ABSTRACT

Title of dissertation: DIAMOND-BASED MODELS
FOR SCIENTIFIC VISUALIZATION

Kenneth Weiss, Doctor of Philosophy, 2011

Dissertation directed by: Professor Leila De Floriani
Department of Computer Science

Hierarchical spatial decompositions are a basic modeling tool in a variety of application domains including scientific visualization, finite element analysis and shape modeling and analysis. A popular class of such approaches is based on the *regular simplex bisection* operator, which bisects simplices (e.g. line segments, triangles, tetrahedra) along the midpoint of a predetermined edge. Regular simplex bisection produces adaptive simplicial meshes of high geometric quality, while simplifying the extraction of crack-free, or *conforming*, approximations to the original dataset. Efficient multiresolution representations for such models have been achieved in 2D and 3D by clustering sets of simplices sharing the same bisection edge into structures called *diamonds*.

In this thesis, we introduce several diamond-based approaches for scientific visualization. We first formalize the notion of diamonds in arbitrary dimensions in terms of two related simplicial decompositions of hypercubes. This enables us to enumerate the vertices, simplices, parents and children of a diamond. In particular, we identify the number of simplices involved in conforming updates to be factorial in the dimension and group these into a linear number of subclusters of simplices that are generated simultaneously. The latter form the basis for a compact pointerless representation for conforming meshes generated by regular simplex bisection and for efficiently navigating the topological connectivity of these meshes.

Secondly, we introduce the *supercube* as a high-level primitive on such nested meshes based on the atomic units within the underlying triangulation grid. We propose the use of supercubes to associate information with coherent subsets of the full hierarchy and demonstrate the effectiveness of such a representation for modeling multiresolution terrain and volumetric datasets.

Next, we introduce *Isodiamond Hierarchies*, a general framework for spatial access structures on a hierarchy of diamonds that exploits the implicit hierarchical and geometric relationships of the diamond model. We use an isodiamond hierarchy to encode irregular updates to a multiresolution isosurface or interval volume in terms of regular updates to diamonds.

Finally, we consider nested hypercubic meshes, such as quadtrees, octrees and their higher dimensional analogues, through the lens of diamond hierarchies. This allows us to determine the relationships involved in generating *balanced* hypercubic meshes and to propose a compact pointerless representation of such meshes. We also provide a local diamond-based triangulation algorithm to generate high-quality conforming simplicial meshes.

DIAMOND-BASED MODELS FOR SCIENTIFIC VISUALIZATION

by

Kenneth Weiss

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:

Professor Leila De Floriani, Chair/Advisor
Professor Larry S. Davis
Professor Samuel N. Goward
Professor David Mount
Professor Hanan Samet
Professor Amitabh Varshney

Acknowledgments

I owe my deepest gratitude to all the people who have made this dissertation possible.

First and foremost, I would like to thank my advisor, Professor Leila De Floriani, for your advice, encouragement, direction and support over the past five years. It has been a pleasure to work with you and to learn from you. I am especially grateful for the time and freedom to develop the ideas in this thesis. Although the scope of the project has changed considerably since our initial discussions, you were always happy to provide helpful thoughts, key suggestion in framing the problems and essential critiques to strengthen the results as this project has evolved.

I would also like to thank the the members of my thesis committee Professors Amitabh Varshney, Hanan Samet, David Mount, Larry Davis and Samuel Goward, for your time and insightful comments and suggestions. To Amitabh, thank you for your guidance during my early graduate career. Your advice on aiming for excellence and on the half-life of research ideas were invaluable in shaping my understanding of the research environment and in finding my role in it. Your suggestion about relating diamonds to quadtrees and octrees was the driving force behind Chapter 10. To Hanan, your framework for spatial decompositions and encyclopedic knowledge of the field were instrumental in my understanding and appreciation of hierarchical models. Thank you for all of your extremely helpful advice on matters related (and unrelated) to research. To Dave, it was a pleasure working with you as your teaching assistant for computer graphics. Your dedication to your work and students is an inspiration.

I would also like to thank my teachers and classmates in the University of Maryland and in Binghamton University. In particular, thank you Professor Lijun Yin for involving me in your research as an undergraduate and for instilling in me a passion for research in computer graphics and visualization. Your training and guidance helped me get through my first few years of graduate school.

I owe my deepest thanks to my family for their love and support, for encouraging me to pursue my academic interests and for placing such a high value on my education. To my father, Peter, for explaining the difference between bits and bytes on those long drives many years ago and for teaching me to always think strategically in chess and in life. To my mother, Bella, for always asking about the details of my work, even when the topics are abstract and the hour is late. To Linda, for always encouraging me to pursue my artistic interests. To Jamie and Ari, for always being there when I need you. To Lillian, David and Ariella, for understanding what it is like to be in graduate school.

Finally, to my wife Aliza, thank you for your endless love and encouragement. I could not have made it through this program without you and I dedicate this thesis to you.

Table of Contents

iii

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

One of the fundamental problems in computer graphics, scientific visualization, geographic data processing, and shape analysis and understanding is how to deal with the huge amount of data that describe the objects of interest. A diverse class of approaches utilizes a hierarchical organization of the field domain to describe subsections of these objects. Examples include the analysis and visualization of two-, three- and higher-dimensional scalar fields, where the domain of the field is adaptively decomposed into nested cells of a simple geometric shape.

The availability of very large meshes describing free-form shapes, terrains, and volume data sets has led to the investigation of mesh-based multiresolution methods to control and adjust the *level of detail (LOD)* in the representation of a given data set. Multiresolution models provide a great deal of flexibility since they compactly encode a large number of different mesh-based representations of a shape, or of a field, and enable the efficient extraction of a variety of different representations at uniform or variable resolutions.

Although an *irregular* sampling of the domain provides a great deal of flexibility in the sampling locations, and can thus represent the features of a problem domain using fewer elements than a regularly sampled domain, this representation has a *geometric overhead*, due to the explicit storage of the coordinates, and a *topological overhead*, due to the explicit representation of the connectivity among the vertices. Conversely, regularly sampled domains have no geometric or topological overhead since the coordinates can be directly inferred from the position of the data points in a grid. However, they suffer from a high *sampling overhead*, that is, the overhead related to the number of samples required to represent the features within the dataset, due to the rigid structure of such grids. When a multiresolution representation of the dataset is considered, the *hierarchical overhead*, related to how finer representations are obtained from coarser ones, must also be considered.

Decompositions based on nested hypercubes, such as quadtrees and octrees, are a typical compromise between the two representations since their vertices lie along a regular grid while aggregating the less relevant regions into larger blocks. A drawback of these techniques is that they introduce an exponential number of vertices and cells (of the order d of the dimension of the domain) during each subdivision. Furthermore, both quadtrees and octrees (as well as their d-dimensional generalizations, that we call 2^d-trees) are less suitable for generating crack-free, or *conforming*, decompositions since the bilinear or trilinear interpolant over each square or cubic cell generates discontinuities on the boundary of adjacent cells that have different sizes. Thus, additional rules must be applied to ensure compatibility between neighboring cells.

In contrast, models based on Maubach's simplicial bisection scheme [Mau95],

| **(a)** Irregular | **(b)** Quadtree | **(c)** RSB Mesh |
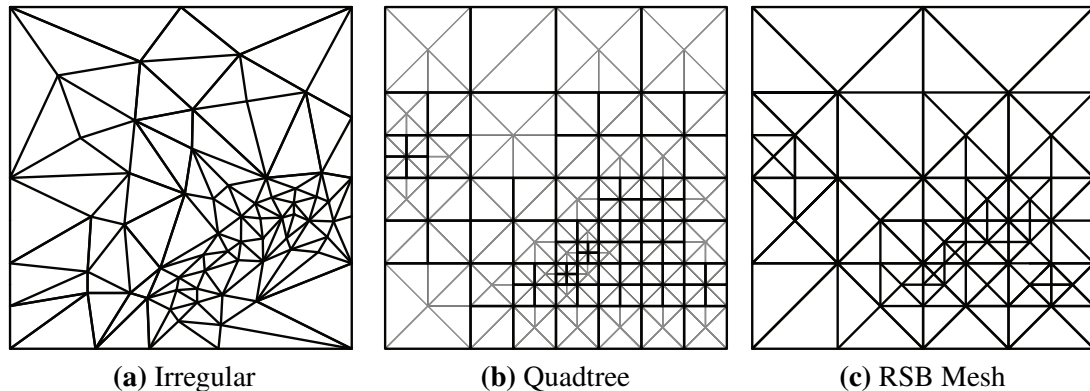
Figure 1.1: (a) Irregular mesh (b) Triangulated balanced quadtree (c) Regular Simplex Bisection (RSB) mesh.

which we refer to as the *Regular Simplex Bisection (RSB)* scheme, enable the generation of more adaptive meshes over the same domain by breaking up each 2^d-tree subdivision into d steps [Pas02] while enforcing conforming modifications to the model. Nested RSB meshes in fact have a higher degree of adaptability to the features of a domain than 2^d-trees, since RSB subdivisions only add a single vertex (rather than an exponential number of vertices), and double the number of cells involved in each subdivision. Furthermore, they have a higher representational power than 2^d-trees, i.e., the set of triangulations derivable from RSB subdivisions is a superset of those derivable from triangulated balanced quadtrees and octrees. On the other hand, compared with multiresolution models based on irregular sampling, such as the *MultiTessellation (MT)* [DPM97], nested RSB meshes can be encoded implicitly and thus have smaller storage costs [DFMP00].

Efficient representations have been proposed for multi-resolution models of scalar fields in 2D and 3D based on clustering maximal simplices (triangles or tetrahedra, in these cases) sharing their bisection edge into a new geometric primitive called a *diamond*. Diamond-based representations exploit the regularity of the vertex distribution, and of the subdivision rule which produces diamonds of certain fixed shapes, to yield an implicit encoding of the hierarchical and geometric relationships among the triangles and tetrahedra, respectively.

These subdivisions have many applications in the analysis and visualization of scientific and medical data. They have been applied to interactive terrain rendering [EKT01, LP02, DWS+97] and multiresolution isosurface extraction [ZCK97, GR99, GDL+02], as well as volume segmentation [KTY+04], surface reconstruction [MVT03] and finite element analysis [Mau95, RL92]. Four-dimensional applications include multiresolution representations for time-varying volumetric datasets [LDS04, LPD+04], acceleration structures for ray tracing [AM07] and the analysis of bivariate complex functions [WB96]. Higher dimensional applications include five-dimensional weather data [HAF+96], fixed point computations [Mau95] and the solution spaces of parametrized equations.

(a) Values associated with cells   **(b)** Values associated with vertices
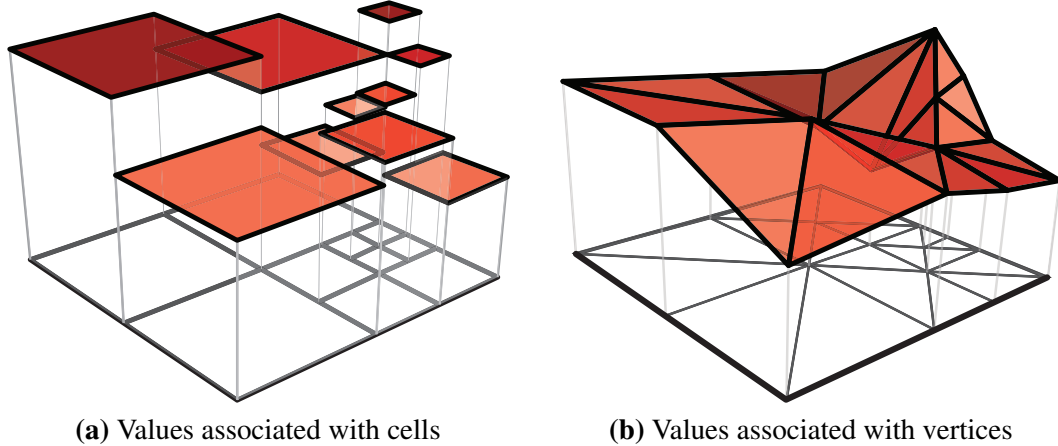
Figure 1.2: (a) Associating scalar values with the cells of a mesh can lead to cracks. (b) A mesh is free of cracks when values are associated with the vertices of a conforming mesh.

1.1 Contribution

In this thesis, we argue that diamonds are the natural representation for generating, modeling and encoding conforming meshes extracted through the Regular Simplex Bisection scheme and introduce several diamond-based approaches for scientific visualization.

Diamond hierarchies of arbitrary dimension. Our first contribution is a formalization of the notion of diamonds in arbitrary dimensions and their relationship to the simplices generated by the Regular Simplex Bisection (RSB) scheme. We frame our discussion of diamonds in terms of two related simplicial decompositions of hypercubes. Through a careful analysis of the properties of these structures, we prove that diamonds can be decomposed as a *cross complex* (a simplicial cross product) of these two structures and we derive the first closed-form equations for the number of vertices, simplices, parents and children of a diamond.

Specifically, we prove that the number of d-simplices in a d-dimensional diamond which must be bisected simultaneously for conforming refinements is factorial in the dimension d of the domain. However, due to the regularity of the diamond subdivision operation, these simplices can be grouped into a linear number of clusters, which we refer to as *parent-child duets*, that are generated simultaneously. Consequently, while simplex-based representations require $O(d!)$ time to update and space to encode an extracted simplicial complex, diamond-based representations can encode the $O(d!)$ simplices of a diamond using only $O(d)$ space, and can subdivide diamonds in $O(d)$ time. Our combinatorial decomposition leads to an implicit pointerless representation for d-dimensional diamonds, enabling efficient representations for hierarchies of diamonds and their extracted meshes.

Supercubes: A high-level primitive for RSB hierarchies. Next, we propose a high-level primitive for hierarchies generated by RSB, which we call *supercubes*. A supercube is a structured set of edges within an RSB hierarchy that captures the intrinsic symmetry within the model. Whereas simplex-based and diamond-based representations contain

several similarity classes of primitives, each with different geometric alignments, super-cubes are all identical (up to translation and scale). Diamonds and supercubes are related by a one-to-one correspondence from the bisection edge of a diamond to an edge of a supercube. Thus, each edge of a supercube corresponds to a distinct *type* of diamond within the hierarchy.

Many operations on RSB hierarchies apply to only a sparse subset of the elements within the hierarchy. For example, due to the uniformity of the regular sampling, large regions of a scalar field's domain are often oversampled. Additionally, in isosurfacing applications, we are only interested in the subsets of the domain that intersect a given iso-surface. However, implementations of these hierarchies have mostly focused on efficient representations for the entire hierarchy, where all simplices or diamonds are implicitly indexed. We propose the use of supercubes as containers for data associated with coherent subsets of the elements of an RSB hierarchy. Since partial representations incur overhead related to the explicit storage of spatial indexes, we consider the dataset's *density*, the number of encoded samples with respect to an encoding of the full hierarchy, as well as its *concentration*, the average number of elements associated with each supercube, to evaluate the effectiveness of a supercube-based representation for a given dataset. We demonstrate that the geometric overhead incurred by a multiresolution model of a complete or partial scalar field is largely reduced through the use of supercubes.

A particularly important application of supercube-based models is in the field of Geographic Information Systems (GIS), where regular grids are the most common for-mat for terrain datasets. Supercubes provide a solution for a long-standing problem of representing subsets of a regular grid. For example, elevation data for rough terrain and coastlines are required to be sampled at a high resolution, while flatter regions, especially those covering large bodies of water, do not require such high sampling resolution. This is especially relevant for global datasets since approximately 70% of the earth's surface is covered by water.

Similarly, for volumetric datasets, the objects of interest are often embedded within a regularly sampled cubic domain. We demonstrate that typical volumetric datasets (from the Volume Visualization repository [Vol]) are oversampled by a factor or three or more, with respect to a lossless supercube-based encoding of the multiresolution scalar field.

Topological navigation on diamond meshes.   In shape modeling and analysis applica-tions, we are often interested in computing local properties about elements within a mesh. Such queries are typically posed in terms of local neighborhoods surrounding a region of the mesh and require efficient support for navigating its topological connectivity. Ex-amples include *visibility queries* on terrain datasets, compression and repair of simplicial meshes, curvature estimation and ray casting algorithms.

However, due to the increasing sizes of datasets, it is important to reduce the storage requirements associated with the mesh's topological connectivity. Not only does this require additional storage space, but it must also be maintained when modifying the mesh, for example, during mesh simplification or refinement.

Diamond meshes are a compact representation for conforming RSB meshes which do not explicitly encode any information related to topological connectivity of its elements.

After analyzing the structure of these meshes based on the manner in which they are generated, we introduce algorithms to perform topological navigation directly on these meshes without requiring any additional information. We demonstrate that in 3D, our representation requires an order of magnitude less space than the state of the art representation for tetrahedral meshes [PBCF93], and that the benefits of this representation increase with the resolution of the extracted mesh.

Isodiamond hierarchies.   Another contribution of the current work is in the visualization of volumetric datasets, which are often analyzed through their surfaces of constant field value, known as *isosurfaces*, or through the subvolumes enclosed by two isosurfaces, known as *interval volumes*. These structures are based on a limited range of the field values and typically occupy a sparse, but spatially widespread, subset of the dataset. Multiresolution representations are useful to analyze and visualize isosurfaces and interval volumes, since at full resolution both can contain from millions to billions of elements. Furthermore, in scientific and medical applications, details at the highest available resolution are required on demand and, thus, simplified approximations of these datasets are not sufficient.

Once a relevant isosurface or interval volume is found, it can be useful to represent it in a more convenient format, e.g. for transmission and offline viewing of the mesh. Indexed tetrahedral meshes are a popular output format for such meshes, where only the vertices and tetrahedra are explicitly stored. However, while this is a useful format for rendering applications, it is not particularly useful for more involved mesh processing tasks that require a notion of spatial proximity between elements. Also, these indexed meshes can also exceed a machine's processing capabilities and often require simplification to be suitable for downstream tasks.

To this aim, we introduce the *Isodiamond Hierarchy* framework for encoding multiresolution isosurfaces or interval volumes extracted from a hierarchy of diamonds. The basic idea here is to exploit the regular nested decomposition of the domain to produce compact hierarchical representations of a single isosurface or a single interval volume. Since the desired isosurface or interval volume is determined in advance, these representations are able to decouple the implicit hierarchical and geometric relationships in the hierarchy of diamonds from the field values of the volume dataset. Thus, such models enable a compact encoding of modifications to the irregular isosurface or interval volume in terms of regular modifications to the corresponding nodes of the hierarchy of diamonds.

We introduce two multiresolution models based on this framework. The first model, which we call a *Relevant Isodiamond (RI) hierarchy*, encodes a collection of *active* modifications to a coarse representation, corresponding to the diamonds in the hierarchy of diamonds intersected by the isosurface or interval volume. An RI hierarchy also encodes all *relevant* modifications, which are ancestors of active modifications that are not intersected by the isosurface or the interval volume. Relevant modifications enable spatial access to the active modifications and guarantee that all the meshes extracted from the RI hierarchy are free of cracks.

