translates to a uniform hexahedral mesh. These two requirements are unrealistic in many practical scenarios, and the latter is even intrinsically impossible to achieve if the input object is a triangle-like or cone-like shape, which could never be mapped to a cuboidal domain without severe geometric compression or stretching.

By combining polycube mappings with our adaptive sampling strategy, we were able to produce quality meshes even starting from very coarse polycubes and highly distorted maps. In Fig. 14 and Fig. 15 we show two extreme examples produced with our hybrid approach, where we mapped the Max Planck head and a hand to the simplest possible polycube: a cube. As shown in the figures, a regular grid sampling would easily produce a mesh that is geometrically unfaithful or lacks important features. Our adaptive approach allowed us to densely sample the portions of the cube where many elements in the input mesh collapsed, counterbalancing map distortion and providing a valid mesh that is suitable for analysis. Since our technique does not rely on a rigid octree structure, it can be applied to any polycube, regardless of its geometric complexity or topology (Fig. 16).

While these are just two toy examples, we believe that this approach offers interesting directions for further research, as it clearly

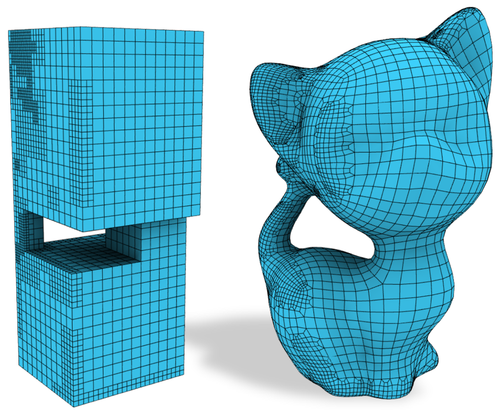


Fig. 16. Adaptively refined polycube map of a grid with non-trivial topology. Our method puts no constraints on the shape or topology of the grid. The original map was computed with PolyCut [Livesu et al. 2013] and then processed with our method.

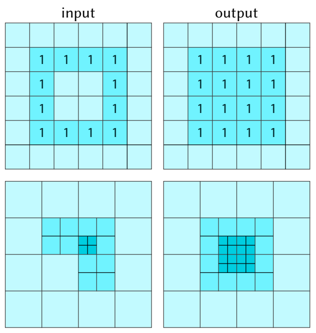
indicates that the total complexity of the polycube-based hexmeshing problem could be better distributed across the various building blocks of the pipeline. By being smarter in the grid sampling step, we can contextually release some of the strict (and sometimes impossible) requirements for the first parts, where the topology of the orthogonal polyhedron and the associated map are generated. Considering that the biggest obstacle for a wider (e.g., industrial) adoption of these techniques lies in the inability to deliver robust algorithms, this hybrid approach could help to increase robustness significantly.

7 CONCLUSION

In this work, we studied the topological mechanisms that permit transforming a generic adaptively refined grid into a grid that is compatible with conforming hexahedral meshing. As clearly shown in our experiments, minimizing the amount of refinement to achieve balancing and pairing is a true bottleneck for the hexmeshing pipeline. Furthermore, it is a surprisingly rich problem, which poses a variety of challenges that we exhaustively addressed in our formulation. Prior works based on octree splitting rules work in a very tight space of solutions. They are fast and easy to implement, but often impose a huge amount of unnecessary refinement.

In this article, we have proposed a novel formulation that significantly enlarges the space of hexmeshable adaptive grids by moving from the application of fixed hierarchical rules to the definition of ILPs. While our approach is a bit slower and less straightforward to implement than previous octree-based solutions, it outperforms them in any other aspect, producing much coarser grids that ensure better performances during further mesh processing. Considering the typical applications in which hexahedral meshes are involved, where mesh generation is run once and for all on high-performance machines, and computation time is less important than the quality of the output, we believe that our findings provide neat advantages and will be readily adopted by practitioners in the field.

