Bisection-based triangulations of nested hypercubic meshes

Kenneth Weiss[1] and Leila De Floriani[2]

[1] University of Maryland, College Park, USA kweiss@cs.umd.edu
[2] University of Genova, Genova, Italy deflo@disi.unige.it

Summary. Hierarchical spatial decompositions play a fundamental role in many disparate areas of scientific and mathematical computing since they enable adaptive sampling of large problem domains. Although the use of quadtrees, octrees, and their higher dimensional analogues is ubiquitous, these structures generate meshes with cracks, which can lead to discontinuities in functions defined on their domain. In this paper, we propose a dimension–independent triangulation algorithm based on regular simplex bisection to locally decompose adaptive hypercubic meshes into high quality simplicial complexes with guaranteed geometric and adaptivity constraints.

1 Introduction

Hypercubes are the basis for many adaptive spatial subdivisions, such as quadtrees in 2D, octrees in 3D and their higher dimensional analogues. Due to their simple definition in arbitrary dimension – any hypercube can be decomposed into 2^d hypercubes covering its domain – such decomposition schemes are widely implemented and have been successfully applied to problems across many different application domains [20].

However, hypercubic meshes do not generate *conforming*, i.e. crack-free, adaptive decompositions. This can be problematic, for example, when values are associated with the vertices of a mesh defining a discrete scalar function, since such cracks can lead to discontinuities in the function defined over the domain. In contrast, simplicial decompositions admit highly adaptive conforming meshes but can require many more cells to cover the same domain.

Downstream applications of these meshes, such as finite element analysis and scalar field modeling and visualization, often require mesh elements to satisfy certain quality constraints related to the shapes of the elements as well as the rate of adaptivity within the mesh. *Geometric* quality constraints can be enforced by using refinement rules that only generate mesh elements from a small set of acceptable modeling primitives [4]. A common *adaptivity* constraint is to ensure that neighboring elements differ in resolution by at most one refinement level, i.e. that the ratio of edge lengths between neighboring elements can be at most 2:1. This constraint

has been considered in several problem domains, including computational geometry [3, 6], scientific visualization [9, 32] and computer graphics [26] under various terms such as *restricted* [26, 23], *smooth* [15] and *balanced* [16, 24].

At the same time, there has been an increasing interest in the research on geometric modeling and scientific visualization to deal with data in medium and high dimensions [14, 5]. This is facilitated by the development of dimension–independent algorithms and data structures. Examples include time-varying volume data sets, which can be viewed as 4-dimensional data sets, and in general applications in four-dimensional space-time, when we consider 3D spatial entities moving over time. Another example is given by applications in six-dimensional phase-space, where data are characterized by three spatial coordinates and three momentum variables.

In this paper, we are concerned with the relationship between nested meshes formed by hypercubes, and their decompositions into high quality simplicial complexes. Specifically, we propose a simplicial decomposition for nested hypercubic meshes generated by applying a *regular simplex bisection* rule to locally triangulate each hypercube in the mesh. Due to the properties of this bisection scheme [13, 25, 4, 29, 31], our algorithm generates adaptive simplicial complexes composed of high quality simplices from a small set of geometric primitives. Furthermore, compatibility between adjacent triangulated hypercubes is implicitly enforced and the complexity of the mesh increases by a dimension–dependent multiplicative constant. Our algorithm depends only on local properties of the edges of the hypercubes in the mesh and can be easily incorporated into existing meshing systems based on nested hypercubic meshes, such as quadtrees, octrees and their higher dimensional counterparts.

The remainder of this paper is organized as follows. After a review of relevant background notions in Section 2, we discuss prior work on balanced hypercubic meshes and their triangulations in Section 3. In Section 4, we review properties of the regular simplex bisection scheme which will be relevant for our triangulation algorithm in Section 5. We discuss empirical results of our algorithm in the context of a 3D isosurfacing application in Section 6 and compare the meshes generated by our algorithm to those of an alternate 3D triangulation algorithm. Finally, we draw some concluding remarks in Section 7.

2 Background notions

In this Section, we review concepts related to polytopic meshes and to nested mesh refinement that we will use in the remainder of the paper.

2.1 Polytopic meshes

A *convex polytope* is a subspace of $\mathbb{R}^n$ bounded by a set of half-spaces. Polytopes generalize line segments (1-polytopes), polygons (2-polytopes) and polyhedra (3-polytopes).

We define a *d-cell* as a convex polytope in some d-dimensional subspace of $\mathbb{R}^n$. Let γ be a d-cell. Then, an i-dimensional *face* γ_i of γ, $0 \leq i \leq d$, is an i-cell on the boundary of γ. We refer to a 0-cell as a *vertex*, a 1-cell as an *edge*, and a $(d-1)$-cell as a *facet*.

A *polytopic mesh* $\mathcal{P}$ is a finite collection of polytopic cells such that (a) if γ is a cell in $\mathcal{P}$, then all faces γ_i of γ also belong to $\mathcal{P}$ (b) the interiors of cells in $\mathcal{P}$ are disjoint. The *dimension*, or *order*, of a polytopic mesh is the maximum of the orders of the cells forming it. In a polytopic mesh, cells that are not on the boundary of any other cells are called *top cells*. Also, in a polytopic mesh of order d with a manifold domain, as we will consider here, all top cells are d-cells. Such meshes are referred to as *pure*.

If, additionally, the intersection of any two cells $\gamma_1, \gamma_2 \in \mathcal{P}$ is a lower dimensional cell on the boundary of γ_1 and γ_2, then $\mathcal{P}$ is said to be *conforming*, or *compatible*. Conforming meshes are important in many applications since they ensure that there are no cracks or T-junctions between adjacent cells.

Hypercubic meshes

Hypercubes are the higher dimensional analogues of line segments (1-cubes), squares (2-cubes) and cubes (3-cubes). An interesting property of hypercubes is that all faces of a d-cube are lower dimensional hypercubes. Given a d-cube h, an *i-face* of h is any i-cube on the boundary of h, where $0 \leq i \leq d$. The number of i-faces of a d-cube is given by $2^{d-i}\binom{d}{i}$. Unless otherwise indicated, we refer to axis-aligned hypercubes, where all such directions are parallel with the Euclidean coordinate axes.

A *hypercubic mesh* is a polytopic mesh containing only hypercubes. Note that a hypercubic mesh can only be conforming if all hypercubes are uniform in size. Two d-cubes h_1 and h_2 in a hypercubic mesh are k-neighbors if their intersection $(h_1 \cap h_2)$ defines a k-cube.