The second multiresolution model that we propose, which we call a *Minimal Isodiamond (MI) hierarchy*, encodes the active modifications as well as a small subset of the

relevant modifications corresponding to the creation of new isosurface or interval volume components. As a consequence, this model requires a careful analysis to guarantee that the extracted meshes do not self-intersect. In addition to the reduced storage requirements, we show that the same uniform- and variable-resolution representation of the encoded isosurface or interval volume can be extracted from the MI hierarchy as from the RI model, but in less time, and using less memory. Both representations support efficient general selective refinement operations to extract conforming meshes at different resolutions from the model using application-dependent selection criteria.

Hierarchies of balanced hypercubes.   Although the focus of this thesis is on nested simplicial decompositions, there are many applications of nested hypercubic grids including those based on quadtrees, octrees and their higher dimensional extensions [Sam06]. Downstream applications typically 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 [Bey00]. A common *adaptivity* constraint is to ensure that neighboring elements differ in resolution by at most one refinement level, i.e. the ratio of edge lengths between *neighboring* elements can be at most 2:1. This constraint has been considered in many different application domains, including computational geometry [BEG94, BG06], scientific visualization [EKT01, WKE99] and computer graphics [VHB87] under various terms such as *restricted* [VHB87, SS92], *smooth* [Moo92] and *balanced* [Moo95, SSB08].

We introduce *hierarchies of balanced hypercubes* defining families of nested hypercubic meshes with balancing restrictions. We provide a formal treatment of the dependency relation among hypercubes in a nested hypercubic mesh, necessary to generate balanced hypercubic meshes. This framework is general enough to encompass traditional quadtrees and octrees, which we refer to as *unbalanced*.

Our analysis stems from a novel reinterpretation of nested hypercubic meshes through the lens of diamond hierarchies, whereby hypercubes are seen as a special class of diamonds. This yields a compact pointerless encoding for balanced hierarchies of hypercubes, which provides random access to the hierarchical ancestors of each hypercube as well as for meshes extracted from such hierarchies. The connection to diamonds also suggests a supercube-based representation for encoding the vertices and cells of a balanced hypercubic mesh, and for a local diamond-based triangulation algorithm to generate conforming RSB meshes from balanced hypercubic meshes.

1.2   Thesis organization

The remainder of this dissertation is organized as follows.

In Chapter 2, we review some background notions on cell and simplicial complexes, on modeling scalar fields and on mesh-based multiresolution models.

In Chapter 3, we review the state of the art on domain decompositions and on visualization of scalar fields, with an emphasis on approaches based on regular simplex

bisection.

In Chapter 4, we introduce our dimension-independent approach for diamonds. We frame our discussion in terms of two related simplicial decompositions of a hypercubic domain, and prove that a diamond can be defined as a *cross complex* of these two decompositions. This enables a comprehensive description of the properties of diamonds in arbitrary dimensions. We conclude with a discussion of diamond-based simplicial complexes extracted from a hierarchy of diamonds and some implications of our diamond-based approach.

In Chapter 5, we introduce the *supercube* as a high-level primitive for hierarchies of nested RSB meshes. After considering the underlying domain decomposition induced by RSB, we investigate the properties of supercubes, and the number of vertices, edges, simplices and diamonds uniquely indexed by each supercube.

We follow this in Chapter 6 with an encoding for diamonds, diamond hierarchies, supercubes and RSB meshes extracted from a hierarchy of diamonds.

We next introduce the Diamond-based Multiresolution Scalar Field (DMSF) model for a scalar field in Chapter 7. The *Full DMSF* model applies to datasets defined over a hypercubic domain of resolution $(2^N + 1)^d$, while the *Partial DMSF* model applies to a subset of samples from a full DMSF that is *closed* with respect to the diamond dependency relation. We discuss the advantages of a partial DMSF from a theoretical perspective, and introduce several practical applications.

In Chapter 8 we reexamine the structure of extracted diamond meshes to define optimal algorithms for topological navigation on 2D and 3D diamond meshes. We then compare our diamond-based and supercube-based representations for RSB meshes to a simplex-based representation and to a compact topological data structure for general simplicial meshes. Compared to the latter, our representations require an order of magnitude less space.

In Chapter 9, we introduce the *Isodiamond Hierarchy* framework for spatial access structures defined on a hierarchy of diamonds. The two multiresolution models for isosurfaces and interval volumes are introduced in Sections 9.3 and 9.4, respectively, and are compared in Section 9.5.

In Chapter 10, we apply our understandings of diamond hierarchies to balanced hierarchies of nested hypercubes. We first analyze the dependency relation induced by balanced hypercubic refinement to define a multiresolution model. Next, we adapt our diamond encoding of Chapter 6 to yield an efficient representation for nested hypercubic meshes. Finally, we introduce a diamond-based triangulation algorithm to convert nested hypercubic meshes into conforming RSB meshes.

We conclude in Chapter 11 with a summary of our results. In particular, we discuss how the supercube primitive relates the families of nested RSB meshes generated by simplex bisections, by diamond subdivisions and by triangulations of nested hypercubic meshes.

Chapter 2

Background notions

In this chapter, we introduce some background notions on cell complexes, on modeling scalar fields and on mesh-based multiresolution models that we use in the remainder of the thesis.

2.1 Cellular 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-dimensional cell, or *d-cell*, as a convex polytope in some d-dimensional subspace of $\mathbb{R}^n$. For convenience, we define the empty cell to have dimension -1. Let c be a d-cell. Then, an i-dimensional *face* c_i of c, denoted $c_i \subseteq c$, where $-1 \leq i \leq d$, is an i-cell on the boundary of c. We refer to a 0-cell as a *vertex*, a 1-cell as an *edge*, and for a given d-cell, we refer to its $(d-1)$-faces as *facets*. The *diameter* of a polytope p is defined as the maximum distance between any two points on the boundary of p.

A *cellular mesh* Π is a finite collection of cells such that (a) if c is a cell in Π, then all faces $c_i \subseteq c$ also belong to Π, and (b) the interiors of cells in Π are disjoint (see Figure 2.1) [Ago05]. The *dimension*, or *order*, of a cellular mesh is the maximum of the dimensions (orders) of the cells forming it. In a cellular mesh, cells that are not on the boundary of any other cells are called *top cells*. In a cellular mesh of order d with a manifold domain, as we will consider here, all top cells are d-cells. Intuitively, a manifold (with boundary) X is a subset of d-dimensional Euclidean space such that each point $\mathbf{x} \in X$ has a neighborhood that is homeomorphic to an open ball or to an open ball intersected by a plane when $\mathbf{x}$ is on the boundary of X. Such a mesh is also referred to as a *pure* cellular mesh.

If, additionally, the intersection of any two cells $c_1, c_2 \in \Pi$ is a lower dimensional cell on the boundary of c_1 and c_2, then Π is said to be *conforming*, or *compatible*. A conforming cellular mesh is also referred to as a *cell complex*. Conforming meshes are important in many applications since they ensure that there are no cracks or T-junctions between adjacent cells.

The *(combinatorial) boundary* of a cell c in a cell complex Π is the set of all cells of Π, excluding c, that are faces of c. The *co-boundary*, or *star* of a cell c, denoted $St(c)$ is the set of cells in Π containing c in its boundary. Note that, a cell $c \in \Pi$ is a top cell if $St(c)$ contains only c. The *link* of a cell c, denoted $Lk(c)$, is the set of all the faces of the cells in $St(c)$ which are not incident to c.

Two cells are *incident* to each other if one of them is a face of the other, while they are *k-adjacent* if they share a k-face: in particular, two vertices are called *adjacent* if they are both incident to a common edge, and two i-cells ($i > 0$) are called *adjacent* if they are

(a) Intersect   **(b)** Crack   **(c)** Conforming
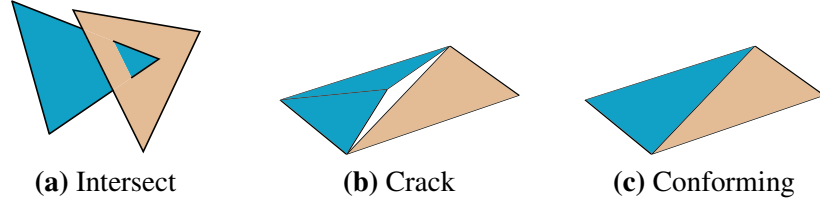
Figure 2.1: A decomposition is *conforming* (c) if its cells do not intersect (a) and it is free of cracks (b).
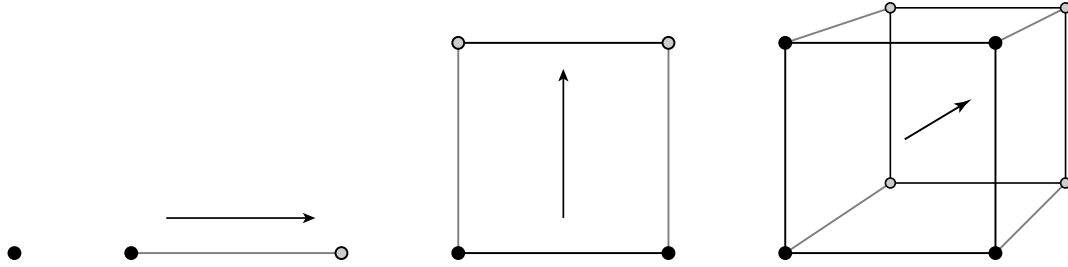
Figure 2.2: Hypercubes: recursive definition.

$(i-1)$-adjacent.

2.1.1 Hypercubic meshes

Hypercubes are a class of polytopes that can be defined in arbitrary dimension. They generalize line segments (1-cubes), squares (2-cubes) and cubes (3-cubes). A d-dimensional hypercube, or d-cube, can be defined recursively: A 0-cube is a single point, and a d-cube is created by extruding a $(d-1)$-cube one unit along a direction orthogonal to the previous $(d-1)$ directions (see Figure 2.2). Thus, d-cubes have twice as many vertices as $(d-1)$-cubes. Unless otherwise indicated, we refer to axis-aligned hypercubes, where all such directions are parallel with the Euclidean coordinate axes.

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 $\binom{d}{i}2^{d-i}$. In particular, a d-cube h has 2^d vertices; $(2^{d-1} \cdot d)$ edges; and $(2 \cdot d)$ facets. The total number of faces of a d-cube is thus:[*]

$$\sum_{i=0}^{d}\binom{d}{i}2^{d-i} = 3^d. \tag{2.1}$$

The diameter of a d-cube h is referred to as a *diagonal* and is defined by a pair of *opposite vertices*, that is, two vertices of h whose only common face is h (see Figure 2.3). Let $(\mathbf{v}_1,\mathbf{v}_2)$ be an unordered pair of opposite vertices of a hypercube h with side length s. Then an edge between $\mathbf{v}_1$ and $\mathbf{v}_2$ forms a diagonal of h and has length $s\sqrt{d}$. Since faces of a hypercube are also hypercubes, a diagonal of an i-face of h has length $s\sqrt{i}$.

---

[*]This is a special case of the binomial theorem, see Appendix C.

10
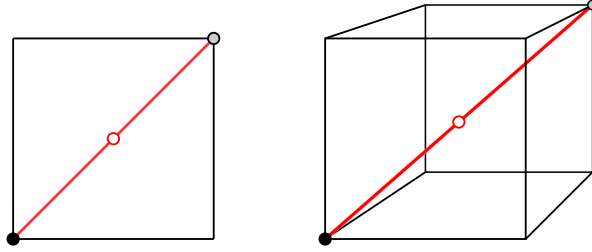
Figure 2.3: Diagonal of a hypercube

A *hypercubic mesh* is a cellular mesh containing only cubes. Note that a hypercubic mesh can only be conforming if all cubes are uniform in size. A hypercubic meshes is referred to as *balanced* if edge lengths of neighboring hypercubes are either equal or have a ratio of 2:1. It is *k*-balanced if all *k*-adjacent hypercubes have this property.

2.1.2 Simplicial meshes

The *simplices* are another class of cells that can be defined in arbitrary dimension. They generalize line segments (1-simplices), triangles (2-simplices) and tetrahedra (3-simplices). 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 cellular 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 complexes, simplices in a simplicial complex do not need to have uniform size.

2.2 Nested mesh refinement

A *nested refinement scheme* consists of rules for replacing a set of cells Γ_1 in a cellular mesh Π with a larger set of cells Γ_2 covering the same domain. When Γ_1 and Γ_2 share the same combinatorial boundary, the refinement does not introduce cracks into Π, i.e. the refinement is conforming.

Recall that two cells τ_1 and τ_2 are *similar* if there is an affine map $\mathcal{A}$ defined by translations, rotations, reflections and uniform scaling between them, i.e. $\tau_1 = \mathcal{A} \cdot \tau_2$. An equivalence class of similar cells is referred to as a *similarity class* of cells. The number of similarity classes generated by successive refinements is an important characteristic of a refinement scheme, since it enables the analysis of properties of all generated cells. In particular, it is important in many applications, such as finite element analysis, that the angles at the vertices are bounded. Such a scheme is referred to as *stable* [Bey00].

The two primary categories of nested refinement schemes for regularly sampled domains are those built on *regular refinement* and on *bisection refinement* [Tra97]. For an example of a nested mesh refinement scheme over irregularly sampled domains, see [DP95].

(a) Regular refinement of 2-cube   **(b)** Regular refinement of 3-cube
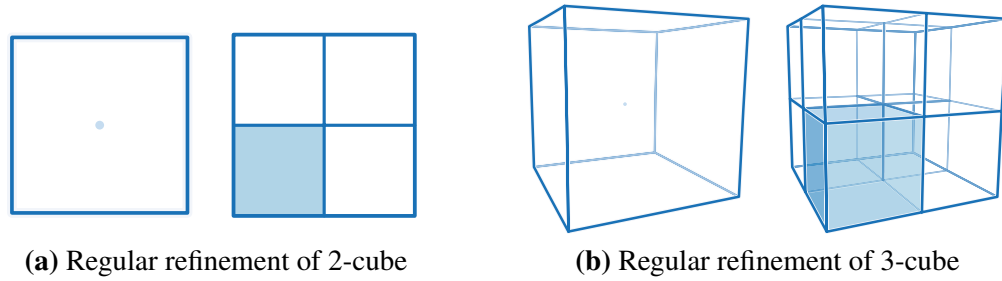
Figure 2.4: Regular refinement of a d-cube h generates 2^d d-cubes all incident on the midpoint of h. (a) A square is decomposed into four squares. (b) A cube is decomposed into eight cubes.

(a) Triangle Refinement   **(b)** Tetrahedral Refinement
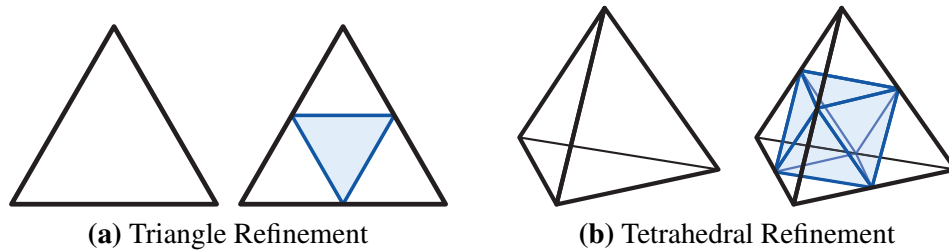
Figure 2.5: Regular refinement of a simplex. (a) A triangle is decomposed into four similar triangles. (b) A tetrahedron is decomposed into four similar tetrahedra and four non-similar tetrahedra covering an octahedral domain O (blue).

2.2.1 Regular refinement

The *regular refinement* of a d-dimensional cell τ is defined by adding vertices at all edge midpoints of τ and decomposing τ into 2^d disjoint cells covering τ [BSW83].

Regular refinement on (hyper)-cubic cells generates quadtree and octree decompositions as well as their higher dimensional analogues, in which all 2^d generated hypercubes share the midpoint of the refined domain as a common vertex (see Figure 2.4).

Regular refinement of a triangle σ generates four triangles that are similar to σ, of which, three triangles are incident to a vertex of σ, while the fourth triangle is defined by the three edge midpoints of σ (see Figure 2.5a). However, on simplicial meshes of *order* $d > 2$, regular refinement is not uniquely defined and can generate multiple similarity classes of simplices. For example, when refining a tetrahedron σ, the four tetrahedra incident to the vertices of σ are similar to σ, while the remaining four tetrahedra obtained by subdividing the octahedral domain O defined by the edge midpoints of σ, are not, in general, similar to σ (see Figure 2.5b).

Regular refinement does not generally create conforming adaptive refinements of a domain, i.e. where the cells can be at different levels of resolution. The *red/green refinement* scheme [BSW83] introduces a set of irregular *closure* refinement rules (*green*) to augment the regular refinement rules (*red*) for patching cracks between regular cells at different resolutions. An additional *balancing* constraint restricts the degree of decomposition between edge-adjacent cells, thereby reducing the number of green refinement rules that need to be considered.

12

(a) Triangle Bisection   **(b)** Tetrahedron Bisection
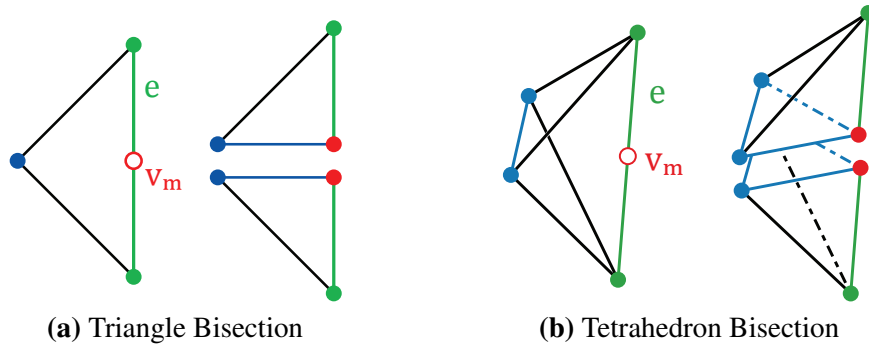
Figure 2.6: A simplex is bisected along the hyperplane defined by the midpoint $\mathbf{v}_m$ (red) of an edge $\mathbf{e}$ (green) and all vertices (blue) not incident to that edge.

2.2.2 Bisection refinement

The second class of nested refinement schemes is defined by *bisection refinement*, in which a cell is bisected along a hyperplane into two cells. When the cells are axis-aligned hyperrectangles which are bisected by axis-aligned hyperplanes, this generates *k-d trees* (see Section 3.1.3).

Alternatively, the *simplex bisection* operation bisects a d-simplex σ along the hyperplane defined by the midpoint $\mathbf{v}_m$ of some edge $\mathbf{e}$ and the $(d-1)$ vertices of σ not incident to $\mathbf{e}$. We refer to $\mathbf{e}$ as the *bisection edge* of σ. In contrast to regular refinement, which generates 2^d cells, simplex bisection generates only two simplices with disjoint interiors covering its domain. Figure 2.6 illustrates the simplex bisection rule in 2D and 3D.

Similarly to regular refinement, the decomposition induced by simplex bisection is not uniquely defined, as it does not specify which edge to bisect. Stable refinement schemes have been considered in which the bisection edge is determined through geometric or typographical properties (based on manipulating an ordered set of vertices). Simplex bisection is not a conforming refinement. Specifically, it introduces cracks between the neighbors of a bisected simplex σ that are incident to its bisection edge, and the two simplices generated during σ's bisection (see Figure 2.1b). This can be remedied by ensuring that all simplices incident to the bisection edge $\mathbf{e}$ are refined concurrently.