Even though our method proved to be superior than previous solutions, there is still space for further improvement. In particular, even though we operate in a bigger space than octree-based methods, our vertex-based formulation does not cover the full set of valid solutions. A failure example is shown in the inset aside (top), where a closed ring of refined elements is thickened because it has uneven width. This refinement is unnecessary because the ring is closed and no side with an odd size is exposed, but our local formulation is not able to catch the global arrangement in a closed cycle. Interestingly, our iterative solution based on local sub-grids may produce closed rings with unit width (inset, bottom), but only if the ring contains coarser elements at one side and finer elements at the other side. This is because we process grid elements from finer to coarser, and local grids may partially overlap. We conjecture that this limitation could be fully addressed by using an alternative formulation that assigns refinement through edges instead of vertices. However, the grid minor associated with an edge has size 2×1 in 2D and $2 \times 2 \times 1$ in 3D, hence there is an odd side which makes the fulfillment of the pairing condition problematic. At present, we cannot guarantee that such a formulation does or does not exist, and we leave this as a direction for future works. Developing a single-pass global solver would permit us to always converge to the global optimum. However, we have shown that expressing the solution in terms of variables associated with the input grid elements seems impossible, because some relevant pieces of the grid will arise only after refinement, suggesting that an interleaved approach that updates the grid in between one solve and the other may be unavoidable. We do not have formal proof for this, but this seems a major obstacle that is hard to overcome. Finally, we believe our generalized approach could be extended to alternative techniques that use Cartesian grids to generate all-hex connectivities, such as methods based on frame fields [Liu et al. 2018]. The extension is non-trivial though, because rotational transitions that directly connect cells not face-adjacent in the grid must be taken into account, also addressing the singularities they generate in the final mesh.



Acknowledgments

The authors are deeply grateful to François Protais, Nicolas Ray, and Dmitry Sokolov for the polycube map of the hand in Fig. 15. Gianmarco Cherchi gratefully acknowledges the support to his research by PON R&I 2014-2020 AIM1895943-1. Authors at CRS4 acknowledge the contribution of Sardinian Regional Authorities under project VIGECLAB and for the CRS4 funding of PhD Projects. Riccardo Scateni was partially funded by ADAM project - CUP F74I19000900007 (FdS, annualità 2018). The work of Marco Livesu was partly supported by EU ERC Advanced Grant CHANGE No. 694515.

REFERENCES

Noam Aigerman and Yaron Lipman. 2013. Injective and bounded distortion mappings in 3D. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–14.

Cecil G Armstrong, Harold J Fogg, Christopher M Tierney, and Trevor T Robinson. 2015. Common themes in multi-block structured quad/hex mesh generation. *Procedia Engineering* 124 (2015), 70–82.

Arthur Bawin, François Henrotte, and Jean-François Remacle. 2020. Automatic feature-preserving size field for 3D mesh generation. arXiv:2009.03984 [math.NA]

Steven E Benzley, Ernest Perry, Karl Merkley, Brett Clark, and Greg Sjaardama. 1995. A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis. In *Proceedings, 4th international meshing roundtable*, Vol. 17. Citeseer, 179–191.

Ted Blacker. 2000. Meeting the challenge for automated conformal hexahedral meshing. In *Proc. 9th international meshing roundtable*. Springer, 11–20.

Matteo Bracci, Marco Tarini, Nico Pietroni, Marco Livesu, and Paolo Cignoni. 2019. HexaLab.net: an Online Viewer for Hexahedral Meshes. *Computer-Aided Design* 110 (2019), 24–36. https://www.hexalab.net/

Gianmarco Cherchi, Pierre Alliez, Riccardo Scateni, Max Lyon, and David Bommes. 2019. Selective Padding for Polycube-Based Hexahedral Meshing. *Computer Graphics Forum* 38, 1 (2019), 580–591. https://doi.org/10.1111/cgf.13593

Gianmarco Cherchi, Marco Livesu, and Riccardo Scateni. 2016. Polycube Simplification for Coarse Layouts of Surfaces and Volumes. *Computer Graphics Forum* 35, 5 (2016), 11–20.

AO Cifuentes and A Kalbag. 1992. A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis. *Finite Elements in Analysis and Design* 12, 3-4 (1992), 313–318.