Simplicial meshes

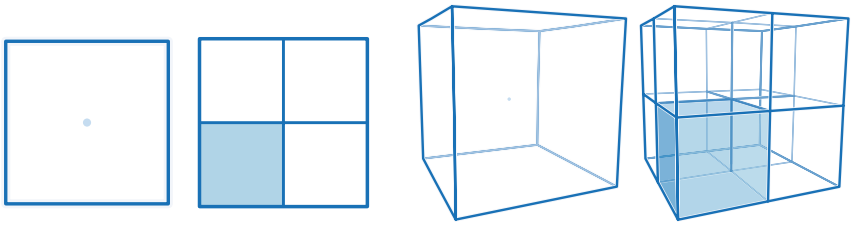
Another family of cells that has members in arbitrary dimension is defined by the *simplices*, which generalize the line segment (1-simplex), the triangle (2-simplex) and the tetrahedron (3-simplex). A d-dimensional *simplex*, or d-simplex, is the convex hull of $(d+1)$ affinely independent points in the n-dimensional Euclidean space. An i-face of a d-simplex σ is the i-simplex defined by any $(i+1)$ vertices of σ. The number of i-faces of a d-simplex is thus $\binom{d+1}{i+1}$.

A *simplicial mesh* Σ is a polytopic mesh containing only simplices, that is, all faces of a simplex $\sigma \in \Sigma$ belong to Σ, and the interiors of simplices from Σ are disjoint. Similarly, a *simplicial complex* is a simplicial mesh that is conforming. A simplicial complex of order d is referred to as a simplicial d-complex. As with cubes, all faces of a simplex are simplices. However, in contrast to hypercubic meshes, simplices in a conforming simplicial mesh do not need to have uniform size.

2.2 Nested mesh refinement

A mesh refinement scheme consists of a set of rules for replacing a set of cells Γ_1 with a larger set of cells Γ_2, i.e. $|\Gamma_1| < |\Gamma_2|$. In a *nested refinement scheme*, Γ_1 and Γ_2 cover the same domain.

In this paper, we are concerned with nested mesh refinement schemes in which the vertices are *regularly distributed*, that is, the vertices of Γ_2 are located at the midpoints of the faces of the cells of Γ_1. The two predominant types of such refinement schemes are *regular refinement* and *bisection refinement* [25, 4].

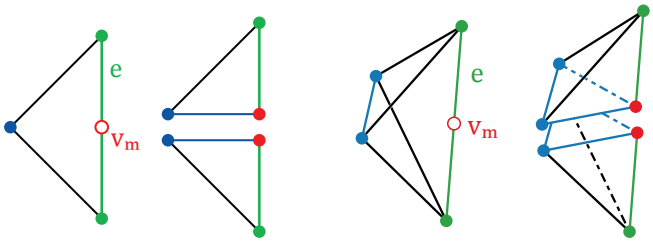


(a) Regular refinement of 2-cube (b) Regular refinement of 3-cube

Fig. 1: Regular refinement of a d-cube generates 2^d cubes.

In regular refinement schemes, a single cell γ is replaced by 2^d cells. Figure 1 illustrates the regular refinement of a square (i.e., a 2-cube) and of a cube (i.e., a 3-cube). When applied to a hypercubic domain, this rule generates quadtrees, octrees and their higher-dimensional counterparts. Similar schemes have been developed for the regular refinement of simplices, leading to data structures referred to as *simplicial quadtrees* [17].

Alternatively, in bisection refinement schemes, a single cell γ is replaced by the two cells obtained by splitting γ along a hyperplane. When applied to meshes composed of hyper-rectangles, and the splitting hyperplane is aligned with one of the bounding facets, this leads to kD-trees. Alternatively, when applied to simplices, this operation is referred to as *simplex bisection*. In this case, the splitting hyperplane cuts through the midpoint v_m of a single edge e of the simplex σ and all vertices of σ not incident to e (see Figure 2).



(a) Triangle bisection (b) Tetrahedron bisection

Fig. 2: A simplex is bisected along the hyperplane defined by the midpoint v_m (red) of an edge e (green) and all vertices (blue) not adjacent to that edge.

3 Related Work

In this Section, we are primarily concerned with work related to *balanced* nested hypercubic meshes and their conforming decompositions over a regularly sampled domain. A comprehensive survey of irregular triangulations can be found in [2].

Adaptive domain decompositions often impose rules on the allowed degree of decomposition among neighboring elements and thus are obtained by refining neighboring elements until the level of subdivision meets certain balancing requirements. Von Herzen and Barr [26] propose the *restricted quadtrees* for modeling parametric surfaces and define triangulation rules for rendering them. A restricted quadtree is a quadtree in which two edge-adjacent squares differ by at most one level in the decomposition. Sivan and Samet [23] first use such decompositions as a spatial data structure, specifically, for terrain modeling, and study different rules for subdividing the squares into triangles.

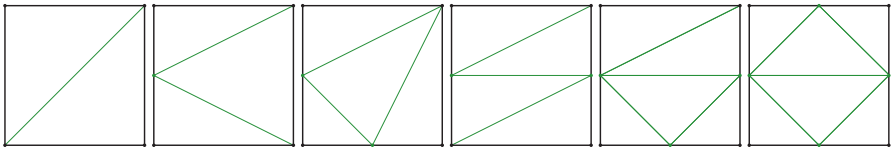
Bern and Eppstein [3] define a *balanced quadtree* as a mesh in which orthogonally adjacent nodes cannot be more than one level of refinement apart, and prove that a balanced quadtree in dimension d is at most a constant factor larger than its unbalanced counterpart. They also prove that simplices generated by using a Delaunay triangulation of a balanced hypercubic mesh have bounded angles. Moore [16] identifies different types of neighbors on which to balance a nested hypercubic hierarchy and uses this to find optimally tight bounds on the cost of balancing quadtrees, with an upper bound of $O(3^d)$. Specifically, a nested hypercubic mesh $\mathcal{C}$ is said to be *k-balanced* if all k-neighbors differ in refinement level by at most one [16]. Our triangulation algorithm (Section 5) depends on edge-balanced nested hypercubic meshes, where hypercubes adjacent along an edge can differ by at most one refinement.