2.3 Modeling scalar fields

Scientific datasets are often given as a discrete set of points V in a domain $\Omega \in \mathbb{R}^d$ where one (or more) field values are associated with each point of V. The points in V can be regularly or irregularly distributed over the domain. In the former case, the data set is *structured*, while it is *unstructured* in the latter case. We assume that the domain Ω of a field F, denoted Ω_F, is a d-dimensional manifold with boundary.

We denote the value of a point $\mathbf{p} \in \Omega_F$ as $F(\mathbf{p})$. Such datasets can be modeled as a mesh Π_F formed by polytopic cells having their vertices at the data points. An interpolating function defined over the cells of Π_F provides values for all points of Ω_F.

We consider models based on simplicial complexes, defined over regularly sampled

(a) Isosurface   **(b)** Interval Volume
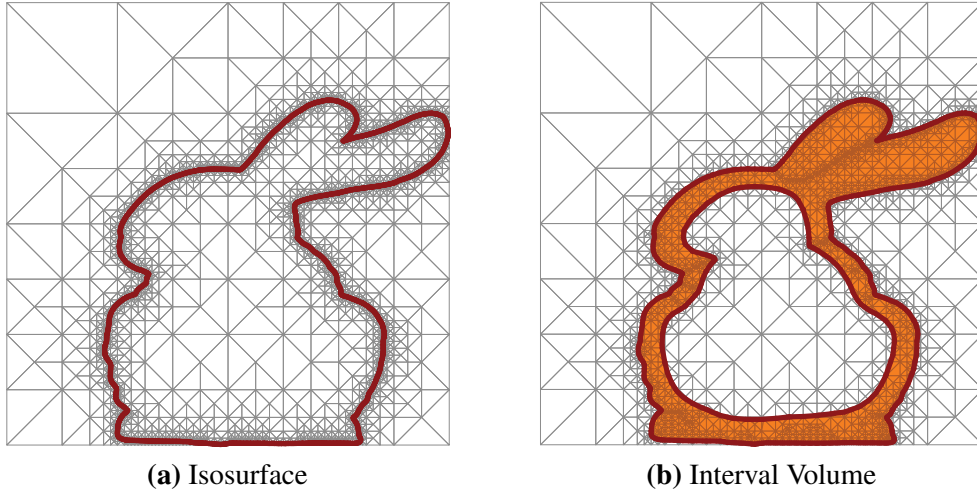
Figure 2.7: Isosurface (a) and interval volume (b) extracted from a nested triangle mesh.

rectilinear grids, in which linear interpolation is used over the simplices forming the mesh. The conforming property is important in this application since cracks in non-conforming meshes correspond to discontinuities in extracted meshes in correspondence to the boundary of adjacent cells of the underlying mesh.

2.3.1 Isosurfaces and interval volumes

One way of analyzing and visualizing such data sets is through surfaces of constant scalar value within the field. An *isosurface* S of *isovalue* κ within F, denoted as S_κ, is defined as

$$S_\kappa = F^{-1}(\kappa) = \{x \in \Omega_F | F(x) = \kappa\}. \tag{2.2}$$

The isosurface S_κ passes through all cells having at least one vertex whose associated field value is greater than κ, and at least one vertex whose value is less than κ. We consider such cells to be *active* or *non-empty* with respect to κ. Otherwise, the cell is considered to be *inactive* or *empty*. Figure 2.7a shows an example of an isosurface extracted from a nested triangle mesh.

Once a particular isovalue κ is chosen, it implicitly defines a binary-valued *sign field* B_κ on the vertices $\mathbf{v} \in V$ whose bits are set when corresponding scalar values of F are greater than κ, namely:

$$B_\kappa(\mathbf{v}) = \begin{cases} 1, & \kappa > F(\mathbf{v}), \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

An alternative method of visualizing regions of a dataset is through an *interval volume*, which is the set of points enclosed between two isosurfaces. Let $K := [\alpha, \beta]$ be defined as the interval between isovalues α and β, where $\alpha \le \beta$. Then the interval volume of *isorange K* within F, denoted as Σ_K or $\Sigma_{[\alpha,\beta]}$, is defined as

$$\Sigma_{[\alpha,\beta]} = F^{-1}([\alpha,\beta]) = \{x \in \Omega_F | \alpha \le F(x) \le \beta\}. \tag{2.4}$$

Once a particular isorange K is chosen, it implicitly defines a ternary-valued *sign field R_K* on the vertices $\mathbf{v} \in V$ whose values are defined by the relative values of F and K, namely:

$$R_K(\mathbf{v}) = \begin{cases} -1, & \text{if } F(\mathbf{v}) < \alpha, \\ 0, & \text{if } \alpha \leq F(\mathbf{v}) \leq \beta, \\ 1, & \text{if } \beta < F(\mathbf{v}). \end{cases} \qquad (2.5)$$

Consequently, an interval volume mesh is bounded by two surfaces: the *lower surface* corresponds to the isosurface of isovalue α while the *upper surface* corresponds to the isosurface of isovalue β. Thus, an interval volume passes through all cells that intersect one, or both, isosurfaces, or that lie between the two surfaces. Figure 2.7b shows an example of an interval volume extracted from a nested triangle mesh.

Note that for a given isorange $K = [\alpha, \beta]$, if α is below the minimum value in the range of F (or alternatively, if β is greater than the maximum value in the range of F), then the lower (upper) surface will not exist. This enables an efficient volumetric representation for a solid object on three-dimensional domains. Similarly, if $\alpha = \beta$, the interval volume degenerates to a standard isosurface.

Within each simplex σ, the sign field can be used to unambiguously triangulate the intersection of the isosurface (or interval volume) with σ. We call this intersection the isosurface (or interval volume) *patch* embedded within σ. Note that, when using linear interpolation, the vertices of the patch are only generated along *active edges* of σ.

2.4 Multiresolution models

Multiresolution models based on the decomposition of a shape into a simplicial mesh compactly encode a large number of different mesh-based representations of the shape. A general dimension-independent framework for mesh-based multiresolution representations, referred to as the *MultiTessellation model* [DPM97, DM02], has been shown to encompass the vast majority of multiresolution models developed in the literature. The three components of a mesh-based multiresolution model of a shape are:

- a coarse mesh Γ_b, called the *base mesh*,

- a set of *modifications U*, and

- a *dependency relation R* defined on the modifications in U.

Each modification specifies a local change to a mesh. It consists of replacing a subset Γ of its cells with another set of cells Γ' and is denoted as $u = (\Gamma, \Gamma')$. Each cell γ in Γ must either be part of the base mesh, or be created by exactly one modification in U. We are typically interested in modifications such that both Γ and Γ' are conforming meshes and the combinatorial boundary of Γ consists of the same set of cells as that of Γ'.

A *direct dependency* relation R is defined over the set U of all modifications as follows: a modification $u_1 = (\Gamma_1, \Gamma_1')$ directly *precedes* a modification $u_2 = (\Gamma_2, \Gamma_2')$ if and only if some cell of Γ_1' is also in Γ_2, i.e., if u_2 removes some cell inserted by u_1. The transitive closure of the dependency relation can be proven to be a partial order over set

U [DM02]. The direct dependency relation can thus be described as a Directed Acyclic Graph (DAG), in which the nodes represent modifications and the arcs represent direct dependency links. We call this graph the *dependency graph* of the model.

Thus, a *multiresolution model* $M = (\Gamma_b, U, R)$ provides a compact way of encoding all conforming meshes that can be obtained by recursively applying the modifications in U to the base mesh Γ_b. Each such mesh is in one-to-one correspondence with a subset of U that is *closed* with respect to relation R. A subset U' of the modifications in U is called *closed* if, for each modification u in U', all predecessors of u, with respect to R, belong to U'. The collection of all closed sets of modifications in a multiresolution model M defines the complete set of meshes that can be extracted from M. By applying all the modifications in U to the base mesh Γ_b, we obtain the *mesh at full resolution* described by M.

2.4.1 Selective refinement

Selective refinement is the basic query operation on a multiresolution model in terms of which all application-dependent queries can be expressed [LRC$^+$02, CDM$^+$04]. A selective refinement query extracts the mesh of smallest size from a multiresolution model based on a user specified predicate referred to as the *selection criterion*. This is equivalent to computing the closed set of modifications from U necessary to extract a mesh Σ, of minimal size, satisfying the specified criterion. The selection criterion can be based on many factors, including approximation error, field-values (e.g. isosurface extraction), proximity to an object of interest (e.g. view-dependent and Region of Interest (ROI) queries) and perception (e.g. screen-space error).

Selective refinement is performed by traversing the dependency graph describing the multiresolution model in a top-down manner starting from the base mesh; in a bottom-up manner starting from the mesh at full resolution; or incrementally from an already extracted mesh. The status of the refinement process is described by a cut on the arcs of the dependency graph, called the *active front*, which separates the set of modifications that have been applied from those that have not.

Chapter 3

State of the art

In this chapter, we review the state of the art approaches for decomposing a regularly sampled domain, with an emphasis on those based on nested simplex bisection over a regular grid. Additionally, we review marching methods for isosurface and interval volume extraction and hierarchical approaches for scalar field visualization.

Nested meshes have been used for various applications within the field of visualization. Examples include quadtrees and octrees formed by squares and cubes [Sam06], tetrahedral meshes generated by the *red/green* tetrahedron refinement [GLE97, Bey00] or meshes formed by tetrahedral and octahedral elements [GG00].

When a nested mesh is used as a spatial index for point data the decomposition can partition the data or the domain [Sam06]. When the subdivision partitions the data, the resulting structure depends on the insertion order of the elements. However, when the partitioning is based on the domain, the resultant structure is independent of the insertion order. In Sections 3.1 and 3.3 we focus primarily on approaches that regularly decompose a domain. We present a more comprehensive survey on the decomposition models and applications of simplex and diamond hierarchies in [WD10d, WD11], where we interpret these structures through the notions we introduce in this thesis.

We enumerate and classify the approaches according to whether their primary function is as a spatial partition of the domain, as a variable resolution representation or as a multiresolution device. The recent survey by Knoll [Kno06] classifies octree-based approaches according to their applications for visualization, including surface extraction, Direct Volume Rendering (DVR) and ray tracing. It discusses some approaches to point location and neighbor finding on pointerless octrees. In [WD08a], we classify approaches for the analysis and visualization of time-varying volume data by distinguishing between approaches that treat the temporal dimension as slices of a three-dimensional domain and those that treat the temporal and spatial dimensions equally, yielding a four-dimensional representation of the domain.

3.1 Domain decompositions

In the following, assume we have a domain $\Omega \subset \mathbb{R}^n$, which we will partition into a set of (possibly overlapping) cells covering the domain. Each such cell c acts as a *container* or *bucket* for data located within the domain of c, such as the points of a scalar field. Cells can have their own internal data structure to organize their indexed items. For example, a cell's data structure could be an (unsorted) linked list of items or it could be more complex, such as a sorted tree.

3.1.1 Uniform grid

Conceptually, the simplest way to subdivide a domain is to partition it into a grid containing (hyper)-rectangular cells of uniform size. Since the cells are uniform in size, and contain the same volume, they can be implicitly indexed. Although this method implicitly partitions all of $\mathbb{R}^n$, optimizations can be utilized for bounded domains.

For example, assume we are dealing with a 2D domain covering $[0, m] \times [0, n]$, where $m, n \in \mathbb{N}^+$, and each cell covers a 1×1 square. Then the lower left corner of the cell c covering an arbitrary point $p = (x, y)$ in the domain can be easily determined from the integer component of x and y. E.g. if $p = (5.78, 4.23)$, then p is located in the cell whose lower left coordinates are $(5, 4)$. A straightforward data structure for a bounded uniform grid is to store the elements in a multidimensional array.

When dealing with sparse datasets within bounded or unbounded domains, hashing can provide an efficient data structure. However, the efficiency is highly correlated with the appropriateness of the hashing function for the data distribution.

Alternatively, when the data is not uniformly distributed, a more efficient representation might be obtained by translating the grid lines along the axes. In such a case, the data is no longer accessible with constant access, and an access structure such as linear scales [Sam06] can be used to efficiently locate points.

3.1.2 Quadtrees, octrees and 2^d-trees

2^d-*trees* are hierarchical spatial decompositions of $\mathbb{R}^d$ based on the regular refinement of hypercubes. Let Ω be the (bounded or unbounded) domain of the 2^d-tree T, then the interiors of the d-cubes of T are pairwise disjoint and cover Ω.

A tree-based nested decomposition of the domain can be defined recursively. Let the *root* of the tree be the cell covering a hypercubic domain $\Omega \in \mathbb{R}^d$. For a tree rooted at node k, let its *children* be the 2^d d-cubes with disjoint interior where the vertices of each child's diagonal contains one of the corner vertices of k and an interior point p of k. The 2^d children of a node k are referred to as *siblings* and k as their *parent*. If p is always chosen as the midpoint of k, as we consider here, then the tree is *regular*.

We now define some properties of the 2^d-tree hierarchy. Let LEVEL(k) be the *level* of a node k. Then the level of a child node c of k is define as LEVEL(k) + 1, and the level of the root r of T is defined as LEVEL(r) = 0. The maximum level LEVEL$_{Max}$(T) of an 2^d-tree T is defined as the maximum of the levels of all nodes in T, i.e, LEVEL$_{Max}$(T)= max{LEVEL(k)}, where k is a node of T.

A node k in an 2^d-tree T is considered to be a *leaf node* if it has no descendants in T, otherwise it is an *internal node*. If all leaf nodes of a 2^d-tree T have the same level, then T is *complete*. Similarly, the *resolution* of the tree is defined as the distance between neighboring leaf nodes of level LEVEL$_{Max}$(T). If all leaf nodes in a 2^d-tree T have the same resolution, then we say that T has *uniform resolution*, otherwise, T has *variable resolution*.

If the intersection of nodes k and j is a facet (i.e. a $(d - 1)$-dimensional face) of k or of j then k and j are denoted as *facet neighbors* or simply as *neighbors*. If the intersection of the interiors of nodes k and j, where LEVEL(k) < LEVEL(j) is nonempty then k is called

an *ancestor* of j and j is a *descendant* of k.

3.1.2.1 Balanced 2^d-trees

In some applications, it can be useful to limit the variability of subdivision between neighboring cells. A *balanced 2^d-tree* is a generalization of the restricted quadtree, introduced by van Herzen and Barr [VHB87], where the level of neighboring cells can differ by at most one. Sivan and Samet [SS92, Siv96] first use such decompositions as a spatial data structure, specifically, for modeling terrain datasets, and study different rules for subdividing the squares into triangles.

Bern and Eppstein [BEG94] 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 [Moo95] identifies different types of neighbors on which to balance a nested hypercubic mesh and uses this to find optimally tight bounds on the cost of balancing quadtrees and an upper bound for balancing nested hypercubic meshes of $O(3^d)$ in three and higher dimensions.

Due to the increasing size of scientific datasets, there have been several out-of-core [TO04, TOL04] and parallel [SSB08] algorithms introduced for balancing large octrees.

An appropriately defined set of local refinements of the nodes of a balanced 2^d-tree can be used to patch the cracks in the decomposition. In 2D, these meshes can be patched using bisection-based rules [VHB87, SS92, Siv96, RHSS98] or through Delaunay-based rules [ZS01, RMSS01], where the triangulation cases are explicitly defined. An advantage of the latter in 2D, is that it does not require the insertion of extra vertices, referred to as *Steiner vertices* [dBSvKO97], to generate conforming meshes. Figure 3.1 illustrates the possible Delaunay-based and bisection-based cases for triangulating a square, with vertices at the midpoints of some of its edges.

In 3D, Plantinga and Vegter [PV07] propose a Delaunay-based triangulation for edge-balanced octrees in which two cubes sharing an edge must be balanced. In this approach, the faces are triangulated according to the 2D Delaunay cases (e.g. [ZS01]) and each cube is tetrahedralized by connecting the face triangulation to the cube's center. Greaves et al. [GMBW96, GB99] propose bisection-based cases in 3D, but do not provide details of the triangulation.

Triangulation cases for the facet-balanced hypercubic meshes of arbitrary dimension were proposed by Castelos et al. [CNS$^+$06] in the J_1^a triangulation. In this scheme, a vertex is inserted at the center of every hypercube and the simplices within a hypercube h are defined by the edges connecting this vertex to the vertices on the boundary of h. Note that facet-balanced meshes are the least restricted of the balanced meshes, and that lower-dimensional faces of facet-balanced meshes can have neighbors that differ in resolution by more than a factor of two (see Figure 10.1 in Chapter 10).

In Chapter 10, we introduce a hierarchy on balanced hypercubic meshes and propose a bisection-based triangulation for edge-balanced hypercubic meshes of arbitrary dimension, based on our model and encoding scheme for diamond hierarchies. Given an
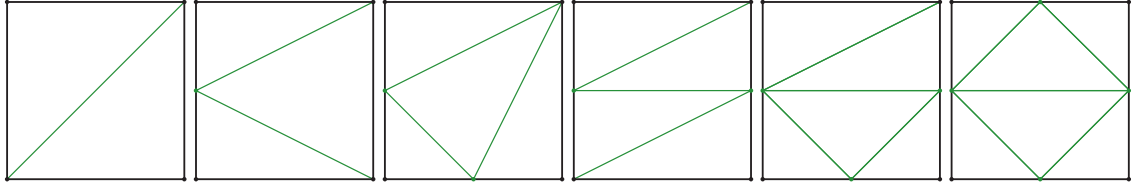
(a) Delaunay-based triangulations of a square
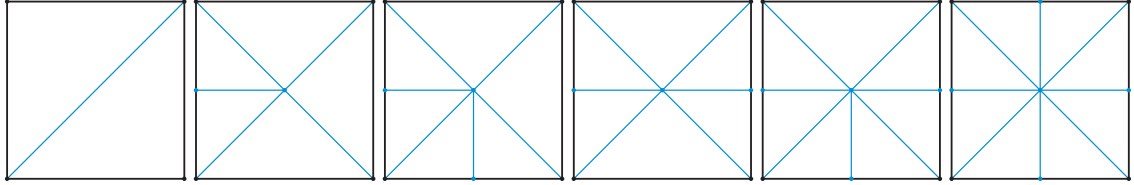
(b) Bisection-based triangulations of a square

Figure 3.1: Unique triangulation cases (up to symmetry) for squares (2-cubes) based on Delaunay triangulation (a) and bisection triangulation (b).

edge-balanced hypercubic mesh, the J_1^a triangulation produces the same domain decomposition as our algorithm, but is represented using simplices rather than diamonds, and thus requires more storage space.

Alternatively, conforming representations have been proposed for generating adaptive quadrilateral or hexahedral meshes from a balanced quadtree or octree, respectively, based on explicit decomposition rules for neighboring cells of different sizes [Sch96]. These rules are based on bisection (2:1) or trisection (3:1) of the hypercube edges. Recent work has focused on simpler decompositions and implementations for quadrilateral [Gar09] and hexahedral meshes [ISS09]. To the best of our knowledge, such decomposition rules have not been generalized to higher dimensional domains.

Compared to simplicial decompositions of balanced hypercubic meshes, 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).