Etienne Corman and Keenan Crane. 2019. Symmetric moving frames. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 87:1–87:16.

Distene SAS. 2020. *MeshGems*. http://www.meshgems.com/volume-meshing-meshgems-hexa.html

Thomas Nelson Erke Wang and Rainer Rauch. 2004. Back to elements-tetrahedra vs. hexahedra. In *Proc. International ANSYS conference*.

Xianzhong Fang, Weiwei Xu, Hujun Bao, and Jin Huang. 2016. All-hex meshing using closed-form induced polycube. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 124:1–124:9.

Xiao-Ming Fu, Chong-Yang Bai, and Yang Liu. 2016. Efficient volumetric polycube-map construction. *Computer Graphics Forum* 35, 7 (2016), 97–106.

Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. 2019. Feature Preserving Octree-Based Hexahedral Meshing. *Computer Graphics Forum* 38, 5 (2019), 135–149.

Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, and Dmitry Sokolov. 2021. Foldover-free maps in 50 lines of code. *arXiv preprint arXiv:2102.03069* (2021).

James Gregson, Alla Sheffer, and Eugene Zhang. 2011. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum* 30, 5 (2011), 1407–1416.

LLC Gurobi Optimization. 2020. *Gurobi Optimizer Reference Manual*. http://www.gurobi.com

Kangkang Hu, Jin Qian, and Yongjie Zhang. 2013. Adaptive all-hexahedral mesh generation based on a hybrid octree and bubble packing. In *Proc. 22nd International Meshing Roundtable - Research Notes*. Sandia National Laboratories, 5B3:1–5B3:5.

Kangkang Hu and Yongjie Jessica Zhang. 2016. Centroidal Voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation. *Computer Methods in Applied Mechanics and Engineering* 305 (2016), 405–421.

Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. 2014. L1-based construction of polycube maps from complex shapes. *ACM Transactions on Graphics (TOG)* 33, 3 (2014), 25:1–25:11.

Yasushi Ito, Alan M Shih, and Bharat K Soni. 2009. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *Internat. J. Numer. Methods Engrg.* 77, 13 (2009), 1809–1833.

Tengfei Jiang, Jin Huang, Yuanzhen Wang, Yiying Tong, and Hujun Bao. 2013. Frame field singularity correctionfor automatic hexahedralization. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (2013), 1189–1199.

Michael Kremer, David Bommes, Isaak Lim, and Leif Kobbelt. 2014. Advanced automatic hexahedral mesh generation from surface quad meshes. In *Proc. 22nd International Meshing Roundtable*. Springer, 147–164.

Franck Ledoux and Jean-Christophe Weill. 2008. An extension of the reliable whisker weaving algorithm. In *Proc. 16th International Meshing Roundtable*. Springer, 215–232.

Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. 2012. All-hex meshing using singularity-restricted field. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 177:1–177:11.

Hongwei Lin, Sinan Jin, Hongwei Liao, and Qun Jian. 2015. Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm. *Computer-Aided Design* 67 (2015), 107–117.

Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. 2018. Singularity-constrained octahedral fields for hexahedral meshing. *ACM Transations on Graphics (TOG)* 37, 4 (2018), 93:1–93:17.

Marco Livesu. 2019. Cinolib: A Generic Programming Header Only C++ Library for Processing Polygonal and Polyhedral Meshes. *Transactions on Computational Science XXXIV* 34 (2019), 64–76. https://github.com/mlivesu/cinolib/.

Marco Livesu, Alessandro Muntoni, Enrico Puppo, and Riccardo Scateni. 2016. Skeleton-driven adaptive hexahedral meshing of tubular shapes. *Computer Graphics Forum* 35, 7 (2016), 237–246.

Marco Livesu, Nico Pietroni, Enrico Puppo, Alla Sheffer, and Paolo Cignoni. 2020. LoopyCuts: practical feature-preserving block decomposition for strongly hex-dominant meshing. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 121:1–121:17.