In applications where conforming meshes are important, restricted quadtree meshes can be triangulated into simplicial complexes [26, 22, 33]. Such meshes can be triangulated based on *red-green* rules [33] or on simplex bisection rules [26, 22]. In the former case, which we refer to as *Delaunay-based triangulations*, cases are explicitly defined, for example, based on a Delaunay triangulation of the cube vertices. Figure 3a illustrates a complete set of triangulation cases (up to symmetry) for a square domain [18]. An alternate set of cases can be found in [33]. An advantage of Delaunay-based triangulation (in 2D) is that it does not require additional vertices, referred to as *Steiner vertices*, to triangulate the mesh. However, in higher dimensions these would likely be required. In contrast, *bisection-based triangulations*, such as the one proposed in Section 5, do not require explicit triangulation cases, but can require additional vertices along the faces of the hypercube adjacent to a higher resolution neighbor. A complete set of bisection-based triangulation cases for a square domain (due to [22]) is illustrated in Figure 3b. An alternative bisection-based approach (in 2D) is to remove unmatched vertices from higher resolution neighbors [19].

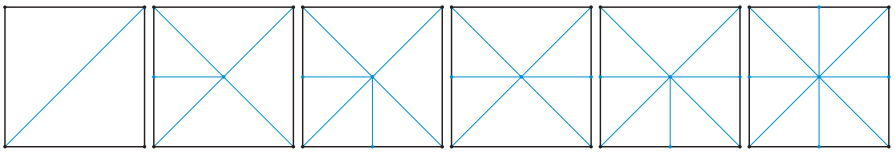
Plantinga and Vegter [18] propose triangulation cases for edge-balanced 3D octrees through the use of a Delaunay-based triangulation of the cube faces. Each cube is then tetrahedralized by connecting the face triangulations to the cube centers. We compare tetrahedral meshes generated by our algorithm to those generated by [18] in Section 6. A bisection-based triangulation has been suggested for balanced octrees [11], but details of this triangulation are not provided.

In higher dimensions, Weigle and Banks [27] propose a simplicial decomposition of hypercubes that recursively adds a vertex to the midpoint of each cell. This algorithm generates (non-adaptive) regular simplex bisection meshes, but has not been proposed in an adaptive setting.

Alternatively, conforming representations have been proposed for adaptive quadrilateral or hexahedral meshes based on explicit decomposition rules for neighboring cells of different sizes [21]. These rules are based on bisection (2:1) or trisection



(a) Delaunay-based triangulations of a square



(b) Bisection-based triangulations of a square

Fig. 3: Unique triangulation cases (up to symmetry) for squares (2-cubes) based on Delaunay triangulation (a) and bisection triangulation (b).

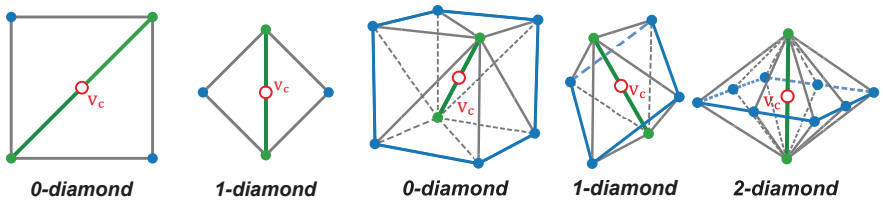
(3:1) of the hypercube edges. Recent work has focused on simpler decompositions and implementations for quadrilateral [10] and hexahedral meshes [12]. To the best of our knowledge, such decomposition rules have not been generalized to higher dimensional domains.

Compared to simplicial decompositions, such as the approach presented in this paper, conforming adaptive quadrilateral meshes and hexahedral meshes can be generated using fewer cells. However, in cases where values must be interpolated within each cell, the latter require multi-linear interpolation (based on 2^d values) while the former admit linear interpolation (based on only $(d+1)$ values).

4 Regular Simplex Bisection Meshes

A popular class of nested simplicial meshes over a regularly sampled hypercubic domain Ω is generated by the *Regular Simplex Bisection (RSB)* scheme. The initial mesh is a d-dimensional cube that has been refined along one of its diagonals into $d!$ simplices of order d. This refinement is the canonical simplicial decomposition of a hypercube, and is often referred to as *Kuhn's subdivision* [4] (see the left images in Figures 4a and 4b). One of the nice properties of a Kuhn-subdivided cube is that its lower dimensional faces are compatibly subdivided Kuhn cubes [7], and thus space can be tiled by translated (or reflected) Kuhn cubes.

Each simplex in this scheme is generated by bisecting an existing simplex along a predetermined edge, which is implicitly determined through a consistent ordering of the vertices [13, 25, 1]. In dimension d, this generates at most d similarity classes of simplices [13] that we refer to as *RSB simplices*. A recent survey on the RSB scheme and its applications can be found in [31].



(a) 2D diamonds (b) 3D diamonds

Fig. 4: The two classes of 2D diamonds (a) and the three classes of 3D diamonds (b).

4.1 Simplex hierarchies

The containment relation among simplices generated by the RSB scheme defines a nested *hierarchy of simplices*, which can be represented as a forest of binary trees in which each *parent* simplex has two *child* simplices. We refer to any mesh generated by the RSB scheme over an initially Kuhn-subdivided mesh as an *RSB mesh*.

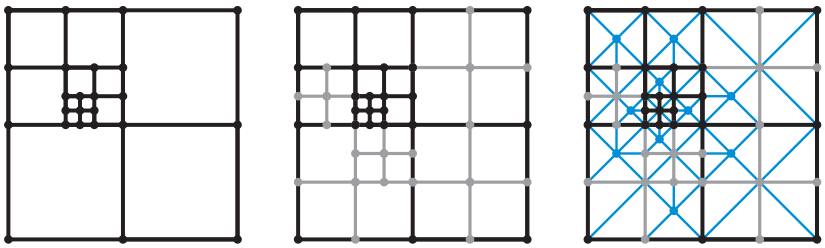
4.2 Diamond hierarchies

RSB meshes can be non-conforming. Thus, a conforming variant of the RSB scheme has been defined, in which the set of all RSB simplices sharing the same bisection edge, typically referred to as a *diamond*, must be bisected concurrently. Since all simplices in a diamond share the same bisection edge, the d classes of RSB simplices correspond to d classes of diamonds.

The bisection edge of a diamond of class i, where $0 \leq i < d$, which we refer to as the *spine* of the diamond, is aligned with the diagonal of an axis-aligned $(d-i)$-cube. The domain of a diamond can be decomposed as a cross product of two hypercubes that intersect at the midpoint of the diamond's spine [29]. In particular, we note that 0-diamonds correspond to Kuhn-subdivided d-cubes. Also, the spine of a $(d-1)$-diamond is aligned with an edge of a hypercube (see the right images in Figures 4a and 4b). Furthermore, a single vertex, which we refer to as the diamond's *central vertex*, is introduced during diamond refinement, at the midpoint of its spine.