3.1.2.2   MX-2^d-trees

In some applications, it is important to store all data at the maximum resolution of the 2^d-tree. However, to reduce the storage requirements, empty nodes can be aggregated. Let T be a 2^d-tree with $\text{LEVEL}_{Max}(T) = \ell$. Then T is an *MX-2^d-tree** if all leaf nodes containing data are located at level ℓ [HS79, Sam06]. Let j be a leaf node in an MX-2^d-tree that does not contain data, then $\text{LEVEL}(j) \leq \ell$. An MX-$2^d$-tree can be generated in a bottom-up manner from a complete 2^d-tree by merging 2^d siblings that do not contain data. If any of the siblings or one of their descendants contain data they cannot be merged.

---

*The prefix *MX* is based on an analogy to sparse matrices, in which adjacent zeros can be aggregated [Sam06].

Since all nodes containing data are leaf nodes at the maximum resolution, MX-2^d-trees can be considered to be uniform resolution.

3.1.2.3   PR-2^d-trees

If we remove the restriction that leaf nodes containing data need to all be at the maximum resolution, then we arrive at a variable resolution 2^d-tree called the *Point Region-2^d-tree* (PR-2^d-tree) [Sam06]. In a PR-2^d-tree, leaf nodes typically contain a single item and are located at the deepest level containing no other data below it. Alternatively, nodes in a *bucket PR-2^d-tree* can contain a predefined number of items [Sam06]. Thus, PR-2^d-trees typically require significantly fewer nodes to represent the same dataset as an MX-2^d-tree.

One disadvantage of PR-2^d-trees is that since the decomposition process is regular the maximum resolution can (in the worst case) be determined by the minimum distance between two data items. Thus certain data distributions can yield unbalanced trees.

3.1.2.4   Pyramids

2^d-trees typically contain all data in leaf nodes, which are accessed via the internal nodes. Some applications require aggregate data to be stored in internal nodes as well. When all leaf nodes are contained at the finest resolution, this structure is called a *pyramid*. When some leaf nodes are located at coarser resolutions, we denote this structure as a *truncated pyramid*.

Such structures can accelerate queries e.g. by culling irrelevant regions of the domain. Another application is in approximating the data contained deeper within the tree. In such a case, the structure is considered a multiresolution representation of the data.

3.1.3   K-d trees

Another popular recursive decomposition of a (hyper)-rectangular domain Ω is to subdivide each node into two (hyper)-rectangles along an axis aligned hyperplane. Such a tree is referred to as a *k-d tree* [Ben75]. A benefit of k-d trees is that they are always binary trees regardless of the dimension of the domain Ω. Furthermore, since nodes subdivide into two nodes rather than 2^d nodes, k-d trees are better able to adapt to the data distribution than 2^d-trees. The choice of axis to subdivide can be determined from the data or can be cycled. Due to the subdivision rule, k-d trees are rarely conforming.

3.1.4   Nested simplicial meshes

Recall from Section 2.2 that the two primary categories of nested simplicial meshes are those based on regular refinement or on simplex bisection, and that both approaches can lead to unstable refinement if the simplicial decompositions are not carefully chosen. Thus, researchers have proposed *geometric* refinement rules, which use the geometry of the simplex to determine the decomposition, and *typographical* rules, which manipulate the order of the vertices of the simplex to determine the decomposition. In the former case, the resulting decompositions are not affine invariant, while the latter case can require

an expensive initialization process. A comprehensive review of simplicial refinement strategies is presented by Bey [Bey00].

3.1.4.1 Regular refinement

Regular refinement always generates stable triangle meshes in which all triangles are similar to their generating triangle. In 3D, the octahedral domain O defined by the edge midpoints of a refining tetrahedron σ requires a decomposition into four tetrahedra that are not similar to their generating triangle in the general case (see Figure 2.5b). Zhang [Zha95] introduces a geometric refinement rule in which O is decomposed into four tetrahedra along its *shortest* diagonal. This generates a *stable* refinement when the angles of the initial mesh are non-obtuse. Other geometric choices, such as refinement along the octahedron's longest diagonal, do not yield stable refinements. Bey's typographical scheme [Bey95] is based on maintaining a specific vertex ordering during refinement. An interesting modification is the *tetrahedral/octahedral* regular refinement scheme [GG98,GG00,CQ06] in which the octahedra generated by regular tetrahedral refinement are also treated as primitives. In this scheme, each tetrahedron is refined into four similar tetrahedra and a single octahedron, while each octahedron is decomposed into six similar octahedra incident to its vertices, and eight similar tetrahedra corresponding to its truncated triangular faces.

Bey shows [Bey00] that his tetrahedral refinement scheme [Bey95] as well as the 2D scheme of Banks et al. [BSW83] are instances of a *d*-dimensional typographical scheme introduced by Freudenthal [Fre42]. Moore [MW95] provides a consistent labeling and simplex enumeration algorithm for the 2^d simplices generated by regular refinement.

Since regular refinement does not generate conforming adaptive refinements of a domain, Banks et al. [BSW83] introduce the *red/green refinement* scheme in 2D, in which the regular refinement rules (*red*) are augmented by a set of irregular *closure* refinement rules (*green*) to patch cracks between regular cells at different resolutions. An additional balancing constraint restricts the degree of decomposition between edge-adjacent cells, thereby reducing the number of green refinement rules that need to be considered. This scheme has been extended to tetrahedral meshes in [Bey95]. The recent *RGB subdivision* scheme for triangle meshes [PP09] introduces *blue* refinement rules, to transition between triangles generated by regular (red) and irregular (green) refinements. The addition of the blue refinements rules enables red and green operations to be applied in arbitrary order.

3.1.4.2 Simplex bisection

In the case of simplex bisection, researchers have also proposed conforming refinement schemes to implicitly determine the bisection edge via *geometric* [Riv84, Riv91, PC00] or *typographical* [Mit91,Bän91,Mau95,Tra97] bisection rules. Rivara's geometric *Longest Edge Bisection (LEB)* scheme over triangle [Riv84] and tetrahedral meshes [Riv91] chooses the longest edge of the simplex as its bisection edge, while Mitchell's typographical *newest vertex bisection* [Mit91] in 2D chooses the edge opposite the most recently introduced vertex. This edge can be implicitly determined through a consistent ordering of the vertices, where, e.g., the newest vertex is always in the final position. This scheme follows

the pioneering work of Sewell [Sew72, Sew79] and has been generalized to tetrahedral meshes as well [Bän91, Kos94, LJ95, AMP00].

Maubach [Mau95] and Traxler [Tra97] generalize these approaches to d-dimensional domains through an implicit typographical scheme (different from, but equivalent to [Mit91] in 2D) that cycles between d rules. In particular, Maubach proves that, when applied to a hypercubic domain subdivided along its diagonal into $d!$ simplices, his scheme generates at most d *similarity classes* of simplices. We refer to nested meshes containing only simplices generated by Maubach's bisection scheme over a regular grid as *Regular Simplex Bisection (RSB)* meshes and provide a detailed overview of this scheme in Section 4.2.2.

Such approaches differ from *red/green refinement* approaches [Bey95] by requiring only one set of *regular* refinement rules (*red*) but not an additional set of *irregular* refinement rules (*green*) to guarantee conforming triangulations. Meshes generated through bisection are significantly more adaptive than those generated by regular refinement since each bisection only generates two new elements while each regular refinement generates 2^d new elements. Thus, the refinement rate for bisection can be viewed as being independent of the dimensionality of the domain [Pas02].

On the other hand, for irregular simplicial complexes, it is unknown if all simplicial d-complexes, $d > 2$, admit a typographical simplex bisection scheme [NSV09]. Kossaczký [Kos94] proposes an initialization refinement algorithm to ensure that arbitrary tetrahedral meshes are admissible, although this increases the number of simplex classes. Stevenson [Ste08] generalizes this approach to higher dimensions.

The containment hierarchy among the simplices generated by regular simplex bisection induces a natural tree representation, in which the nodes are simplices and the two children of a simplex σ are the simplices generated by bisecting σ. This relationship can be captured using a binary tree, often referred to as a *bintree* [DWS+97, EKT01, LRC+02], whose root is one of the $d!$ simplices subdividing a hypercubic domain Ω along its diagonal. Thus, a nested RSB mesh can be modeled as a forest of $d!$ binary trees, which we call a *hierarchy of simplices*. This model is often referred to as a *hierarchy of (right) triangles* when the domain is two-dimensional, and as a *hierarchy of tetrahedra* for three-dimensional domains. If the tree is encoded using an array, the parents and children of a tetrahedron can be implicitly determined by their array indices. This representation is used in [ZCK97, GR99, GP00, LDS01, MDM04].

We are often interested in generating crack-free, or *conforming*, meshes, since cracks in the mesh correspond to discontinuities in scalar fields defined on the vertices. Methods of ensuring continuity have been proposed based on symbolic *neighbor-finding* operations [Heb94, LDS01], *saturated* error metrics (in which the errors are monotonic with respect to the hierarchical dependency relation) [OR97, Paj98, GRW00] or an implicit clustering of tetrahedra sharing a common bisection edge into a *diamond* primitive [DWS+97, GDL+02, DM02]

A pointer-based explicit neighbor-finding algorithm for a hierarchy of triangles is presented in [LRC+02]. Hebert [Heb94] introduces a symbolic neighbor finding operation for tetrahedra within a nested tetrahedral mesh Σ. Thus, tetrahedra in Σ no longer require pointers to their four adjacent tetrahedra in Σ. However, since Hebert encodes the bintree hierarchy using pointers, some neighbor-finding operations require time logarithmic in the size of the mesh.

When a nested RSB mesh is encoded as a forest of simplex trees, each simplex can be indexed through a *location code* [Sam90, Sam06]. Assuming that each child of a simplex is labeled using a single bit as (i.e. 0 or 1), a location code of a simplex σ is encoded as a tuple containing the label of σ's root simplex, its depth in the bintree, and a binary string indicating the traversal path from the root of the bintree to σ. Location codes enable pointerless data structures such as hash tables and B-trees to index the simplices.

Lee et al. [LDS01] introduce a pointerless neighbor finding algorithm for hierarchies of tetrahedra based on location codes. In this scheme, each neighboring tetrahedron's location code can be found in worst-case constant time through hardware bit-shifting operations. This enables the extraction of tetrahedral meshes satisfying a given unsaturated error criterion using fewer tetrahedra in the same amount of time as it takes to satisfy a saturated error criterion. Lee et al. extend this approach to constant-time neighbor finding in four-dimensional *hierarchies of pentatopes* in [LDS04].

Atalay et al. [AM04, AM07] extend the symbolic approach of Hebert [Heb94] and Lee et al. [LDS01, LDS04] to arbitrary dimensions. They use hardware bit-shifting to find each neighbor's location code in $O(\log d)$ time, and use the local subtree of simplices within a hypercubic domain to find the vertices of a simplex in constant time.

Maubach [Mau96] proves that the set of neighbors can be characterized by a connected $(d-2)$-surface and conjectures that this surface is simply-connected. Although the neighbor-finding algorithm must run $O(|Neighbors(\sigma)|)$ times for conforming bisections to a simplex σ, to the best of our knowledge, there have been no previous attempts to describe the number of such neighbors of a d-simplex on a regular grid. In Chapter 4, we prove that there are $O(d!)$ such neighbors, and cluster these into $O(d)$ sets of simplices that are simultaneously generated for efficient extraction and encoding of conforming modifications to an RSB mesh.

Gerstner et al. [GP00] pre-compute a *saturated* error for each vertex splitting a tetrahedron in a three-dimensional RSB mesh. The saturation condition implicitly forces all longest-edge neighbors to split, thus ensuring a conforming mesh. This scheme does not enable general navigation on the extracted meshes but it is suitable for parallel implementation [GR99] and front-to-back traversal [Ger02].

An alternate approach is to cluster the set of d-simplices that share a bisection edge into a new primitive, referred to as *diamonds*. Efficient encodings of diamonds have been developed in 2D [LP02, HDJ05] and 3D [GDL$^+$02], where the regularity of the updates and vertex distribution enables an implicit encoding of the geometric and hierarchical relationships among the simplices.

Pascucci's Slow Growing Subdivision (SGS) scheme [Pas02] generalizes the diamond subdivision paradigm to general d-dimensional meshes. In the SGS scheme, a d-dimensional cell of class i is subdivided by inserting a vertex v at its center and replaced with pyramid-shaped cells with apex at v and bases defined by its facets. All pyramids sharing the same $(d-i-1)$-dimensional face of a class 0 cell are then merged into a new cell of class $(i+1) \mod d$. This paradigm provides intuition for the subdivision process, but not for the geometric shape of a diamond or the complexity of its corresponding simplicial complex.

We generalize the notion of diamonds to arbitrary dimensions as a cross product of two simplicially decomposed hypercubes, providing a constructive definition for all

classes of diamonds (see Chapter 4 and [WD09a]). We introduce the *supercube* as a high-level primitive for regular simplex bisection (see Chapter 5 and [WD08c, WD09b, WD11]). Supercubes provide a means of analyzing all the relationships among the simplices and diamond generated through successive RSB operations, as well as an efficient means of associating information with sparse subsets of these elements. We introduce a pointerless encoding for diamonds as well as an implicit representation for the multiresolution hierarchy in Chapter 6. Once a conforming RSB mesh is extracted, there are many applications which require efficient navigation of the mesh's local connectivity. We introduce algorithms for navigating 2D and 3D diamond meshes in Chapter 8 based on only the presence or absence of vertices and diamonds within the mesh.

3.2 Marching cells

3.2.1 Isosurfaces

A popular divide and conquer approach to isosurfacing scalar fields is the Marching Cubes algorithm [LC87]. In this, and related algorithms (see [NY06] for a recent survey), the domain is subdivided into cubic (or, more generally, polytopic) *cells* which are contoured locally.

This is accomplished by first labeling the vertices of each cell based on the relative value of their field values compared to a specified isovalue. Edges that intersect the isosurface are referred to as *active*, and their vertices have different labels. *Active cells* contain at least one active edge. The *isosurface patch* extracted from an active cell consists of one or more simply connected triangulated polygons whose vertices lie on the active edges. A globally *conforming* isosurface is achieved by ensuring that each isosurface patch is crack-free and that the interface between adjacent patches are crack-free.

The vertex generation and patch triangulation steps are typically accelerated through precomputed lookup tables based on the configuration of a cell's vertex labels. Assuming a predetermined order for the vertices of a cell c, the *bitpattern* of c is the ordered list of labels of c's vertices. When the cells are homogeneous this can lead to an efficient encoding in a lookup table, i.e. for cubic cells defined by 8 vertices, this leads to $2^8 = 256$ cases.

To simplify the analysis and ensure that the cases have the correct geometric and topological structure, it is common to classify all possible configurations of isosurface patches that can occur within a cell. These approaches typically use a combination of symmetry, reflection and topological considerations to reduce the configuration space to a restricted number of cases. Ambiguous configurations [Dur88], i.e. those that admit more than one valid triangulation, can be resolved using higher order interpolation functions [NH91, Nie03], gradient calculations [WVG90] or neighboring cells [Nie03].

The Marching Tetrahedra approach [PT90, WVG90, NB93, TPG99, Pas04] operates on tetrahedral meshes, which do not suffer from ambiguities due to the use of linear interpolation. However when the tetrahedra are obtained through subdividing cubic cells (into 5 or 6 tetrahedra each), the output surface will have more triangles than a surface generated using Marching Cubes (up to a factor of four [CSA03]). Irregular tetrahedral meshes can yield variable resolution isosurfaces.

Other Marching methods include the Marching Octahedra and Modified Marching Hexahedra methods [CSA03], which operate on Body Centered Cubic (BCC) lattices and the Generalized Marching Method [HSSZ97] which operates on meshes with more than 2 vertex categories, as found in segmented volumes.

Algorithms for isosurface extraction from 4D scalar fields have also been proposed [RH99, BWC00], although they disagree on the number of cases for the 4-cube. These differences are reconciled in [BLS04], in which marching approaches are distinguished based on the types of cells, the dimension, the number of vertex categories and the symmetries used. Weigle and Banks [WB96, WB98] have proposed a recursive algorithm for isosurface extraction from simplicial meshes, counting 5 possible different cases for a 4-simplex. Bhaniramka et al. [BWC04] provide marching cases for convex cells of arbitrary dimensions.

An alternative set of intersection cases can be created for surfaces *dual* to those of the marching cubes, where a single vertex is generated within each active cell, and a single quadrilateral is generated for each intersected edge [Gib98, JLSW02]. A dual marching tetrahedra approach was recently proposed by Nielson [Nie08]. Since vertices can be freely placed within cells (rather than strictly on edges), dual methods can generate better approximations using fewer triangles. However, care must be taken to ensure that the resultant surfaces remain manifold [SJW07].

3.2.2 Interval volumes

Interval volumes were introduced concurrently by by Guo [Guo95] and by Fujishiro et al. [FMS95]. Guo [Guo95] introduced *interval sets* as a bridge between direct volume rendering (DVR) and indirect volume rendering (e.g. isosurface extraction) techniques. In this method, the boundary surfaces are approximated using *alpha surfaces* [EM94] and the volume is triangulated using a Delaunay triangulation.

Fujishiro et al. [FMS95, FMST96] extract interval volumes from cubic cells using an extension of the Marching Cubes algorithm [LC87]. Their cases are defined for the *half-regions* containing a single surface. When a cell contains both surfaces, mesh patches are generated on the fly by intersecting the two half-regions. Global mesh consistency is ensured through the use of Nielson and Hamann's asymptotic decider [NH91] cases rather than the original marching cubes cases.

Nielson and Sung [NS97] offer a case-based lookup table for interval volumes over tetrahedral meshes. They reduce the configuration space to 15 unique cases and provide a consistent global triangulation of the polyhedral interval volume patches.

Bhaniramka et al. [BWC00] create interval volume cases for cubes by first extracting an isosurface from a four dimensional hypercube and then projecting the isosurface into an interval volume in the cube. This concept is extended to 4D interval volume extraction using 5D isosurfaces in [JSW03]. Bhaniramka et al. [BZX⁺04] further extend this dimension lifting idea to the extraction of interval volumes from convex cells of arbitrary dimension and introduce several new visualization techniques for static and time-varying interval volume datasets. Since it is not feasible to represent all possible cases, they use lazy evaluation to dynamically generate cases on demand and cache the most frequently observed cases [BWC04].

Zhang et al. [ZBS03] introduce a technique for extracting adaptive interval volume meshes from volume data. Octree cells containing a boundary of the interval are triangulated using a modified dual contouring technique [JLSW02], while cells completely within the interval are triangulated by inserting additional vertices, referred to as Steiner points.

3.3 Hierarchical data structures for scientific visualization

Due to the size of extracted isosurfaces, there has been much research on accelerating the extraction process and on simplifying the resulting meshes.

There are two significant efficiency issues related to the marching cubes style of isosurface extraction. First, the algorithm wastes a significant amount of time processing *empty* or *inactive* cells, i.e. cells that do not intersect the isosurface. Second, the extracted isosurface is typically over-triangulated, i.e. it contains too many triangles. This is problematic since it increases processing costs in all downstream applications, including rendering, without improving the fidelity of the surface. Thus, interactivity with such meshes can be limited.