Marco Livesu, Luca Pitzalis, and Gianmarco Cherchi. 2021. Optimal Dual Schemes for Adaptive Grid Based Hexmeshing. arXiv:2103.07745 [cs.GR]

Marco Livesu, Alla Sheffer, Nicholas Vining, and Marco Tarini. 2015. Practical Hex-Mesh Optimization via Edge-Cone Rectification. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 141:1–141:11.

Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. PolyCut: monotone graph-cuts for PolyCube base-complex construction. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 171:1–171:12.

Loïc Maréchal. 2009. Advances in octree-based all-hexahedral mesh generation: handling sharp features. In *Proc. 18th international meshing roundtable*. Springer, 65–84.

Scott A Mitchell and Stephen A Vavasis. 1992. Quality mesh generation in three dimensions. In *Proc. 8th annual symposium on Computational geometry*. 212–221.

Steven J Owen. 1998. A survey of unstructured mesh generation technology. In *Proc. 9th international meshing roundtable*. 267–291.

Nicolas Ray, Dmitry Sokolov, Maxence Reberol, Franck Ledoux, and Bruno Lévy. 2018. Hex-dominant meshing: mind the gap! *Computer-Aided Design* 102 (2018), 94–103.

Teseo Schneider, Yixin Hu, Xifeng Gao, Jeremie Dumas, Denis Zorin, and Daniele Panozzo. 2019. A Large Scale Comparison of Tetrahedral and Hexahedral Elements for Finite Element Analysis. arXiv:1903.09332 [cs.NA]

Robert Schneiders. 1996. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with computers* 12, 3-4 (1996), 168–177.

Robert Schneiders. 1997. An algorithm for the generation of hexahedral element meshes based on an octree technique. In *Proc. 6th International Meshing Roundtable*. 195–196.

Robert Schneiders. 2000a. Algorithms for quadrilateral and hexahedral mesh generation. In *Proc. VKI Lecture Series on Computational Fluid Dynamic*. 2000–2004.

Robert Schneiders. 2000b. Octree-based hexahedral mesh generation. *International Journal of Computational Geometry & Applications* 10, 4 (2000), 383–398.

Robert Schneiders and Rolf Bünten. 1995. Automatic generation of hexahedral finite element meshes. *Computer Aided Geometric Design* 12, 7 (1995), 693–707.

Robert Schneiders, Roland Schindler, and Frank Weiler. 1996. Octree-based generation of hexahedral element meshes. In *Proc. 5th International Meshing Roundtable*. Sandia National Labs, 205–216.

Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. 2008. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer* 24, 4 (2008), 249.

Jason F Shepherd and Chris R Johnson. 2008. Hexahedral mesh generation constraints. *Engineering with Computers* 24, 3 (2008), 195–213.

Matthew L Staten, Robert A Kerr, Steven J Owen, Ted D Blacker, Marco Stupazzini, and Kenji Shimada. 2010. Unconstrained plastering—Hexahedral mesh generation via advancing-front geometry decomposition. *Internat. J. Numer. Methods Engrg.* 81, 2 (2010), 135–171.

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. Polycube-maps. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 853–860.

Timothy J Tautges. 2001. The generation of hexahedral meshes for assembly geometry: survey and progress. *Internat. J. Numer. Methods Engrg.* 50, 12 (2001), 2617–2642.

Wei Wang, Yong Cao, and Tsubasa Okaze. 2021. Comparison of hexahedral, tetrahedral and polyhedral cells for reproducing the wind field around an isolated building by LES. *Building and Environment* 195 (2021), 107717.

Haiyan Wu, Shuming Gao, Rui Wang, and Jinming Chen. 2018. Fuzzy clustering based pseudo-swept volume decomposition for hexahedral meshing. *Computer-Aided Design* 96 (2018), 42–58.

Kaoji Xu, Xifeng Gao, Zhigang Deng, and Guoning Chen. 2017. Hexahedral meshing with varying element sizes. *Computer Graphics Forum* 36, 8 (2017), 540–553.

Yongjie Zhang and Chandrajit Bajaj. 2006. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer methods in applied mechanics and engineering* 195, 9-12 (2006), 942–960.