The containment relation of all simplices within a diamond defines a *direct dependency relation* among the diamonds. Thus, the *parents* of a diamond δ are those diamonds whose refinement generates a simplex belonging to δ, and without which δ cannot be refined. The direct dependency relation can be modeled as a *Directed Acyclic Graph (DAG)* whose nodes are diamonds and whose arcs connect a diamond with its children. The diamonds, together with the dependency relation, define a multiresolution model that is referred to as a *hierarchy of diamonds*.

A hierarchy of diamonds is used to efficiently extract variable-resolution conforming RSB meshes, which we refer to as *diamond meshes*, satisfying the direct dependency relation of the hierarchy. Since each simplex σ in a diamond mesh Σ_d can be uniquely associated with a single diamond, Σ_d can be compactly encoded in terms of its diamonds. An efficient encoding for diamonds has been proposed in [29]



(a) Unbalanced (b) Edge-balanced (c) Triangulated

Fig. 5: Overview of triangulation algorithm (in 2D). Given an arbitrary nested hypercubic mesh (a), we first edge-balance the mesh (b) and then apply a local bisection-based triangulation to each hypercube (c).

based on the observation that each of the $O(d)$ parents of a diamond δ contributes $O(d!)$ simplices to δ upon its refinement. Thus, each diamond in Σ_d can be encoded using only $O(d)$ space.

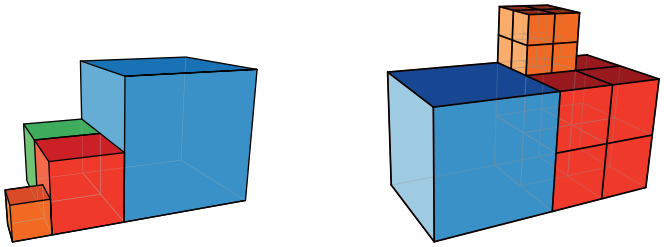
5 Triangulating nested hypercubic meshes

In this Section, we introduce a simple dimension–independent algorithm to triangulate a nested hypercubic mesh $\mathcal{C}$. Our triangulation algorithm is based on the observation that 0-diamonds cover a hypercubic domain, and thus, nested hypercubic meshes can be considered as a special case of (non-conforming) diamond meshes consisting of only 0-diamonds. Our triangulation is based on a local bisection refinement within each hypercube of $\mathcal{C}$. This converts $\mathcal{C}$ into a conforming *RSB* mesh Σ covering the domain of $\mathcal{C}$, i.e. Σ is a simplicial complex defined by a collection of *Regular Simplex Bisection (RSB)* simplices. The properties of RSB simplices [13, 31] ensure the quality of the triangulation. Specifically, the simplices in the mesh belong to at most d similarity classes of well-shaped simplices, and the valence of a vertex is at most $2^d d!$.

Our algorithm consists of three stages. First, we *edge-balance* the mesh (Section 5.1). This ensures that each face of a hypercube within $\mathcal{C}$ is refined by at most one internal vertex (see Figure 6). Next, we iterate the vertices of the edge-balanced mesh and cache them (Section 5.2). This replaces potentially expensive neighbor-finding operations with a single vertex lookup on each edge of the hypercube. Finally, we triangulate each hypercube locally using a diamond-based bisection refinement (Section 5.3). We conclude with a proof that the generated mesh Σ is a simplicial complex and find bounds on the complexity of Σ with respect to $\mathcal{C}$. Our algorithm is summarized in Figure 5.

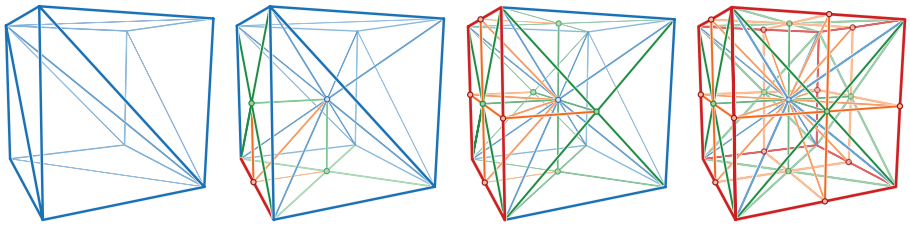
5.1 Mesh balancing

Let $\mathcal{C}$ be a (variable-resolution) nested hypercubic mesh obtained through regular refinement of an initial hypercubic domain Ω.



(a) Edge-balanced (b) Not edge-balanced

Fig. 6: The refinement level of edge-neighbors in an edge-balanced hypercubic mesh (a) can differ by at most one. Otherwise, faces of hypercubes within the mesh can contain more than one internal vertex (b).



(a) Unrefined (b) Refined edge (c) Refined face (d) All edges refined

Fig. 7: Bisection-based triangulation of a hypercube (shown in 3D) depends entirely on its refined edges.

For our triangulation algorithm, we require $\mathcal{C}$ to be an edge-balanced nested hypercubic mesh. This ensures that the faces of each hypercube need to be refined at most once, as well as the quality of the generated elements. Otherwise, the edges of its cubes might require more than one refinement, as can be seen in Figure 6b, where the edge of the blue cube (at level $\ell - 1$) adjacent to the orange cube (at level $\ell + 1$) has more than one internal vertex.

If the input mesh $\mathcal{C}_{in}$ is not edge-balanced, we can convert it to an edge-balanced mesh $\mathcal{C}$ using, e.g., Moore's greedy algorithm [16]. Note that this increases the number of hypercubes in the mesh by at most a constant factor (assuming a fixed dimension d) [28, 16].

5.2 Vertex caching

Our local triangulation algorithm only refines hypercube edges that have at least one smaller edge-neighbor. Since hypercubes can have many edge-neighbors, neighbor-finding operations can be cost-prohibitive at runtime. However, since $\mathcal{C}$ is edge-balanced, any refined neighbors of a hypercube h along an edge e will contain a vertex located at the midpoint v_m of e (see Figures 6a and 5b). Once we determine if v_m exists in $\mathcal{C}$, we no longer require these neighbor-finding operations.

Algorithm 1 CacheVertices($\mathcal{C}$)

Require: $\mathcal{C}$ is an *edge-balanced* nested hypercubic mesh.
Ensure: *Vertices*($\mathcal{C}$) is a spatial index on the vertices of cubes in $\mathcal{C}$.
1: *Vertices*($\mathcal{C}$) $\leftarrow \emptyset$.
2: **for all** hypercubes $h \in \mathcal{C}$ **do**
3: **for all** vertices $v \in h$ **do**
4: Insert v into *Vertices*($\mathcal{C}$).