3.3.1 Hierarchy as spatial access structure

The first issue can be remedied by creating an index of the active cells. There are two predominant indexing techniques: *hierarchical spatial indexing* and *value indexing*. The most widely-used spatial indexing technique is Wilhelms and Van Gelder's *Branch-On-Need Octree (BONO)* [WVG92] (sometimes referred to as the *Min-Max octree*), which provides hierarchical access to active cells by aggregating field values within nested regions and associates with each node of the octree the minimum and maximum field values of its descendants. BONO efficiently handles datasets whose dimension are not powers of 2, and uses a caching mechanism to minimize field value lookups of neighboring cells. Patches are only extracted from cells at the full resolution, thus ensuring the extraction of the same exact mesh as the original marching cubes algorithm [LC87].

In contrast, the value-based partitioning approaches [LSJ96, SHLJ96, CMM+97] attempt to minimize isosurface extraction time by reorganizing the data into a two-dimensional *span space* on the range of the field. In span space techniques, each cell is projected into a two-dimensional point whose coordinates are the minimum and maximum field values spanned by the cell. Efficiency is achieved through optimized data structures on the span space rather than the spatial coordinates of the cells. The *Near Optimal IsoSurface Extraction (NOISE)* algorithm [LSJ96] uses a k-d tree to efficiently extract an isosurface from a 3D volumetric data set in $O(\sqrt{n} + k)$ time, where n is the total number of cells, and k is the number of active cells. *Isosurfacing in Span Space with Utmost Efficiency (ISSUE)* [SHLJ96] improves the performance of the NOISE algorithm and provides a parallel algorithm by subdividing the span space into an axis-aligned lattice where each tile contains approximately the same number of cells.

Interval trees [Ede80] are another efficient method for performing range queries, and have been used by van Kreveld [VK96] to extract isocontours from triangulated terrain

data. Cignoni et al. [CMM+97] use interval trees to index cells in span space, and prove that their isosurface extraction algorithm operates in optimal $O(\log n + k)$ time.

3.3.2 Multiresolution field representations

In this section, we review multiresolution representations for scalar field visualization with a focus on approaches for interactive terrain rendering and isosurface extraction. We conclude with a review of the few techniques dealing with higher dimensional datasets (i.e. $d \geq 4$).

3.3.2.1 Two dimensional domains

The ability to extract variable-resolution representations of the terrain, such as having high resolutions in selected areas of interest or along a view frustum is a fundamental issue in interactive terrain processing. View-dependent representations are important for achieving interactivity in rendering. The recent survey by Pajarola and Gobbetti [PG07] presents a comprehensive review of approaches for interactive rendering of regularly sampled terrain datasets, including those based on RSB triangles and diamonds. Here, we discuss techniques based on RSB triangles and diamonds, as well as representations based on incomplete diamond hierarchies to deal with data points at the vertices of a sparse regular grid. We first review approaches applying either on a triangle-based or on a diamond-based hierarchy to a complete regular $(2^N + 1)^2$ grid of scalar values.

Evans et al. [EKT97, EKT01] introduce a representation of a terrain based on a hierarchy of triangles that they call *Right Triangulated Irregular Networks (R-TINs).*[†] They use *location codes* for encoding the nodes of a hierarchy of triangles and observe that conforming RSB meshes are balanced, i.e. neighboring simplices can differ in refinement depths by at most one. Specifically, neighbors along a triangle's hypotenuse can be at most one refinement depth higher in the hierarchy, while neighbors along the other two edges can be at most one refinement depth lower in the hierarchy. Thus, a single bit per edge is sufficient to determine which of the two possible triangles is present in the mesh. To aid in mesh extraction, they associate an error with each triangle, and note that this requires approximately four times as much storage as associating errors with the vertices.

Lindstrom et al. [LKR+96] were the first to consider the simplification dependency relation among the vertices of a nested RSB mesh. They define a triangle's *split vertex* as the vertex introduced during triangle bisection. Two triangles are *fused* when the inverse of simplex bisection operation is applied to them. Fusion removes the two children of a triangle and replaces it with their parent by removing the split vertex.

They observed that each split vertex corresponds to triangles in two different branches of the hierarchy of triangles (see Figure 3.2), and that cracks are introduced into the mesh if only one of those branches is fused. They propose a conforming triangle fusion operation that replaces both pairs of sibling triangles with their parent triangles (i.e. from Figure 3.2b to 3.2a).

For out-of-core access to large datasets, they store the terrain as a set of square blocks, each containing $(2^k + 1)^2$ samples (for some k), where samples on the boundaries

---

[†]A TIN (Triangulated Irregular Network) is an irregularly sampled triangle mesh describing a terrain.

of each block are duplicated. Their error metric only takes the value of the split vertex into account and is thus unable to guarantee global error bounds on the extracted mesh. For efficient rendering, they encode the entire mesh as a generalized triangle strip (i.e. one that allows swapping of vertices).

(a) Fused   **(b)** Split

Figure 3.2: A pair of triangles in an RSB must be bisected concurrently if they share the same bisection edge (blue edge in (a)). The *split vertex* (hollow blue) of the triangles in (a) depends on the four split vertices (hollow white) of the triangles in (b). Filled vertices belong to triangles while hollow vertices are the midpoint of a triangle's bisection edges.

In the *Real-time Optimally Adapting Meshes (ROAM)* approach, Duchaineau et al. [DWS+97] introduce a view-dependent incremental selective refinement for a hierarchy of triangles. Recall from Section 2.4.1 that incremental selective refinement is a variant of the basic selective refinement query in which the initial mesh can be a previously extracted RSB mesh. They also introduce *split* and *merge* operations, corresponding to conforming triangle bisection and fusion operations, respectively, and note that any RSB mesh can be obtained from any other one via a series of merges and splits. Frame-to-frame coherence is supported through the use of a dual queue system, where one queue holds mergeable triangles and the other holds refineable triangles.

Due to the shape of its domain, they introduce the term *diamond* to refer to the two triangles sharing a split vertex (see Figure 3.2a). To reduce visual artifacts incurred by refinements, they implement a *geomorphing* operator, that gradually moves the split vertex (which we refer to as its *central vertex*) of a diamond into its new location.

Later work, such as *ROAM Using Surface Triangle Clusters (RUSTiC)* [Pom00] *Cached Aggregated Binary Triangle Trees (CABTT)* [Lev02] and others [CGG+03a,HDJ05, BGP09], exploits the graphics hardware more efficiently through the use of *batched* updates to an RSB mesh. Specifically, in these methods, each *macro* update to the model corresponds to the insertion of triangles from several hierarchy depths lower in the hierarchy. These batched triangles are typically preprocessed into triangle strips for efficient streaming to the graphics card. Despite increasing the number of triangles required to satisfy a given selection criterion, these methods are able to reduce the processing overhead on the CPU.

Pajarola [Paj98] considers the refinement dependency relation for triangle bisection over a nested RSB mesh.[‡] By reversing the dependency relation arcs introduced by

---

[‡]Although this method is called *Restricted Quadtree Triangulation*, it uses the vertex dependency relation of diamonds, and the underlying triangulation does not correspond to a quadtree, so we classify it as a diamond-based approach.

(a) Original   **(b)** Level 1   **(c)** Level 2   **(d)** Level 3

Figure 3.3: Batched triangulations at three levels of resolution. Each additional level has an increased complexity of 2^d simplices.

Lindstrom et al. [LKR⁺96], he observes that conforming modifications occur when both triangles sharing a base vertex are bisected concurrently. Thus, each vertex depends on the split vertices of two triangles higher in the hierarchy.

Pajarola provides a few simple operations to extract a vertex's level as well as the *orientation* of its corresponding bisection edge directly from its coordinates. Specifically, the *level* of a vertex can be derived from its spatial location by finding the index of the least significant non-zero bit in the binary representation of its x and y coordinates. If both the x and y coordinates contain a non-zero bit at this location, the vertex is located at the center of a quadtree node. Otherwise, it is located at the midpoint of one of its edges.

Puppo [Pup98] describes how the vertex dependency relation can be viewed as a special case of the *MultiTesselation (MT)* framework of De Floriani et al. [DPM97].

Hebert [Heb98] models a nested RSB mesh in terms of two interlaced quadtrees: The standard axis-aligned quadtree and a *quincunx quadtree*, rotated 45° relative to the standard lattice.§ The center points of nodes from the quincunx lattice coincide with edge midpoints while those of the axis-aligned quadtree coincide with square midpoints. Hebert also provides an indexing scheme for the vertices that is equivalent to location codes for quadtrees [Sam06].

Lindstrom and Pascucci [LP02] independently analyze the vertex relationships of a nested RSB mesh in terms of two interleaved quadtrees. They introduce a hierarchical indexing scheme for the diamonds that follows the traversal order of the hierarchy. Their error metric includes an approximation error as well as a nested bounding sphere radius for distance-dependent refinement (as first introduced by Blow [Blo00]). In the context of out-of-core rendering, they demonstrate that hierarchical indexing performs an order of magnitude better than array indexing which performs an order of magnitude better than block-based indexing.

Cignoni et al.'s *Batched Dynamic Adaptive Meshing (BDAM)* [CGG⁺03a] follows up on the clustering idea of RUSTiC [Pom00] by encoding a set of modifications to an irregular triangle mesh. Thus, they use the containment hierarchy induced by bisections as an efficient spatial access structure for conforming irregular triangle meshes describing terrains. This enables efficient transmission of terrain data to the GPU in batches. A corresponding texture hierarchy, encoded using a quadtree, enables color texture mapping

---

§A quincunx is a geometric arrangement of five dots (⚄) that commonly appears on dice and playing cards.

which produces more realistic images using fewer triangles.

Gerstner [Ger03b] introduces an implicit optimally tight octagon-shaped bounding hierarchy for distance-dependent rendering based on the limit shape of the domain covered by a diamond's hierarchical descendants [Tan95, BLV03] (see Figure 3.4). He also shows how the various independent saturated error metrics can be combined at runtime, to achieve dynamic refinements to the saturated selection criterion.

(a) Octagonal descendant domain   (b) Nested descendant domains
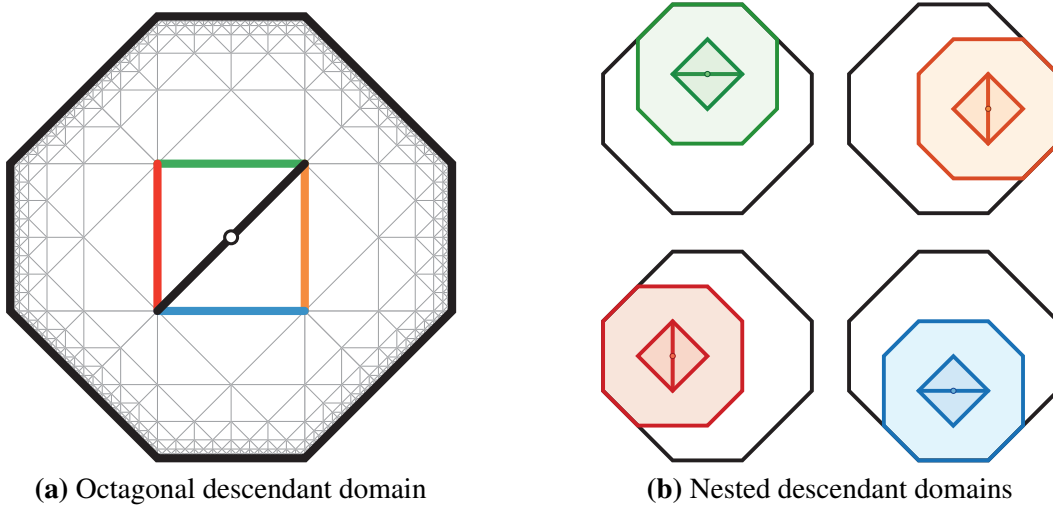
Figure 3.4: (a) The limit shape of a diamond's hierarchical descendants is an octagon with the edge lengths of its triangles. (b) The descendant domains of its four children (with colored central vertices) are nested within its octagonal descendant domain.

Hwa et al. [HDJ04, HDJ05] present a batched update approach based on diamonds and provide closed-form equations for the offsets to vertices and neighbors of a diamond as well as its parents and children. They improve on the spatial access structure of BDAM by using the regular clustering of RUSTiC. Furthermore, their textures are encoded using a diamond hierarchy, and are thus more closely aligned with the hierarchy defined by the height values than the quadtree of BDAM [CGG⁺03a].

Gobbetti et al. [GMC⁺06] demonstrate that batch-updated diamond-based approaches can be competitive with the non-conforming state of the art grid-based approaches [LH04] in terms of compression rates and runtime performance. Lindstrom and Cohen [LC10] propose a batch-updated variable-rate GPU compression scheme for RSB meshes in which the batches are regularly refined. The recent work by Goswami et al. [GMBP10] utilizes a batch-based hierarchy for parallel rendering of large datasets across multiple machines and displays.

In [YWD11], we introduce parallel algorithms for extracting conforming triangle meshes on the GPU from a two-dimensional hierarchy of diamonds in a parallel terrain processing framework.

Compact encodings for incomplete hierarchies. All the above methods exploit the regularity of the data distribution by encoding the multiresolution model as a regular

grid where all vertices are present. Consequently, many of the above techniques exploit the regularity of the dataset to reduce the geometric and topological overhead of the multiresolution model. Since all vertices in a $(2^N + 1)^2$ terrain are present, they can be stored linearly and accessed using row-major ordering (or a more complicated indexing scheme [LP02, Sam06]). Furthermore, the implicit dependency relation can be used to locate parents, children and neighboring triangles, enabling pointerless representations.

However, in cases where values for portions of the domain are unavailable or the field is oversampled, efficient representations of such *incomplete hierarchies* can significantly reduce storage requirements and processing times. Although simplex and diamond hierarchy approaches that represent the dependency relation explicitly, i.e. with pointers, can handle incomplete hierarchies, we focus here on the approaches that exploit the regularity of the sampling to reduce the hierarchical overhead incurred for each retained sample in the dataset.

To the best of our knowledge, the only method to encode an incomplete nested RSB mesh has been proposed by Gerstner [Ger03a]. He introduces a compression method based on a linearization of the domain using a Sierpinski space-filling curve associated with the complete hierarchy of triangles. He uses a containment hierarchy as a multiresolution representation, and encodes in each node of the resulting partial tree the number of nodes that need to be skipped if the node is not to be refined. Variable-size relative pointers are used to indicate the number of bytes to skip if a node is not refined, and an average overhead of 3 bytes per vertex is reported. However, since each node is accessed via pointers stored in its parents, this representation does not provide random access to the data in the model. Gerstner [Ger03a] also introduces the notion of incorporating higher resolution blocks of samples into a coarse dataset. Although few details are given regarding how to accomplish this, he indicates that this requires the addition of some interpolated samples to align the new data with the hierarchy.

Our work on supercubes (see Chapters 5 and 7 as well as [WD08c, WD09b]) implicitly clusters data associated with the diamonds of an incomplete hierarchy. In contrast to the approach of Gerstner [Ger03a], supercube-based representations provide random access to the data associated with each diamond. Empirically, the supercube encoding has an overhead of less than one byte per encoded sample, while the approach of Gerstner exhibits an average overhead of three bytes per sample. Both approaches provide a means of embedding higher resolution regions of data within the multiresolution terrain model (see Section 7.5.4).

3.3.2.2 Three dimensional domains

In this subsection, we review approaches for nested RSB meshes in 3D, with a focus on methods used to model multiresolution volume datasets and to enable efficient extraction of isosurfaces.

Zhou et al. [ZCK97] extend the triangle fusion algorithm of Lindstrom et al. [LKR+96] to create a conforming *tetrahedral fusion* operation over nested RSB tetrahedral meshes. They compute the dependency relation of the vertices through a recursive application of tetrahedral bisection to the initial six tetrahedra subdividing the cubic domain. The dependencies are explicitly encoded in a *dividing point* table, where the dividing point is

equivalent to the split vertex of [LKR⁺96] and to the central vertex in our terminology, and are used to extract simplified isosurfaces from the hierarchy.

One interesting feature of their approach is that it incorporates a method to ensure that the topology of the simplified surface matches that of the surface at full resolution. This is accomplished by disallowing fusion of tetrahedra whose bisection edge vertices lie on the same side of the isosurface, but whose split vertex lies on the opposite side of the isosurface.

Gerstner and Pajarola [GP00] note that the topology preserving approach of Zhou et al. [ZCK97] is too conservative. Although their proposed solution is based on bintrees, they consider changes to an isosurface's topology at the diamond level. To do so, they define isosurface cases within each diamond based on the relative values of a diamond's vertices and the given isovalue. Their saturated topology-based error metric encodes the range of isovalues in which the topology of the extracted mesh changes.

However, topology preservation limits the adaptability of the approach since all nodes in which the topology changes must be present in all extracted meshes. Consequently, surfaces with complicated topology can never be simplified beyond a certain point. To resolve this, they introduce a *topology control* mechanism which weights the topology metric at each node. As an example, they demonstrate how this can clean up a noisy isosurface by reducing the genus of the extracted mesh.

Gerstner and Rumpf [GR99] accelerate the extraction of isosurfaces using parallel processors and a view-dependent saturated error metric. Using this approach, they can extract isosurfaces up to 2.4 times faster with four processors than with one. Additionally, they introduce a mechanism to cull the isosurface backfaces through the use of a view-dependent curvature-based error criterion, reducing the size of the extracted isosurfaces by a factor of two. They note that the gradient of a vertex is necessary for smooth shading, but requires three times as much space to encode as the scalar values. Thus, rather than encoding the gradient, they compute it on the fly and cache the values in a hash table. In follow-up work [Ger02], Gerstner describes a hierarchical scheme to compute the gradient of a tetrahedron from that of its parent. When used in conjunction with a sorting scheme for tetrahedra based on the bisection's splitting plane, this enables back-to-front isosurface extraction, which they use to extract multiple transparent isosurfaces during a single traversal of the hierarchy.

Gregorski et al. [GDL⁺02] generalize the ROAM algorithm [DWS⁺97] to tetrahedral meshes using a diamond-based approach. They avoid the need to deal with boundaries by treating the domain as a 3-torus of resolution $(2^N)^3$. In contrast to the explicit dependency relation encoding of Zhou et al. [ZCK97], Gregorski et al. implicitly encode the dependency relation of the entire hierarchy in terms of scaled offsets from 26 *archetypal* diamonds, corresponding to the possible *oriented directions* of a diamond's bisection edge. Specifically, they identify eight diamonds whose bisection edge is aligned with a cube diagonal (referred to as 0-diamonds), twelve diamonds with bisection edges aligned with a face diagonal of a cube (referred to as 1-diamonds) and six diamonds with bisection edges aligned with cube edges (referred to as 2-diamonds).

They encode the locations of the vertices of a diamond as well as the central vertices of its parents and children as scaled *offsets* from its central vertex. Access to a diamond's entries in this dependency table requires only its central vertex, its level and its type (i.e.

orientation), saving 6–12 pointers (i.e. 24–48 bytes) per diamond compared to the encoding of Zhou et al. [ZCK97]. Although they indicate how the vertices and the dependency relation are recovered, they do not discuss how such offset tables are generated or how the orientation of a diamond is determined.

Our encoding of diamonds (see Chapter 6 and [WD08b, WD09a, WD10b]) extends this by providing an efficient means of determining a diamond's class and level as well as the orientation of its spine directly from the binary representation of its central vertex. We, therefore, only require the coordinates of a diamond's central vertex.