Algorithm 2 TriangulateHypercubicMesh($\mathcal{C}$)

Require: $\mathcal{C}$ is an *edge-balanced* nested hypercubic mesh.
Require: *Vertices*($\mathcal{C}$) contains all vertices of hypercubes in $\mathcal{C}$.
Require: Σ_h is an RSB mesh covering hypercube $h \in \mathcal{C}$.
Ensure: $\Sigma = \bigcup_{h \in \mathcal{C}} \{\Sigma_h\}$ is a conforming RSB mesh covering $\mathcal{C}$.
1: **for all** hypercubes $h \in \mathcal{C}$ **do**
2: $\Sigma_h \leftarrow \emptyset$.
3: Let δ_h be the 0-diamond corresponding to h.
4: Insert δ_h into Σ_h.
5: **for all** edges $e \in h$ **do**
6: Let vertex v be the midpoint of e.
7: **if** $v \in$ *Vertices*($\mathcal{C}$) **then**
8: Let δ_e be the $(d-1)$-diamond associated with edge e.
9: Insert δ_e into Σ_h.
10: LocalRefineDiamond(δ_e, Σ_h, h).

Algorithm 3 LocalRefineDiamond(δ, Σ_h, h)

Require: The domain of diamond δ intersects h.
Require: Σ_h is a conforming RSB mesh restricted to the domain of hypercube h.
Ensure: δ is refined in Σ_h.
1: **for all** diamonds $\delta_p \in$ Parents(δ) **do**
2: Let v_p be the central vertex of δ_p.
3: **if** δ_p is not refined **and** $v_p \cap h \neq \emptyset$ **then**
4: LocalRefineDiamond(δ_p, Σ_h, h).
5: *// Refine δ by bisecting all of its simplices within Σ_h*
6: RefineDiamond(δ, Σ_h).

We therefore cache the vertices of $\mathcal{C}$ in a hash-table, (see Algorithm 1). Since each d-cube contains 2^d vertices, the cost of this step on a mesh with $|h|$ hypercubes is $O(2^d \cdot |h|)$, and the average cost of each vertex lookup is $O(1)$.

5.3 Hypercube triangulation

Our triangulation algorithm (see Algorithm 2) generates a globally conforming RSB mesh Σ through a local triangulation Σ_h of each hypercube $h \in \mathcal{C}$. This triangulation is entirely determined from a hypercube's refined edges.

We first convert each hypercube $h \in \mathcal{C}$ to an RSB mesh Σ_h defined by the 0-diamond associated with h (lines 2–4 of Algorithm 2). Since this is a Kuhn-subdivision of h (see Section 4), it contributes $d!$ simplices to Σ_h. See Figure 7a for an example in 3D.

For each edge $e \in h$ that is refined in a neighboring hypercube, we add the $(d-1)$-diamond δ_e associated with edge e to Σ_h, and *locally* refine δ_e within Σ_h. As mentioned in Section 5.2, we can determine the refined edges of a hypercube by checking if the midpoint of each edge is a vertex in $\mathcal{C}$ (lines 6–7).

The function LocalRefineDiamond(δ_e, h) (Algorithm 3) ensures that all diamond ancestors of δ_e whose central vertex intersects h (up to δ_h) are added to Σ_h. This satisfies the transitive closure of the diamond dependency relation, restricted to the domain of h, of each refined edge of h (see [29] for more details).

Figure 7 shows some possible triangulations Σ_h of a 3-cube h. In Figure 7a, none of the edges of h are refined, so Σ_h is defined by the 0-diamond associated with h and contains $d!$ simplices. This implies that all edge-neighbors of h in $\mathcal{C}$ are at the same level of refinement or one level higher in the hierarchy.

In Figure 7b one of the edges (red) of h is refined. The two facets of h adjacent to this edge are refined in Σ_h, as is the center of h. This triangulation occurs when a single edge-neighbor of h that is not a facet-neighbor, is refined.

Figure 7c illustrates the triangulation Σ_h when all four edges along a facet of h are refined. This corresponds to the case where a facet-neighbor of h is refined.

Figure 7d illustrates the triangulation Σ_h of h when all edge-neighbors of h are refined. Σ_h is a *fully-subdivided hypercube* [29], and is defined by $2^d \cdot d!$ simplices. Observe that all faces of h in Σ_h are refined.

The following Theorem proves that Algorithm 2 always produces a simplicial complex. Furthermore, the complexity of the generated mesh Σ with respect to the input hypercubic mesh $\mathcal{C}$ is bounded by a constant that depends only on the dimension d of the domain.

Theorem 5.1 *Given an edge-balanced hypercubic mesh $\mathcal{C}$ defined by $|h|$ hypercubes, Algorithm 2 generates a conforming RSB mesh $\Sigma = \bigcup_{h \in \mathcal{C}} \{\Sigma_h\}$ and is defined by $|\sigma|$ RSB simplices, where $|h| \cdot d! \leq |\sigma| < |h| \cdot 2^d \cdot d!$*

Proof. To show that Σ is conforming, we need to prove that (a) the triangulation Σ_h of each hypercube h is locally conforming, and (b) the boundaries $\Sigma_h \cap \Sigma_{h'}$ between neighboring hypercubes h and h' are also conforming.

The first constraint is satisfied since diamond refinement always generates a conforming RSB mesh. The function LocalRefineDiamond(δ_e, h) can be viewed as the triangulation of the root diamond in a hierarchy of diamonds after some of its edges have been refined.

The second constraint relies on the edge-balancing constraint of the input mesh $\mathcal{C}$, as well as the properties of Kuhn-subdivided and fully-subdivided hypercubes

(see [29, 31]). Note that, vertex-adjacent hypercubes that are not edge-adjacent are always conforming since their intersection is a vertex.

We first consider the case where the two neighboring hypercubes h and h' are at the same level of refinement. Since opposite pairs of lower dimensional faces of a Kuhn-subdivided hypercube are conforming, unrefined faces of neighboring hypercubes at the same resolution are conforming. Next, since our refinement rule in Algorithm 2 depends on the closed refinement of the edges, the lower dimensional faces in $h \cap h'$ are guaranteed to bisect in Σ_h and $\Sigma_{h'}$, i.e. the parents of a diamond δ_e associated with edge e, restricted to $h \cap h'$ will be identical for Σ_h and $\Sigma_{h'}$.

Finally, if h and h' are not the same size, assume, without loss of generality, that the level of h is ℓ and that of h' is $(\ell + 1)$. Due to the edge-balancing constraint on $\mathcal{C}$, it is not possible for faces of h' that belong to $h \cap h'$ to be refined. Thus, the only cases we need to consider are the refinement of faces of h in $h \cap h'$. Since the edges in $h \cap h'$ are refined by Algorithm 2, all higher dimensional faces are refined as well.

We conclude the proof by discussing the complexity of Σ. Let $|h|$ be the number of hypercubes in $\mathcal{C}$ and $|\sigma|$ be the number of simplices in Σ. Since Σ_h minimally contains the $d!$ simplices obtained through a Kuhn-subdivision of h (i.e. the simplices in its corresponding 0-diamond), the lower bound on $|\sigma|$ is $|h| \cdot d!$ simplices. This lower limit is obtained when $\mathcal{C}$ is a uniform resolution hypercubic mesh.

Similarly, since each edge (i.e. the $(d-1)$-faces) of a hypercube in $\mathcal{C}$ can be refined at most once, all j-faces, $j < (d-1)$, can be refined at most once. Thus, each local triangulation Σ_h, in the worst case, is a fully-subdivided hypercube. Σ_h therefore contributes at most $2^d \cdot d!$ simplices. This gives an upper bound on $|\sigma|$ of $|h| \cdot 2^d \cdot d!$. This upper bound is not tight since it is not possible for all edges of all hypercubes within a hypercubic mesh to be refined at the same time. □

6 Results

As a proof of concept, we demonstrate the bisection-based algorithm of Section 5 in an adaptive 3D isosurfacing application. We compare triangulations extracted from edge-balanced cubical meshes using bisection-based and Delaunay-based triangulations as well as triangulations extracted from a corresponding hierarchy of diamonds (see Table 1). In each case, $\mathcal{C}$ is a nested hypercubic mesh (in 3D), Σ_h is its associated bisection-based triangulation (extracted using Algorithm 2), Σ_p is its associated Delaunay-based triangulation (using the Algorithm of Plantinga and Vegter [18]) and Σ_d is a diamond-based RSB mesh extracted from a corresponding hierarchy of diamonds using the same extraction criteria. In all cases, the error associated with a cube or a diamond is the maximum interpolation error (computed using barycentric interpolation on its simplicial decomposition) among the points within its domain.

Recall that in the Delaunay-based triangulation of [18], Steiner vertices are inserted at the midpoint of every cube, but no additional vertices are added to their faces. In contrast, our bisection-based triangulation only adds Steiner points to cubes that have a smaller edge-neighbor, but can also add Steiner points to a hypercube's faces. As we can see from Table 1, the overhead of our algorithm in the 3D case compared to the Delaunay-based triangulation (in terms of the number of vertices and tetrahedra) is approximately 10%. However, since the bisection-based

Dataset	N	Mesh type	Vertices $\lvert v\rvert$	Primitives $\lvert h\rvert$ or $\lvert\delta\rvert$	Tetrahedra $\lvert\sigma\rvert$
Fuel	6	$\mathcal{C}$	20.0 K	15.3 K	-
		Σ_h	37.5 K	43.7 K	218 K
		Σ_p	35.3 K	-	206 K
		Σ_d	26.7 K	23.2 K	87.5 K
Hydrogen	7	$\mathcal{C}$	82.2 K	62.4 K	-
		Σ_h	156 K	187 K	928 K
		Σ_p	147 K	-	853 K
		Σ_d	108 K	93.0 K	357 K
Bunny	8	$\mathcal{C}$	627 K	467 K	-
		Σ_h	1.20 M	1.45 M	7.20 M
		Σ_p	1.09 M	-	6.43 M
		Σ_d	848 K	735 K	2.73 M
Engine	8	$\mathcal{C}$	1.11 M	850 K	-
		Σ_h	2.06 M	2.52 M	12.7 M
		Σ_p	1.97 M	-	11.6 M
		Σ_d	1.60 M	1.40 M	5.29 M
Tooth	8	$\mathcal{C}$	241 K	181 K	-
		Σ_h	461 K	556 K	2.76 M
		Σ_p	421 K	-	2.48 M
		Σ_d	325 K	281 K	1.05 M
Bonsai	8	$\mathcal{C}$	1.69 M	1.31 M	-
		Σ_h	2.97 M	3.54 M	17.9 M
		Σ_p	3.0 M	-	17.6 M
		Σ_d	2.20 M	1.94 M	7.57 M
Head	8	$\mathcal{C}$	1.04 M	784 K	-
		Σ_h	1.99 M	2.39 M	11.9 M
		Σ_p	1.82 M	-	10.7 M
		Σ_d	1.38 M	1.20 M	4.20 M
Armadillo	8	$\mathcal{C}$	262 K	195 K	-
		Σ_h	513 K	621 K	3.0 M
		Σ_p	458 K	-	2.70 M
		Σ_d	349 K	301 K	1.12 M

Table 1: Number of vertices $|v|$, primitives (cubes $|h|$ or diamonds $|\delta|$), and tetrahedra $|\sigma|$ in variable resolution meshes extracted from scalar fields of maximum resolution N (i.e., datasets defined by $(2^N+1)^3$ samples). For each dataset, $\mathcal{C}$ is an edge-balanced nested hypercubic mesh, Σ_h is a conforming diamond mesh generated from $\mathcal{C}$ using Algorithm 2, Σ_p is the tetrahedral mesh extracted from $\mathcal{C}$ using the Delaunay-based triangulation algorithm of Plantinga and Vegter [18] and Σ_d is a diamond mesh extracted from the corresponding *hierarchy of diamonds*.

algorithm generates conforming RSB meshes that satisfy the direct dependency relation of a hierarchy of diamonds, they can be efficiently encoded as *diamond meshes*, requiring $O(|\delta|)$ space, rather than as irregular simplicial meshes, requiring $O(|\sigma|)$ space [30]. Furthermore, while our bisection-based algorithm is defined in a dimension–independent manner, there would be difficulties in generalizing the Delaunay-based algorithm to higher dimensions. For example, a four-dimensional version of the Delaunay-based algorithm would require explicit triangulation cases for the different edge refinement configurations of the cubical faces of a 4-cube.

From Table 1, we see that nested cubical meshes $\mathcal{C}$ require approximately 66% the number of primitives (hypercubes) as their corresponding diamond meshes Σ_d to satisfy the same constraints. However, their triangulations Σ_h generate approximately 2.5 times as many tetrahedra as Σ_d. We can see this in Figure 8, which illustrates a cubical mesh extracted from the 201^3 Bunny dataset (Figure 8a) as well as its bisection-based triangulation (Figure 8b), and the diamond mesh extracted from a corresponding hierarchy of diamonds (Figure 8c), for the isosurface depicted in Figure 8d.