The diamond-based scheme of Gregorski et al. uses ROAM's [DWS+97] dual-queue selective refinement algorithm to exploit the frame-to-frame coherence between extracted meshes during view-dependent isosurface extraction. This is accomplished by initializing the extraction with the RSB mesh extracted at the previous frame. This scheme is also used during small adjustments to the isovalue. They compress the scalar values, field gradient and min-max ranges within each diamond from 19 bytes to 4 bytes. Additionally, they rearrange the data hierarchically [LP02] and use the operating system's virtual memory paging for cache-coherent out-of-core memory management. Recently, Gregorski et al. [GSDJ09] proposed a method to further accelerate view-dependent isosurface extraction through the use of occlusion culling [LH98].

Linsen et al. [LGP+04] use diamond connectivity [Pas02] as an adaptive subdivision basis for volumetric datasets. They use trilinear B-spine wavelets to downsample the dataset (i.e. rather than the more commonly used subsampling) to generate similar approximations using approximately 10-15% fewer tetrahedra.

Marchesin et al. [MDM04] consider the orientation *edge vectors* of a tetrahedron's edges, and prove that the components of any edge vectors of a tetrahedron at level k in a hierarchy of tetrahedra are either -2^k, 0 or 2^k. They exploit this to create an efficient enumeration algorithm for the points within a tetrahedron that is similar to a raster scan conversion algorithm. The algorithm first enumerates the axis aligned planar slices of a tetrahedron. Within each triangular slice, it enumerates all axis aligned lines. Finally, it enumerates the points within each line.

They use the nested hierarchy for view-dependent Direct Volume Rendering (DVR) and examine the implications of applying non-conforming bisections, which can be used to extract smaller meshes much faster than a conforming algorithm, but can introduce significant noise into the visualization. In some cases, though, the resulting image was determined to be of sufficient quality.

3.3.2.3 Higher dimensional domains

Approaches for higher dimensional domains have primarily focused on representations for time-varying volumetric datasets. The temporal dimension of such datasets can be treated as a set of values in the same 3D location [GSDJ04], or as a fourth spatial dimension [LPD+04, LDS04].

Lee et al. [LDS04, Lee06] extend their tetrahedral neighbor-finding algorithm to 4D *hierarchies of pentatopes*. As in the 3D case, they use hardware bitshifting operations to guarantee a worst-case constant time neighbor finding operation. One source of inefficiency in their approach is that they store an approximation error for each pentatope

(i.e. 4-simplex) rather than for each diamond. Results on small datasets of resolution 33^4 indicate that an unsaturated error metric can reduce the size of extracted RSB meshes by 1% compared to a saturated metric.

Gregorski et al. [GSDJ04], apply their diamond-based isosurface extraction framework [GDL$^+$02] to time-varying volumetric datasets modeled as a stack of volumetric datasets covering the same volumetric domain. They exploit the temporal coherence of the dataset by initializing the isosurface extraction for each new time-step with the RSB mesh extracted during the previous time-step. Additionally, they propose a batched update approach for the subtrees of tetrahedra within each diamond [Pom00] to reduce the granularity of each modification and accelerate hardware rendering of isosurfaces.

Linsen et al. generalize their $\sqrt[3]{2}$ approach [LGP$^+$04] to 4D grids with the $\sqrt[n]{2}$ scheme [LPD$^+$04]. This enables treating the temporal dimension in time-varying volume data as a fourth spatial dimension. An advantage of their approach is that their wavelet-based downsampling approach approximates the data at lower resolutions rather than subsampling the data. They then use volume rendering techniques to render the extracted hypervolume.

Atalay and Mount [AM07] use a hierarchy of pentatopes as a point-location structure to accelerate ray tracing of atmospheric effects. In this scheme, each ray is represented as a 4D point, and values for unrepresented points are interpolated based on the RSB decomposition. Compared to a non-adaptive approach, the hierarchy of pentatopes achieved a six times savings in time. Further optimizations were achieved by applying a lazy neighbor-finding algorithm to patch cracks locally where necessary, resulting in a further 3 times savings in extraction time and 9.3 times in space.

3.3.3 Adaptive representations for extracted meshes

Algorithms have also be proposed to obtain variable resolution surfaces by post processing a surface extracted from volume datasets. One approach is to first extract the mesh at full resolution (e.g. using Marching Cubes [LC87]) and to then simplify the mesh in a post-processing step [SZL92, GH97]. However, this requires processing time and storage for the isosurface at full resolution. Other approaches [WDSB00, GCBB01] involve generating a coarse topologically correct representation of the mesh from the volume dataset followed by a refinement process that satisfies certain constraints such as surface error and triangle aspect ratio. Such approaches can suffer (in speed or accuracy) from not taking the volumetric representation into account, e.g. the resulting meshes are no longer guaranteed not to self-intersect.

Another class of algorithms target adaptive grids to reduce the number of extracted isosurface primitives. However, care must be taken to prevent cracks in the surface when extracting from neighboring cells of different sizes. The key distinction of these representations is that they do not store the field values but, instead, they represent the sign field at each vertex of the hierarchical grid. Consequently, they only represent a single (variable- or multi-resolution) mesh and do not support scalar field operations such as distance computations and Boolean operations.

The two most common solutions to the cracking problem are to either effectively increase the resolution of the larger cells by using the values of the higher resolution cells

or to decrease the resolution of the higher resolution cells by using the interpolated values from the lower resolution cells rather than the actual field values. To reduce the number of possible cases, balanced octrees [VHB87, Siv96] are typically (but not always [KKDH07]) employed.

Muller and Stark [MS93] introduce a top-down *splitting box* algorithm yielding an adaptive isocontour. Edges of cells can contain more than two vertices, but can only contain a single label transition. Since only the values along edges are checked, internal features can be missed. Cracks are handled by removing higher resolution points (i.e. internal to cubes), effectively simplifying shared boundaries of adjacent faces at different resolutions to line segments.

In the approach of Shu et al. [SZK95], the domain is first subdivided into a coarse grid. Each cell of this grid is then subdivided if a curvature constraint is not met. Cracks introduced between adjacent cells of different resolutions are patched using polygons whose boundaries are defined by the intersection of the interface between the cell domains and both of the isosurface patches. This approach is optimized through lookup tables on the possible configurations of the two isosurface patches (they reduce this to 22 possible cases). Since this approach does not check field values within its cells, it too can miss features of the full resolution isosurface. The authors propose (but do not implement) an adaptation of BONO [WVG92] to catch these missing features. The meshes extracted using this approach are approximately 55% smaller than those of the original MC algorithm [LC87].

Shekhar et al. [SFYC96] patch cracks on interfaces between cells of different resolution by forcing higher resolution features to align with their lower resolution neighbors, effectively subsampling the data. This approach is similar to [MS93] in that higher resolution features are aligned with those of their lower resolution neighbors. Ohlberger and Rumpf [OR97] present a similar solution using subsampling. Although subsampling approaches achieve a continuous representation, they can change the topology of the extracted surfaces from that of the fine resolution data.

In the work of Westermann et al. [WKE99] samples at higher resolution are averaged, resulting in smoother extracted surfaces; however, this modifies the underlying dataset. Their work is simplified by using balanced octrees, which greatly reduce the number of cases that need to be considered. Finally, they force the boundaries of lower resolution nodes to match those the higher resolution ones. Kazhdan et al. [KKDH07] generalizes the above approach without requiring the octree to be balanced.

Mello et al. [MVT03] extract an approximate variable-resolution representation of an isosurface from a 3D point cloud by first computing the sign field of the vertices of a nested RSB mesh.

The Adaptive Dual Contouring method proposed in [JLSW02] represents the isosurface as an octree in which the leaves contain the isosurface, and the connectivity between nodes is implicitly determined through the face neighbors in the octree. This method enables a bottom-up simplification process, but does not guarantee that the extracted meshes are conforming. Later approaches [GK03, SJW07] ensure that the extracted surface is manifold, but do not guarantee that it is free of self-intersections. Conversely, the approach of Ju and Udeshi [JU06] ensures that the surface does not intersect, but does not guarantee that it is manifold. Zhang et al. [ZBS03] extend the adaptive dual contouring method for

variable-resolution interval volumes.

Most of these approaches support a (bottom-up) simplification process, rather than the more general (top-down) selective refinement query, i.e., it is not possible to extract variable-resolution meshes of minimal size satisfying an error criterion. Alternatively, a multiresolution representation of the extracted mesh enables the generation of variable-resolution meshes satisfying an application-dependent error criterion at runtime.

3.3.4 Multiresolution representations for extracted meshes

Pascucci and Bajaj [PB00] introduce an interruptible progressive isosurfacing algorithm over RSB hierarchies in 2D and 3D. They introduce a small set of primitives which they use to define all possible updates to the isocontour as vertices are introduced at the midpoint of a simplex's bisection edge. They define a one-to-one correspondence between isosurface modifications and the conforming modifications that generate them.

However, since the modifications are defined in terms of basic mesh operations, it becomes more difficult to define the different intersection cases and to guarantee correctness as the dimension and complexity of the cases increases (e.g. for conforming modifications to interval volume meshes). Also, they do not discuss a data structure for the extracted meshes or how this structure might be queried.

The advantage of such a scheme over offline mesh decimation (such as [GH97]) is that the generated surfaces are guaranteed not to self intersect due to the embedding volumetric grid and can be computed interactively. However, the vertices of these surfaces are more constrained than those created by general mesh decimation algorithms. They prove that for a hierarchy with n nodes, the extraction of an isosurface of size k requires $O(k \log n)$ size and time.

Borgo et al. [BPSC04] extract isosurfaces from nested tetrahedral RSB meshes one depth at a time using a breadth-first traversal, requiring two depths of the DAG to be in memory at once. They use an explicit mapping to pass isosurface vertices from parent tetrahedra to child tetrahedra to reduce redundant computation of isosurface vertices and report a three times savings in extraction time on datasets of sizes 64^3 and 128^3

Lewiner et al. [LVLM04a,LVLM04b,LLVM06] introduce a progressive hierarchical representation for isosurfaces extracted from LEB grids in terms of the *tubular neighborhood* of an isosurface, which includes all cells intersected by the isosurface. Their model defines a total order for the modifications, rather than a partial order, limiting the number of encoded meshes.

Our *isodiamond hierarchy* framework (see Chapter 9 and [WD08b, WD10b]) improves on previous approaches by abstracting the mesh extraction method to treat multiresolution isosurfaces and interval volumes in the same framework. It supports efficient selective refinement queries to extract conforming meshes satisfying an application dependent selection criterion and requires significantly less memory than the multiresolution scalar field model. We also introduce minimal isodiamond hierarchies as a means of reducing the storage and computational costs associated with extracting variable resolution isosurfaces and interval volumes.

3.4 Discussion

In this chapter, we have classified and analyzed approaches for representing nested meshes, with a focus on RSB approaches and on how they have been applied to model multiresolution scalar fields.

We now present a taxonomy of these approaches in Tables 3.1 and 3.2. We first distinguish between simplex-based approaches, which we list in Table 3.1 and diamond-based approaches, which we list in Table 3.2. For the purposes of this taxonomy, we classify methods that utilize the dependency relation among the vertices of a nested RSB meshes as diamond-based approaches.

Within each table, we first classify methods based on the underlying dimension of the scalar field's domain and then by the queries that these models support. Such queries can be run from a coarse base domain in a top-down manner, from the full resolution mesh in a bottom-up manner or incrementally from a previously extracted mesh. We also distinguish between the class of selection criteria supported by the approach: those based on approximation error, isosurface error and view-dependent criteria which depend on an object's distance to the viewpoint.

For simplex-based approaches the precomputed errors can be associated with the simplices or with the split vertices. The former can require significantly more storage, but can return smaller meshes as a result of a query. Similarly, a saturated error metric enables simpler queries but can also increase the size of its resultant meshes. The precomputed approximation error can be based on the approximation error between its current value and that of its subdivided children at the next depth, which we refer to as a *local* error metric. Alternatively, it can be based on the maximum interpolation error over all samples within its domain, which we refer to as a *total* error metric [LRC+02].

Our final classification relates to the optimizations introduced or implemented by the various approaches to enable interactive queries on large datasets. This includes compressed meshes in the form of triangle or tetrahedral strips, view frustum culling and cache-coherent access to subsets of the dataset. Since the underlying data structure in all approaches are simplex-based or diamond-based nested RSB meshes, the optimizations developed for one scheme can usually be applied to the other schemes, but are useful for our taxonomy in distinguishing among the various methods.

Table 3.1: Taxonomy of simplex-based approaches

Approach	Dimension	Extraction	Approximation Error	Distance to Viewpoint	Isosurface	Associated with	Saturated	Hierarchical	Frustum Culling	Frame-frame coherence	Simplex Stripping	Out-of-core	Clustered updates	Incomplete Field	Parallel
Evans et al. [EKT01]	2D	Top-down	✓	✓		Simplex		Global							
Lee et al. [LDS01]	3D	Incremental	✓			Simplex		Global	✓						
Lee et al. [LDS04]	4D	Top-Down	✓			Simplex		Global							
ROAM [DWS+97]	2D	Incremental	✓	✓		Vertex	✓	Local	✓	✓	✓		[Pom00,Lev02]		
Marchesin et al. [MDM04]	3D	Incremental	✓	✓		Vertex	✓	Local	✓	✓					
Gerstner [Ger03a]	2D	Top-down	✓			Vertex	✓	Local		✓				✓	
Gerstner [Ger03b]	2D	Top-down	✓	✓		Vertex	✓	Global	✓						
Gerstner et al. [GP00,GR99,Ger02]	3D	Top-down	✓		✓	Vertex	✓	Local							[GR99]
Pascucci [Pas04]	3D	Top-down	✓	✓		Vertex	✓	Local	✓		✓				✓
BDAM [CGG+03a]	2D	Top-down	✓	✓		Vertex	✓	Local	✓		✓	✓		[CGG+03b]	
Tetrapuzzles [CGG+04]	3D	Top-down	✓	✓		Vertex	✓	Local	✓		✓	✓	✓	✓	✓
Lewiner et al. [LVLM04b]	3D	Top-down			✓	Vertex		Local							
Balmelli et al. [BLV03]	2D	Bottom-up	✓			Vertex		Global							

Table 3.2: Taxonomy of diamond-based approaches

Approach	Dimension	Extraction	Approximation Error	Distance to Viewpoint	Isosurface	Saturated	Hierarchical	Frustum Culling	Frame-frame coherence	Simplex Stripping	Out-of-core	Hierarchical Layout	Clustered updates	Incomplete Field
Lindstrom et al. [LKR+96]	2D	Bottom-Up		✓			Local	✓	✓	✓				
Pajarola [Paj98]	2D	Top-Down and Bottom-Up	✓			✓	Global	✓		✓				
SOAR [LP02]	2D	Top-Down	✓	✓		✓	Local	✓		✓	✓	✓		
Hwa et al. [HDJ05]	2D	Incremental		✓		✓	Local	✓	✓	✓	✓	✓	✓	✓
Yalçın et al. [YWD11]	2D	Bottom-up	✓	✓			Global	✓						
Zhou et al. [ZCK97]	3D	Bottom-up			✓	✓	Local							
Gregorski et al. [GDL+02]	3D	Incremental	✓			✓	Global	✓	✓		✓	✓		
Gregorski et al. [GSDJ04]	3D + Time	Incremental	✓			✓	Global	✓	✓		✓	✓	✓	
Linsen [LGP+04,LPD+04]	3D,4D	Top-Down	✓				Global					✓		
Borgo [BPSC04]	3D	Top-Down	✓	✓		✓	Local							
Weiss [WD08b,WD10b]	3D	Top-Down	✓				Global							
Weiss et al. [WD08c,WD09b]	2D,3D	Top-Down	✓				Global						✓	✓

Chapter 4

Diamond hierarchies of arbitrary dimension

In this chapter, we provide a theoretical foundation for diamonds of arbitrary dimensions through a constructive decomposition of diamonds in terms of two simplicially decomposed hypercubes. This enables us to characterize the distinct *classes* of diamonds, and to derive closed-form equations for the number of simplices, vertices, parents and children of each diamond class. In particular, we prove that d-dimensional diamonds have $O(d!)$ simplices, and thus, when using a simplex-based approach, conforming refinements require factorial time and space. In contrast, since these simplices are generated during the refinement of a diamond's $O(d)$ parents , conforming refinements on a diamond-based approach require only linear time and space.

This chapter is organized as follows. We first introduce the *cross simplex* and *cross complex* in Section 4.1 as a means of generating higher dimensional simplices and simplicial complexes from simplices in lower dimensional affinely independent spaces. We then discuss a family of simplicial decompositions of a hypercube in Section 4.2 related through the *Regular Simplex Bisection (RSB)* operator. This operation has been used in the literature to define a *hierarchy of simplices*, which we review in Section 4.3. We then introduce diamonds and their hierarchical dependency relations in Sections 4.4 and properties of diamonds in Section 4.5. We discuss querying algorithms on a hierarchy of diamonds in Section 4.6 and conclude in Section 4.7 with a discussion on some implications of our decomposition.

4.1   Cross simplex and cross complex

We utilize the *simplicial join* operation [RS72, Lic99] to generate higher-dimensional simplices from a pair of affinely independent simplices and refer to the result as a *cross simplex*. Given an a-simplex σ_a and a b-simplex σ_b in affinely independent subspaces, the cross simplex is the d-simplex $\sigma = \sigma_a \otimes \sigma_b$, defined by the vertices of σ_a and σ_b, where $d = a + b + 1$.

For example, if σ_a is a triangle (2-simplex) defined by 3 vertices and σ_b is a vertex (0-simplex) that is not coplanar with the vertices of σ_a, then the cross simplex $\sigma = \sigma_a \otimes \sigma_b$ is the tetrahedron (3-simplex) defined by the vertices of σ_a and σ_b (see Figure 4.1a).

Given a simplicial i-complex Σ_i and a simplicial j-complex Σ_j whose cells are pairwise affinely independent, we define the *cross complex* $\Sigma_d = \Sigma_i \otimes \Sigma_j$ as the simplicial d-complex whose d-simplices are cross simplices of cells from Σ_i and Σ_j, i.e., $\forall$ cells $\sigma \in \Sigma_d$, $\sigma = \sigma_i \otimes \sigma_j$, where σ_i is a cell of Σ_i and σ_j is a cell of Σ_j (see Figure 4.1b). We note that this operation is defined when all pairs of simplices from the two complexes are affinely independent, but the two complexes themselves do not need to be affinely independent.

(a) Cross simplex
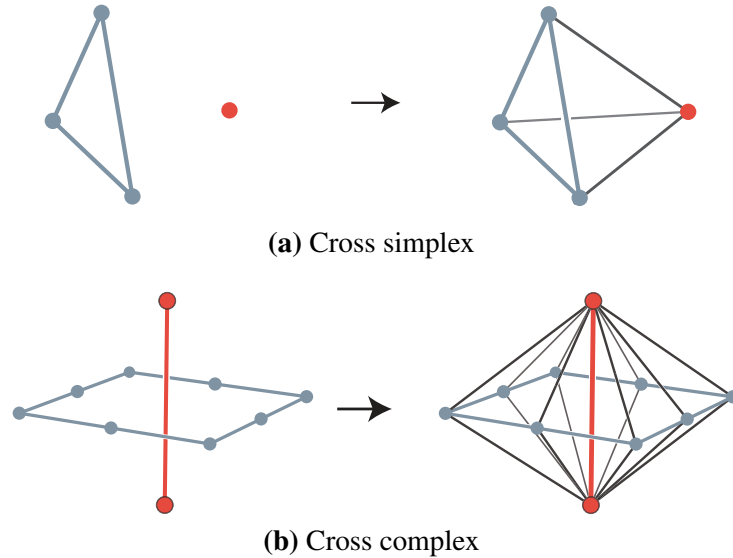
(b) Cross complex

Figure 4.1: (a) The cross simplex of a triangle (blue) and a vertex (red) is a tetrahedron. (b) The cross complex defined by an eight edge complex (blue) and a one edge complex (red) is composed of eight tetrahedra.