7 Concluding Remarks

In this paper, we introduced a dimension–independent algorithm for triangulating nested hypercubic meshes. This triangulation was motivated by the observation that 0-diamonds have a hypercubic domain, and thus nested hypercubic meshes can be viewed as (non-conforming) diamond-meshes composed of only 0-diamonds.

We compared our bisection–based triangulation to the Delaunay-based triangulation of Plantinga and Vegter in the 3D case. Although our 3D triangulation has a slight overhead with respect to the number of simplices and vertices, it admits an efficient representation as a diamond-based RSB mesh, which can significantly reduce the storage costs.

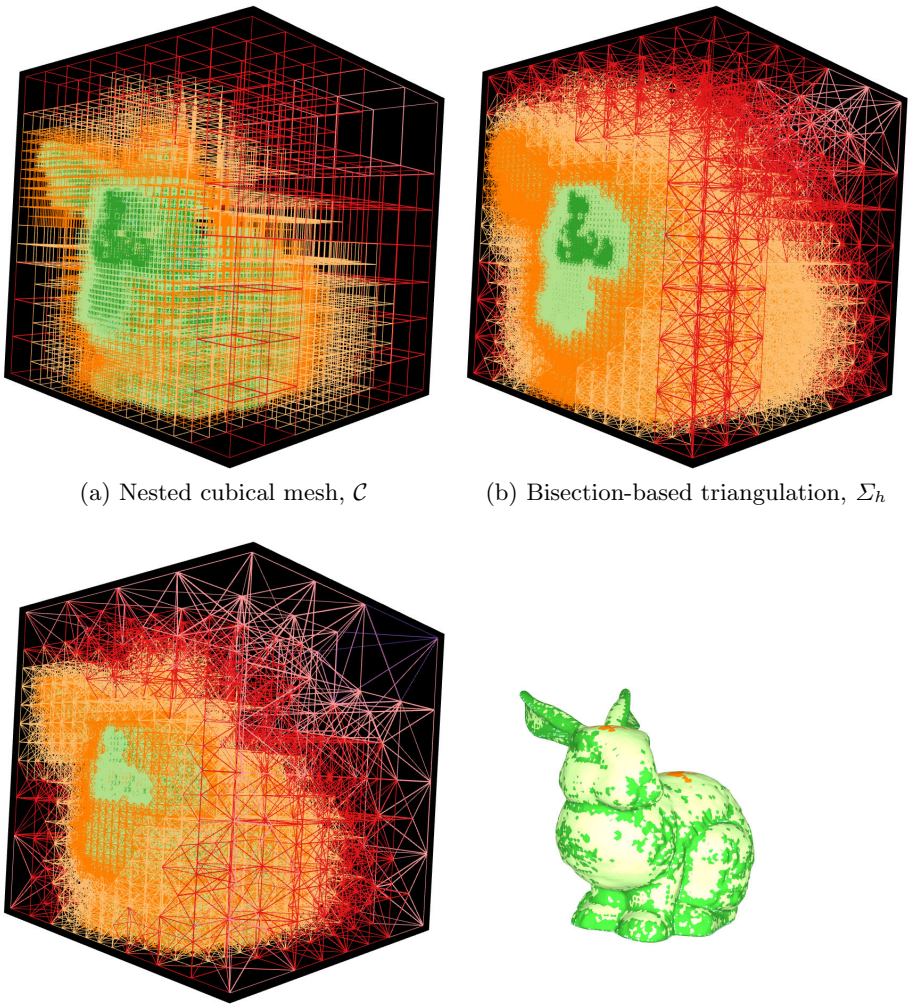
Furthermore, since our algorithm is defined in arbitrary dimensions, it can be easily incorporated into existing implementations of nested hypercubic meshes to generate high-quality triangulations for use in downstream applications. We are currently working on a 4D implementation of our algorithm to explore the benefits of this approach for the analysis and visualization of time-varying volumetric datasets.

As future work, we are investigating the relationship between meshes generated by hierarchies of simplices, which can be non-conforming, those generated by hierarchies of diamonds, which are always conforming, and those generated by a bisection-based triangulation of a nested hypercubic mesh. Let us call the family of meshes generated by simplex bisection as $\mathcal{S}$, the family of meshes generated by diamond refinement as $\mathcal{D}$ and the family of meshes generated by triangulating nested hypercubic meshes according to Algorithm 2 as $\mathcal{H}$. It has previously been observed [8] in the 2D case that $\mathcal{H} \subset \mathcal{D}$.

Since each diamond refinement is defined in terms of several simplex bisection operations, and since the hypercube triangulation scheme is defined in terms of diamond refinements, the following relationship holds in arbitrary dimension:

$$\mathcal{H} \subset \mathcal{D} \subset \mathcal{S}. \tag{1}$$

Note that each relationship defines a proper subset. I.e. there exist RSB meshes that are not conforming (i.e. $\Sigma \in \mathcal{S}$, but $\Sigma \notin \mathcal{D}$), as well as conforming RSB meshes



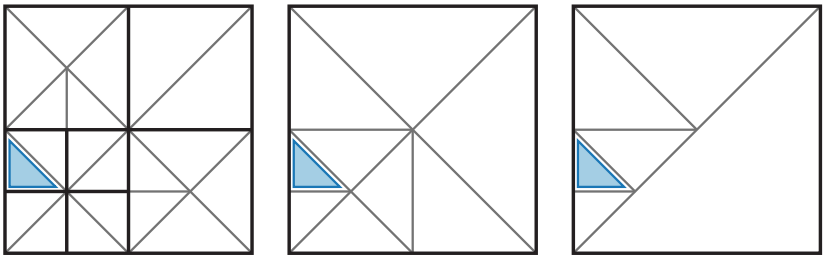
(a) Nested cubical mesh, $\mathcal{C}$

(b) Bisection-based triangulation, Σ_h

(c) Diamond mesh, Σ_d

(d) Extracted isosurface

Fig. 8: Decompositions of the 201^3 bunny dataset (a-c) associated with isovalue $\kappa = 0$ (d), colored by level of resolution. An edge-balanced cubical mesh (a) with error less than 0.3% contains $156K$ cubes. Its bisection-based triangulation Σ_h (b) contains $691K$ diamonds. A diamond-based mesh Σ_d (c) contains $166K$ diamonds.



(a) $\Sigma_h \in \mathcal{H}$ (b) $\Sigma_d \in \mathcal{D}$ (c) $\Sigma_s \in \mathcal{S}$

Fig. 9: Minimal RSB triangulations to generate a given RSB simplex (blue triangle) for nested hypercubic mesh (a), hierarchy of diamonds (b) and hierarchy of simplices (c). Note that $\Sigma_d \notin \mathcal{H}$ since it does not correspond to an edge-balanced hypercubic mesh, and $\Sigma_s \notin \mathcal{D}$ since it is not conforming.

that are not extractable from a triangulated edge-balanced hypercubic mesh. (i.e. $\Sigma' \in \mathcal{D}$, but $\Sigma' \notin \mathcal{H}$, see Figure 9 for examples in 2D).

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful suggestions. This work has been partially supported by the National Science Foundation under grant CCF-0541032. We would also like to thank Michael Kazhdan for providing us with the software used to generate the Armadillo distance field, and Ramani Duraiswami and Nail Gumerov for the Bunny dataset. All other datasets are courtesy of `volvis.org`.

References

1. Atalay, F., Mount, D.: Pointerless implementation of hierarchical simplicial meshes and efficient neighbor finding in arbitrary dimensions. International Journal of Computational Geometry and Applications **17**(6), 595–631 (2007)
2. Bern, M., Eppstein, D.: Mesh generation and optimal triangulation. In: D. Du, F. Hwang (eds.) Computing in Euclidean geometry, 1, pp. 23–90. World Scientific (1992)
3. Bern, M., Eppstein, D., Gilbert, J.: Provably good mesh generation. Journal of Computer and System Sciences **48**(3), 384 – 409 (1994)
4. Bey, J.: Simplicial grid refinement: on freudenthal's algorithm and the optimal number of congruence classes. Numerische Mathematik **85**(1), 1–29 (2000)
5. Bhaniramka, P., Wenger, R., Crawfis, R.: Isosurface construction in any dimension using convex hulls. IEEE Transactions on Visualization and Computer Graphics **10**(2), 130–141 (2004)

6. Brönnimann, H., Glisse, M.: Octrees with near optimal cost for ray-shooting. Computational Geometry **34**(3), 182 – 194 (2006)
7. Dahmen, W.A., Micchelli, C.A.: On the linear independence of multivariate b-splines, i. triangulations of simploids. SIAM Journal on Numerical Analysis **19**(5), 993–1012 (1982)
8. De Floriani, L., Magillo, P.: Multiresolution mesh representation: models and data structures. In: M. Floater, A. Iske, E. Quak (eds.) Principles of Multiresolution Geometric Modeling, Lecture Notes in Mathematics, pp. 364–418. Springer Verlag, Berlin (2002)
9. Evans, W., Kirkpatrick, D., Townsend, G.: Right-triangulated irregular networks. Algorithmica **30**(2), 264–286 (2001)
10. Garimella, R.: Conformal refinement of unstructured quadrilateral meshes. In: Proceedings of the 18th International Meshing Roundtable, pp. 31–44. Springer (2009)
11. Greaves, D.M., Borthwick, A.G.L.: Hierarchical tree-based finite element mesh generation. International Journal for Numerical Methods in Engineering **45**(4), 447–471 (1999)
12. Ito, Y., Shih, A., Soni, B.: Efficient hexahedral mesh generation for complex geometries using an improved set of refinement templates. In: Proceedings of the 18th International Meshing Roundtable, pp. 103–115 (2009)
13. Maubach, J.M.: Local bisection refinement for n-simplicial grids generated by reflection. SIAM Journal on Scientific Computing **16**(1), 210–227 (1995)
14. Min, C.: Local level set method in high dimension and codimension. Journal of Computational Physics **200**(1), 368–382 (2004)
15. Moore, D.: Simplicial mesh generation with applications. Ph.D. thesis, Cornell University, Ithaca, NY, USA (1992)
16. Moore, D.: The cost of balancing generalized quadtrees. In: Proc. ACM Solid Modeling, pp. 305–312. ACM (1995)
17. Moore, D., Warren, J.: Adaptive simplicial mesh quadtrees. Houston J. Math **21**(3), 525–540 (1995)
18. Plantinga, S., Vegter, G.: Isotopic meshing of implicit surfaces. The Visual Computer **23**(1), 45–58 (2007)
19. Roettger, S., Heidrich, W., Slusallek, P., Seidel, H.: Real-time generation of continuous levels of detail for height fields. In: Proceedings Central Europe Winter School of Computer Graphics (WSCG), pp. 315–322 (1998)
20. Samet, H.: Foundations of Multidimensional and Metric Data Structures. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann (2006)
21. Schneiders, R.: Refining quadrilateral and hexahedral element meshes. In: 5th International Conference on Grid Generation in Computational Field Simulations, pp. 679–688. Mississippi State University (1996)
22. Sivan, R.: Surface modeling using quadtrees. Ph.D. thesis, University of Maryland, College Park (1996)
23. Sivan, R., Samet, H.: Algorithms for constructing quadtree surface maps. In: Proc. 5th Int. Symposium on Spatial Data Handling, pp. 361–370 (1992)
24. Sundar, H., Sampath, R.S., Biros, G.: Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. SIAM Journal of Scientific Computing **30**(5), 2675–2708 (2008)
25. Traxler, C.T.: An algorithm for adaptive mesh refinement in n dimensions. Computing **59**(2), 115–137 (1997)

26. Von Herzen, B., Barr, A.H.: Accurate triangulations of deformed, intersecting surfaces. In: Proceedings ACM SIGGRAPH, pp. 103–110. ACM, New York, NY, USA (1987)
27. Weigle, C., Banks, D.: Complex-valued contour meshing. In: Proceedings IEEE Visualization, pp. 173–180. IEEE Computer Society (1996)
28. Weiser, A.: Local-mesh, local-order, adaptive finite element methods with a posteriori error estimators for elliptic partial differential equations. Ph.D. thesis, Yale University (1981)
29. Weiss, K., De Floriani, L.: Diamond hierarchies of arbitrary dimension. Computer Graphics Forum (Proceedings SGP 2009) **28**(5), 1289–1300 (2009)
30. Weiss, K., De Floriani, L.: Supercubes: A high-level primitive for diamond hierarchies. IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Visualization 2009) **15**(6), 1603–1610 (2009)
31. Weiss, K., De Floriani, L.: Simplex and diamond hierarchies: Models and applications. In: H. Hauser, E. Reinhard (eds.) EG 2010 - State of the Art Reports, pp. 113–136. Norrköping, Sweden (2010)
32. Westermann, R., Kobbelt, L., Ertl, T.: Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. The Visual Computer **15**(2), 100–111 (1999)
33. Zorin, D., Schroder, P.: A unified framework for primal/dual quadrilateral subdivision schemes. Computer Aided Geometric Design **18**(5), 429–454 (2001)