(a) 1-cube   **(b)** 2-cube   **(c)** 3-cube
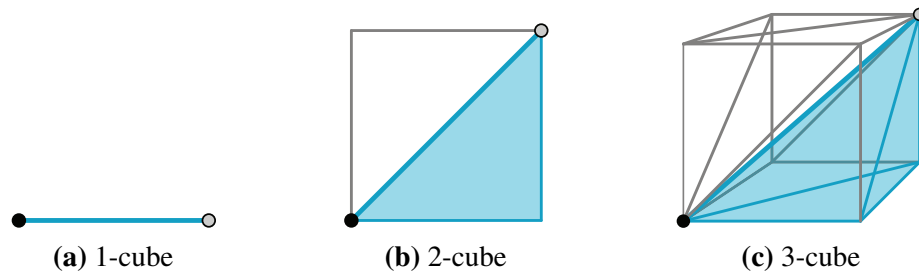
Figure 4.2: Kuhn-subdivided hypercubes in (a) 1D (b) 2D (c) 3D. One of the $d!$ simplices is highlighted in blue. All edges are aligned with the diagonal of an axis-aligned hypercube.

4.2   Simplicial decomposition of hypercubes

We are often interested in generating simplicial complexes that cover a hypercubic domain. To this aim, we first consider the canonical subdivision of a hypercube into $d!$ simplices along a diagonal. This decomposition was initially proposed by Freudenthal [Fre42] and was popularized by Kuhn [Kuh60] in the context of fixed point calculations. We next consider a family of nested decompositions, which we call the Maubach complexes, generated by successively applying regular simplex bisection to the top simplices of a Kuhn-subdivided hypercube.

4.2.1   Kuhn subdivisions

The Kuhn-subdivision of a d-dimensional cube h, which we denote as $\mathcal{K}(h)$, is a simplicial complex [AG79] defined by the $d!$ top simplices, sharing a common diagonal of h.
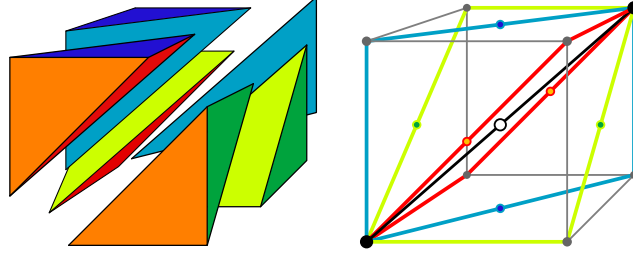
Figure 4.3: Decomposition of a 3-cube into 3! = 6 simplices.

Figure 4.2 illustrates Kuhn-subdivided d-cubes for $d \in \{1, 2, 3\}$, and highlights one of the $d!$ simplices.

Assume, without loss of generality, that a unit d-cube h is embedded in a subspace $[0, 1]^d$ of $\mathbb{R}^n$. Let 0^d and 1^d denote a pair of opposite vertices forming a diagonal $\psi = (0^d, 1^d)$. Also, let e_0 denote 0^d and e_i the ith unit vector in $\mathbb{R}^d$, e.g. $e_1 = (1, 0, 0, \ldots)$, $e_2 = (0, 1, 0, 0, \ldots)$.

We refer to the d-simplex with vertices

$$v_i = \sum_{0 \leq j \leq i} e_j$$

as the *base simplex*, which we denote as S_0. For example, when $d = 3$, S_0 has vertices $(0, 0, 0)$, $(1, 0, 0)$, $(1, 1, 0)$ and $(1, 1, 1)$. The base simplex for $d \in \{1, 2, 3\}$ is highlighted in blue in Figure 4.2.

Let π be a permutation of the integers $\{0, 1, \ldots, d-1\}$ and let $\mathbf{v}' = \pi\mathbf{v}$ indicate the application of permutation π to the coordinates of vertex $\mathbf{v}$. For example, if $\mathbf{v} = (1, \frac{1}{2}, 0)$ and $\pi = \{2, 0, 1\}$, then $\mathbf{v}' = \pi\mathbf{v} = (0, 1, \frac{1}{2})$. Finally, let $\pi\sigma$ denote the application of permutation π to each vertex of simplex σ.

Then, the Kuhn-subdivision of hypercube h, which we denote as $\mathcal{K}(h)$, is defined by the $d!$ simplices of order d obtained by mapping each distinct d-permutation π onto the vertices of the base simplex S_0. Formally,

$$\mathcal{K}(h) = \{ \pi S_0 \mid \pi \text{ is a permutation of } \{0, 1, \ldots, d-1\}\}. \tag{4.1}$$

Figure 4.3 illustrates the 3! = 6 tetrahedra of a Kuhn-subdivided 3-cube.

Since coordinate permutations do not modify 0^d or 1^d, every d-simplex in $\mathcal{K}(h)$ contains diagonal ψ of h. Also, the ith vertex v_i of any cell $\sigma \in \mathcal{K}(h)$ contains $(d - i + 1)$ coordinates of value 0 and i coordinates of value 1. Thus, the edge $(0^d, v_i)$ of σ is a diagonal of an i-face of h, and edge $(v_i, 1^d)$ is a diagonal of a $(d - i)$-face of h. Kuhn subdivisions can be generalized to any d-cube h' with diagonal $\psi' = (v_1, v_2)$ by an affine mapping from the vertices of ψ' to $(0^d, 1^d)$.

An interesting property of Kuhn subdivisions, which will be of use later and which we prove now, is that it provides a Kuhn-subdivision to all faces of the initial hypercube.

Theorem 4.2.1. *Let $\mathcal{K}(h)$ be the simplicial decomposition of a d-cube h, and h_i an i-face of h. Then $\mathcal{K}(h_i) = h_i \cap \mathcal{K}(h)$ is an i-dimensional Kuhn subdivision of the domain of h_i.*

43

Proof. If $i = 0$ then $\mathcal{K}(h_i)$ is trivially a Kuhn subdivision. Assume, without loss of generality, that h is a unit d-cube with diagonal $\psi = (0^d, 1^d)$. We show that the $(d-1)$-faces of h are Kuhn-subdivided. Since d was arbitrary, the proof for the remaining i-faces follows by induction.

Consider the simplicial $(d-1)$-complex Σ obtained by removing vertex 1^d from every simplex $\sigma \in \mathcal{K}(h)$. All cells of Σ are defined by d vertices and are thus $(d-1)$-simplices. In fact, since ψ was the only diagonal of h, and all vertices of $\mathcal{K}(h)$ lie on its boundary, we can decompose Σ into d subcomplexes, each containing simplices within a $(d-1)$-dimensional axis aligned hyperplane of $\mathbb{R}^d$. In the nth such subcomplex $\Sigma_n \subset \Sigma$ this hyperplane can be defined by the equation $x_n = 0$. Thus, the d vertices of a cell $\sigma \in \Sigma_n$ are of the form

$$v'_k = \pi(\sum_{j=0}^{n-1} e_j + \sum_{j=n+1}^{i} e_j). \tag{4.2}$$

By projecting Σ_n onto the $(d-1)$-dimensional subspace of $\mathbb{R}^d$ that excludes coordinate x_n, we obtain the $(d-1)!$ cells of a Kuhn subdivided $(d-1)$-cube (compare Equation 4.2 to Equation 4.1).

Similarly, the simplicial complex defined by removing vertex 0^d gives us Kuhn subdivisions for the d remaining $(d-1)$-faces of h, where each hyperplane is of the form $x_n = 1$. $\qquad\square$

4.2.2 Maubach's typographical bisection scheme

Recall from Section 2.2.2 that a d-simplex σ is bisected along one of its edges $\mathbf{e}$, by inserting a new vertex $\mathbf{v}_m$ at the midpoint of $\mathbf{e}$ and bisecting σ along the hyperplane defined by $\mathbf{v}_m$ and the $(d-1)$ vertices of σ that are not incident to $\mathbf{e}$ (see Figure 2.6 for examples in 2D and 3D). This creates two new d-simplices, covering the same domain as σ, each containing vertex $\mathbf{v}_m$ and one (but not both) of the endpoints of $\mathbf{e}$.

Maubach's bisection scheme [Mau95] specifies the *bisection edge* for any top simplex σ in an initial simplicial d-complex Σ or generated by repeated application of a typographical bisection rule to the cells of Σ. It depends only on the ordering of the vertices of σ, and on the subdivision depth ℓ_σ of σ, which is initialized to zero for any cell in the original complex Σ. Given a d-simplex

$$\sigma = (\mathbf{v}_0, v_1, \ldots, v_{k-1}, \mathbf{v}_k, v_{k+1}, \ldots, v_d),$$

where $k = d - (\ell_\sigma \bmod d)$, the bisection edge is defined by vertices $\mathbf{v}_0$ and $\mathbf{v}_k$, and its midpoint is $\mathbf{v}_m = (\mathbf{v}_0 + \mathbf{v}_k)/2$. The two d-simplices generated by the bisection rule have vertices

$$\sigma_0 = (\ \mathbf{v}_0, v_1, \ldots, v_{k-1}, \ \mathbf{v}_m, \ v_{k+1}, \ldots, v_d)$$
$$\sigma_k = (\ \ v_1, v_2, \ldots, \mathbf{v}_k, \ \ \mathbf{v}_m, \ v_{k+1}, \ldots, v_d),$$

and the depth of these simplices is incremented, e.g.

$$\ell_{\sigma_0} = \ell_{\sigma_k} = (\ell_\sigma + 1).$$

| 0-triangle | 1-triangle | 0-tetrahedron | 1-tetrahedron | 2-tetrahedron |

(a) 2D RSB classes　　　　　　　　　**(b)** 3D RSB classes
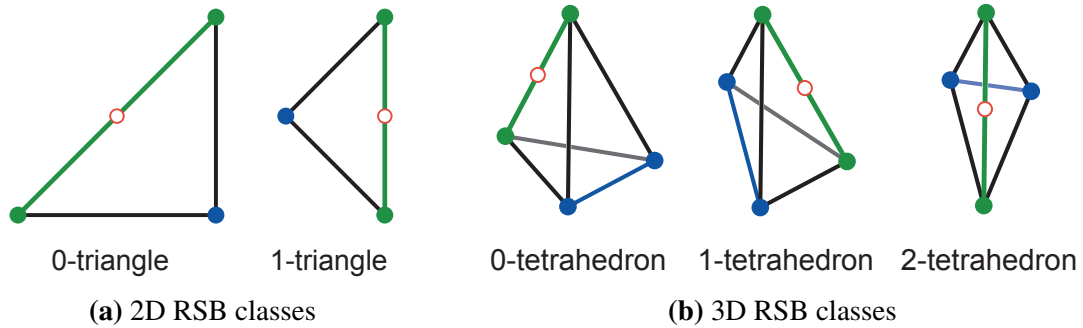
Figure 4.4: The d-dimensional RSB scheme has d similarity *classes* of top simplices. In 2D, there are two classes of triangles (a) while in 3D, there are three classes of tetrahedra (b). The bisection edge (green) of a class i simplex is aligned with the diagonal of an axis aligned $(d-i)$-cube.

Maubach proves that when his bisection scheme is applied to a Kuhn-subdivided d-cube h whose simplex vertices are ordered as in Section 4.2.1, the generated d-simplices belong to at most d similarity classes [Mau95]. Recall that simplices are similar if there is an affine mapping consisting of only uniform scaling, reflection, rotation and translation between them. Since coordinate permutations are rigid mappings, the $d!$ cells in $\mathcal{K}(h)$ belong to the same similarity class. Furthermore, all cells at depth $(\ell \mod d)$ belong to the same similarity class.

We note that, although the term *Longest Edge Bisection (LEB)* [Riv84, Riv91] has been applied to this family of decompositions, it is no longer applicable when the dimension d is greater than three. To see this, consider the d-dimensional unit cube h with edge length 1 and diagonal length $\sqrt{d}$, where $d > 3$. All simplices in $\mathcal{K}(h)$ contain the diagonal of the cube as well as at least one edge of h (see Figure 4.2). After bisecting a simplex σ of $\mathcal{K}(h)$, the resultant simplices contain an edge e' of length $\sqrt{d}/2$ as well as some edges of the original cube h. Neither edge e' nor edges of h are bisected in the $(d-2)$ bisections that follow. In the dth bisection step, the bisection edge is an edge of h (of length 1), but e' has length greater than or equal to 1 since $\sqrt{d}/2 = 1$ when $d = 4$ and is greater than 1 when $d > 4$.

We therefore use the term *Regular Simplex Bisection (RSB)* to describe the simplex bisection operation applied to a Kuhn-subdivided hypercubic domain according to Maubach's scheme. We refer to any simplex generated by successive RSB operations as an *RSB simplex*, and to nested meshes consisting of RSB simplices as *RSB meshes*.

We denote the cells of $\mathcal{K}(h)$ as *class*-0 simplices, and to an RSB simplex σ of order d as a *class-i* simplex if $i = (\ell_\sigma \mod d)$. Observe that the bisection edge $\psi = (\mathbf{v}_0, \mathbf{v}_{d-i})$ of a class-i simplex is aligned with the diagonal of a $(d-i)$-cube. Figure 4.4 illustrates the two classes of RSB triangles and the three classes of RSB tetrahedra generated using the RSB scheme.

Consider the family of simplicial d-complexes $\mathcal{M}_i(h)$, which we refer to as the *Maubach complexes*, generated through successive bisections to the cells of $\mathcal{K}(h)$, where i denotes the depth of the d-simplices in $\mathcal{M}_i(h)$ and $\mathcal{M}_0(h)$ contains the $d!$ cells of $\mathcal{K}(h)$. Since each d-simplex in $\mathcal{M}_0(h)$ is replaced by two d-simplices in $\mathcal{M}_1(h)$, $\mathcal{M}_1(h)$ contains

$2 \cdot d!$ cells, and in general, $\mathcal{M}_i(h)$ contains $2^i \cdot d!$ cells. Figures 4.5 and 4.6 illustrates the first few Maubach complexes for $d = 2$ and $d = 3$, respectively.

(a) $\mathcal{M}_0(h) = \mathcal{K}(h)$ **(b)** $\mathcal{M}_1(h)$ **(c)** $\mathcal{M}_2(h) = \mathcal{F}(h)$
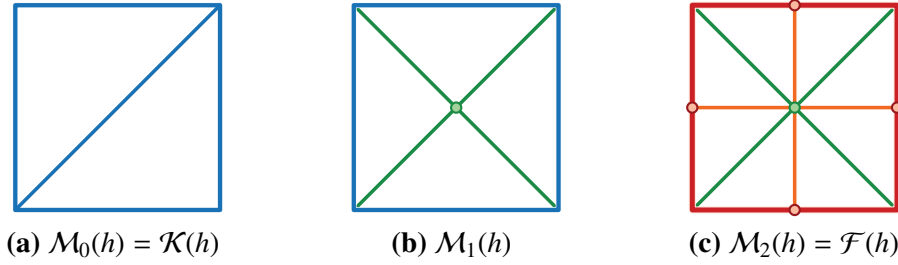
Figure 4.5: Three consecutive Maubach complexes $\mathcal{M}_i(h)$ in 2D. (a) $\mathcal{M}_0(h)$ is equivalent to $\mathcal{K}(h)$ and has $2! = 2$ triangles. (b) $\mathcal{M}_1(h)$ has $2 \cdot 2! = 4$ triangles. (c) $\mathcal{M}_2(h)$ is equivalent to $\mathcal{F}(h)$ and has $2^2 \cdot 2! = (2 \cdot 2)!! = 8$ triangles.

(a) $\mathcal{M}_0(h) = \mathcal{K}(h)$ **(b)** $\mathcal{M}_1(h)$ **(c)** $\mathcal{M}_2(h)$ **(d)** $\mathcal{M}_3(h) = \mathcal{F}(h)$
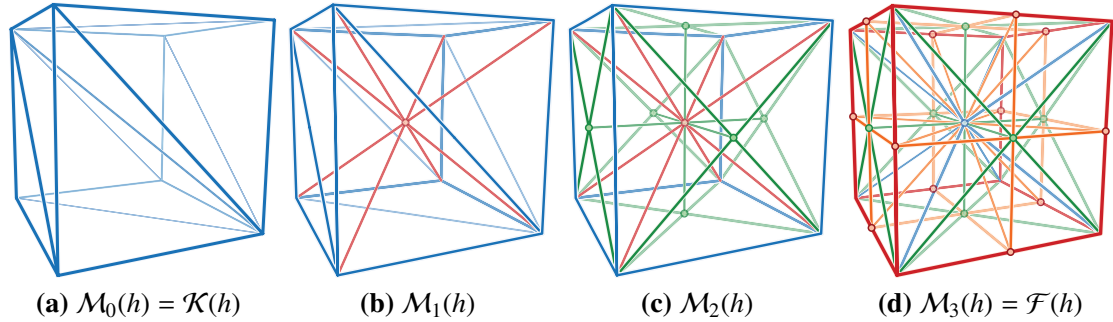
Figure 4.6: Four consecutive Maubach complexes $\mathcal{M}_i(h)$ in 3D. (a) $\mathcal{M}_0(h)$ is equivalent to $\mathcal{K}(h)$ and has $3! = 6$ tetrahedra. (b) $\mathcal{M}_1(h)$ has $2 \cdot 3! = 12$ tetrahedra. (c) $\mathcal{M}_2(h)$ has $2^2 \cdot 3! = 24$ tetrahedra. (c) $\mathcal{M}_3(h)$ is equivalent to $\mathcal{F}(h)$ and has $2^3 \cdot 3! = (2 \cdot 3)!! = 48$ tetrahedra.

4.2.3   Fully subdivided hypercubes

Let us analyze the properties of the dth Maubach complex, $\mathcal{M}_d(h)$, which we refer to as a *fully subdivided* hypercube, and denote as $\mathcal{F}(h)$. $\mathcal{F}(h)$ is a simplicial d-complex defined by the $2^d \cdot d!$ cells resulting from d bisections of the cells in $\mathcal{K}(h)$. Each such cell belongs to class-0 and is a factor of two smaller than those of $\mathcal{K}(h)$ [Moo92].

We simplify the notation by observing that $2^d \cdot d!$ can be defined in terms of the *double factorial* function [Mes48] as

$$2^d \cdot d! = (2d)!!$$

where the double factorial $n!!$ is equal to 1 if $n \in \{0, 1\}$ and $n \cdot (n - 2)!!$ otherwise. The values of $(2d)!!$ for $d \in [1, 2, 3, 4]$ are $[2, 8, 48, 384]$.*

---

*We provide a more thorough treatment of the double factorial function and its properties in Appendix A.

Let h be a d-cube with midpoint $\mathbf{v}_c$, $\mathcal{K}(h)$ the Kuhn subdivision of h along diagonal ψ, and $\mathcal{F}(h)$ its corresponding fully subdivided hypercube. We first show that each cell of $\mathcal{F}(h)$ has a bisection edge defined by a vertex of h and the midpoint $\mathbf{v}_c$ of h.

Lemma 4.2.1. *For all cells $\sigma \in \mathcal{F}(h)$, $\mathbf{v}_c$ is a vertex of σ. Furthermore, the bisection edge of σ is defined by $\mathbf{v}_c$ and one of the 2^d vertices of h.*

Proof. This follows from the generation of $\mathcal{F}(h)$ in terms of the Maubach complexes $\mathcal{M}_i(h)$ starting with $\mathcal{M}_0(h) = \mathcal{K}(h)$. After the first application of the bisection rule to the cells of $\mathcal{M}_0(h)$, all cells σ of $\mathcal{M}_1(h)$ have the midpoint $\mathbf{v}_c$ of ψ as their d^{th} vertex. Since none of the next $(d-1)$ bisections modify the dth vertex, all cells of $\mathcal{M}_d(h) = \mathcal{F}(h)$ contain $\mathbf{v}_c$.

Since all cells in $\mathcal{F}(h)$ are class-0, the bisection edge is determined by the first and last vertices of σ. As described above, the last vertex of σ is $\mathbf{v}_c$. Since σ is a class-0 simplex, its bisection edge must be the diagonal of a d-cube. The only edges of $\mathcal{F}(h)$ that satisfy this constraint are those between $\mathbf{v}_c$ and a vertex of h. $\qquad\square$

Recall that for a d-cube h, $\mathcal{K}(h)$ contains $d!$ class-0 simplices. An alternate interpretation of $\mathcal{F}(h)$ is as a collection of Kuhn-subdivided subcubes covering the domain of h and centered at the midpoint of h.

Corollary 4.2.2. $\mathcal{F}(h)$ *consists of 2^d Kuhn-subdivided d-cubes covering h and with side length half that of h. Thus, each of the 2^d subcubes contributes $d!$ cells to $\mathcal{F}(h)$ for a total of $2^d d! = (2d)!!$ cells.*

Similarly to the Kuhn subdivision, all i-faces of a fully subdivided d-cube are fully subdivided i-cubes.

Theorem 4.2.3. *Each i-face h_i of a fully subdivided d-cube $\mathcal{F}(h)$ is a fully subdivided i-cube $\mathcal{F}(h_i)$.*

Proof. Consider the simplicial $(d-1)$-complex Σ obtained by removing the vertex $\mathbf{v}_c$ at the midpoint of $\mathcal{F}(h)$ from each cell $\sigma \in \mathcal{F}(h)$. Since each j-face h_j of a Kuhn-subdivided cube is a Kuhn-subdivided j-cube, the removal of $\mathbf{v}_c$ from a Kuhn-subdivided subcube within $\mathcal{F}(h)$ adds the $(d-1)!$ cells of a Kuhn-subdivided $(d-1)$-cube to each of the d facets of h on which it is incident (see the proof of Theorem 4.2.1 for details). Since there are 2^{d-1} subcubes incident with each facet h_i of h, h_i contains $2^{d-1}(d-1)! = (2(d-1))!!$ cells. Since these $(d-1)$-simplices are from a Kuhn-subdivided $(d-1)$-simplicial complex, they are all class-0 cells of dimension $d-1$. Further, all cells of $\mathcal{F}(h_i)$ contain the midpoint of h_i coinciding with the midpoint of the 2^{d-1} subcubes adjacent to h_i. $\qquad\square$

This enables us to compute the number of vertices in $\mathcal{F}(h)$.

Corollary 4.2.4. *Since $\mathcal{F}(h)$ contains the midpoint of all i-faces of h, $\mathcal{F}(h)$ contains $\sum \binom{d}{i} 2^{d-i} = 3^d$ vertices[†]. Each vertex coincides with a face of h.*

Using the above properties, we can define a fully subdivided d-cube as a cross-complex of its boundary faces and $\mathbf{v}_c$.

[†]This summation is a special case of the binomial theorem, see Appendix C.

Corollary 4.2.5. *Let h_i denote one of the $2 \cdot d$ facets of h. A fully subdivided d-cube $\mathcal{F}(h)$ with midpoint $\boldsymbol{v}_c$ can be decomposed as the cross-complex of the $(d-1)$-simplices from each fully subdivided facet $\mathcal{F}(h_i)$ with the singleton simplicial complex $\{\boldsymbol{v}_c\}$. E.g.*

$$\mathcal{F}(h) = \{\bigcup \mathcal{F}(h_i) \otimes \{\boldsymbol{v}_c\} \mid h_i \text{ is a facet of } h\}.$$

Corollary 4.2.5 motivates the double factorial notation. Each of the $2 \cdot d$ facets h_i of h contributes the $(2(d-1))!!$ cells generated by $\mathcal{F}(h_i) \otimes \boldsymbol{v}_c$, so $\mathcal{F}(h)$ is composed of $2d \cdot (2(d-1))!! = (2d)!!$ cells.

We are also interested in the simplicial complex defined by the simplices on the $2 \cdot d$ facets h_i on the boundary of $\mathcal{F}(h)$, which we call a *fully-subdivided d-cube boundary* and denote as $\mathcal{B}_F(h)$. Thus,

$$\mathcal{B}_F(h) = \{\bigcup \mathcal{F}(h_i) \mid h_i \text{ is a facet of } h\}.$$

is the simplicial $(d-1)$-complex defined by $(2d)!!$ cells of dimension $(d-1)$. Each such cell corresponds to a cell of $\mathcal{F}(h)$ where the vertex at the center of $\mathcal{F}(h)$ has been removed. Figure 4.7 shows examples of fully subdivided i-cube boundaries for $i = 1, 2, 3$, and highlights (in red) the midpoint of each facet of $\mathcal{B}_F(h)$.

(a) 1-cube   **(b)** 2-cube   **(c)** 3-cube
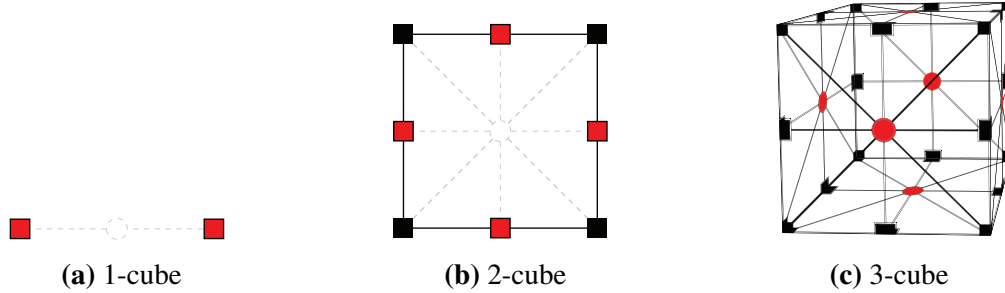
Figure 4.7: Fully subdivided i-cube boundary $\mathcal{B}_F$ for (a) 1-cube (b) 2-cube and (c) 3-cube, containing 2 vertices, 8 edges and 48 triangles, respectively.

4.3 A hierarchy of RSB simplices

Consider a d-dimensional hypercubic domain h initially decomposed into $d!$ cells as $\mathcal{K}(h)$. A hierarchical *containment relationship* exists between cells in consecutive Maubach complexes. The two d-simplices σ_1 and σ_2 generated through a bisection operation on simplex σ are the *children* of σ, and conversely, σ is the *parent* of σ_1 and σ_2.

This nested relationship can be captured as a *simplex tree*, a binary tree whose root is a d-simplex from $\mathcal{K}(h)$. Furthermore, the entire nested simplicial complex can be represented as a forest of $d!$ simplex trees whose roots are the class-0 cells of $\mathcal{K}(h)$. We call this forest of simplex trees a *hierarchy of RSB simplices*, which we will also refer to as a *hierarchy of simplices* [AM07] in general, and as a *hierarchy of (right) triangles* [EKT01,

(a) Hierarchy of RSB triangles   **(b)** Simplex containment relation
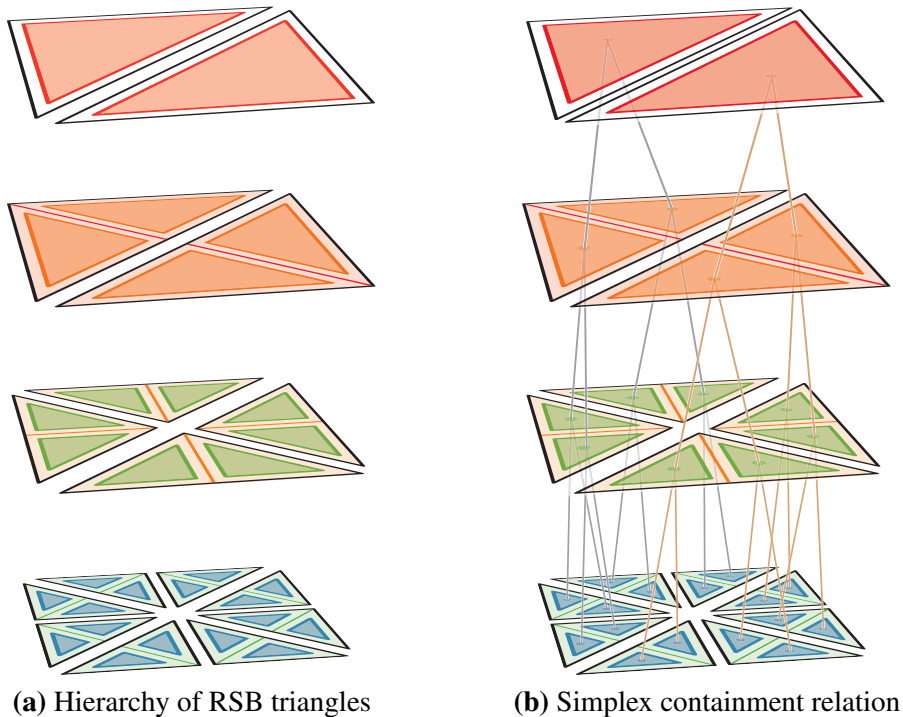
Figure 4.8: A hierarchy of RSB triangles. (a) The triangles at four depths (two levels) of the hierarchy. (b) The simplex containment relation defines a forest of binary trees.

DWS⁺97,Ger03a] a *hierarchy of tetrahedra* [Heb94,ZCK97,GR99,LDS01] or a *hierarchy of pentatopes* [LDS04] in 2D, 3D and 4D, respectively.

The *depth* of a simplex is defined recursively as 0 for the bintree roots, and one greater than the depth of its parent otherwise. All root simplices belong to $\mathcal{K}(h)$, so they are all class-0 simplices. Since RSB is used to generate the simplices at successive depths, all simplices at the same bintree depth belong to the same class of simplices. Furthermore, since there are d classes of simplices in a hierarchy of simplices, and the classes repeat cyclically, the *class* of a simplex σ at depth m is $(m \mod d)$. The simplices at d successive depths define a *level* of the hierarchy. The *level* of a simplex at depth m is then $\lfloor m/d \rfloor$.

The bisection edge of σ is the diagonal ψ of a $(d-i)$-cube, and the vertex introduced during the bisection of σ coincides with the midpoint of ψ.

Figure 4.8 illustrates the hierarchical relationship between the triangles in a hierarchy of triangles at four successive depths (two levels). Note that the hierarchy of triangles contains two binary trees, and that each triangle has two children that cover its domain, thus forming a nested decomposition of the square domain.

The fundamental operation performed on a hierarchy of simplices is the extraction of adaptive simplicial complexes via a *selective refinement* query (see Section 2.4.1). Let σ be a cell of a simplex tree T and σ_1 and σ_2 its children. Since σ_1 and σ_2 cover the same domain as σ, the hierarchical relationship between cells of T defines a nested simplicial mesh. Thus, since $\mathcal{K}(h)$ is a simplicial decomposition of the domain, repeated application of the simplex bisection operation to cells in the forest always provides a non-overlapping

simplicial decomposition of the domain.

However, due to the local nature of an individual RSB operation, it does not, in general, generate valid simplicial complexes. Consider the faces of a cell σ in a simplicial complex Σ generated according to the regular simplex bisection rule. Since Σ is a simplicial complex, all faces adjacent to those of σ intersect only at common faces. However, after σ is bisected along edge ψ, faces that were previously incident to ψ are no longer conforming (see Figure 2.1b).

Thus, the bisection rule requires additional constraints to ensure the generation of valid simplicial complexes. Namely, (a) the depth of all cells incident to bisection edge ψ of a cell σ must be equal to that of σ before the bisection; and (b) all such cells must be bisected concurrently with σ. To satisfy this constraint, we must first find the set of neighbors of cell σ along bisection edge ψ. The so-called *neighbor-finding* operation, finds all d-simplices adjacent to σ along its bisection edge. Neighbor finding can be accomplished by storing pointers to each of the $d + 1$ neighboring cells [Mau95] or symbolically by manipulating *location codes* that uniquely identify each cell in the forest [Heb94,Mau96,EKT01,LDS01,LDS04,AM07]. Symbolic neighbor-finding enables a pointerless representation for cells in the forest, thus enabling each neighbor-finding operation to be carried out in $O(1)$ time. However, since each neighbor must be found, this operation must be performed $O\left(\left|Neighbors(\sigma)\right|\right)$ times.

4.4 A hierarchy of diamonds

We have seen that conforming updates to a simplicial complex generated using the RSB scheme are related to the set of RSB simplices surrounding a common bisection edge. An alternative model can be defined by clustering all d-simplices sharing a common bisection edge into a new primitive, called a *diamond* [DWS+97,GP00,GDL+02,Pas02] and considering the hierarchical relationships between diamonds rather than those between RSB simplices.

An RSB *diamond* is the set of all d-simplices in a d-dimensional hierarchy of RSB simplices with a common bisection edge, called the *spine* of the diamond. Since all d-simplices within a diamond are congruent, there are d similarity *classes* of diamonds in correspondence to the d similarity classes of RSB simplices. We refer to a diamond whose d-simplices belong to class-i as an *i-diamond* and note that its spine is aligned with the diagonal of an axis-aligned $(d - i)$-cube. Figure 4.9 illustrates the two classes of two-dimensional diamonds and the three classes of three-dimensional diamonds.

4.4.1 Diamond subdivision

A diamond δ is subdivided by bisecting all of its d-simplices using the RSB scheme. Thus, subdivision doubles the number of cells within δ and we denote its corresponding *subdivided diamond* as δ_s (see Figure 4.10 for examples in 2D and 3D).

Diamond subdivision is an instance of *stellar subdivision* [Ale30,New31,Lic99] and is a conforming refinement in arbitrary dimensions. An important property of diamond subdivision is that all changes occur within the interior of the subdividing diamond δ. Consequently, the faces on the boundary of δ are unaffected by its subdivision. The local

(a) 2D Diamond Classes   **(b)** 3D Diamond Classes
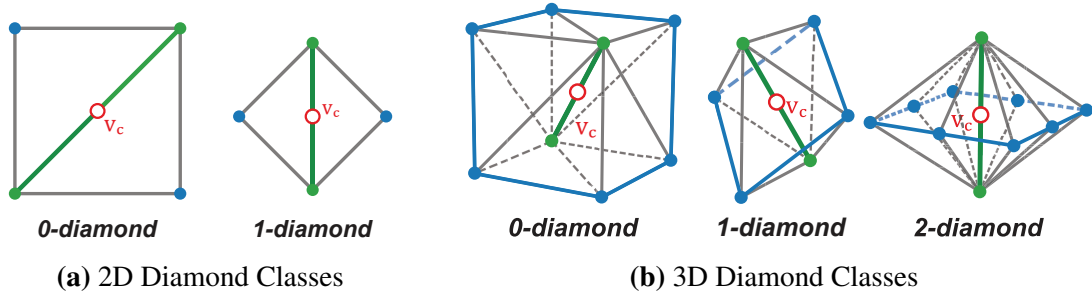
Figure 4.9: The two classes of diamonds in 2D (a) and the three classes of diamonds in 3D (b). The *spine* of an i-diamond (green edge) is aligned with the diagonal of a $(d - i)$-cube.

(a) 2D   **(b)** 3D
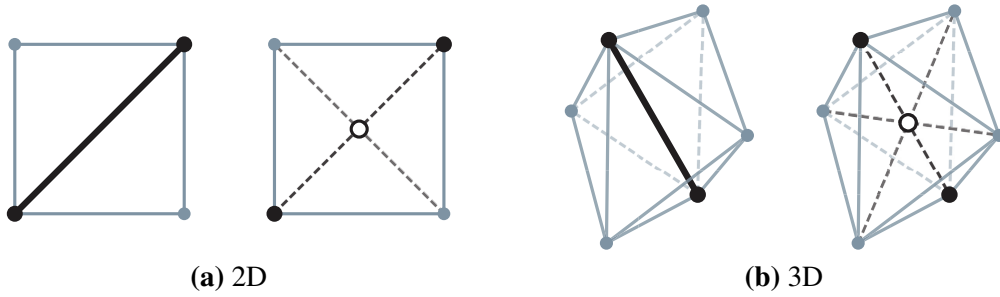
Figure 4.10: Diamond subdivision in 2D (a) and 3D (b).

effect of the subdivision of a diamond δ is to (a) remove its spine (b) add a vertex $\mathbf{v}_c$ at the midpoint of its spine, which we refer to as its *central vertex* and (c) add edges from $\mathbf{v}_c$ to each vertex $\mathbf{v}$ of δ, which we refer to as its *subdivision edges*. A diamond can be uniquely identified by its spine, or alternatively, by its central vertex, the midpoint of the spine.

4.4.2 Diamond dependency relation

The hierarchical relationship among RSB simplices defines a *direct dependency relation* on the diamonds. In contrast with the containment relationship among the simplices within the hierarchy, which can be represented as a forest of binary trees, the diamond dependency relationship defines a partial order on the diamonds, which can be described using a *Directed Acyclic Graph (DAG)*.

A *hierarchy of diamonds* is a multiresolution model (see Section 2.4), which we denote as Δ. When defined over a cubic domain Ω,

- the *base mesh* of Δ is defined by the 0-diamond decomposing Ω and contains all simplices of $\mathcal{K}(\Omega)$;

- a *modification* in Δ is a pair defined by an (unsubdivided) diamond δ and by the subdivided diamond δ_s associated with it, and is denoted as $u = (\delta, \delta_s)$ (see Figure 4.11); and

- the dependency relation is defined as in Section 2.4 and thus Δ is described by a dependency graph which is a DAG.

Let $u_p = (\delta_p, \delta_{p_s})$ be a modification that directly precedes $u_c = (\delta_c, \delta_{c_s})$ in the dependency graph of Δ. Then, there is at least one d-simplex in the subdivided diamond δ_{p_s} that is also in the unsubdivided diamond δ_c. We call δ_p a *parent* diamond of δ_c and, conversely, δ_c a *child* diamond of δ_p in the hierarchy. Figure 4.11 shows the direct dependency relation between two modifications $u_p = (\delta_p, \delta_{p_s})$ and $u_c = (\delta_c, \delta_{c_s})$ in 2D. u_p is a parent of u_c since δ_{p_s} and δ_c have a triangle in common (light blue).

(a) Parent $u_p = (\delta_p, \delta_{p_s})$ **(b)** Child $u_c = (\delta_c, \delta_{c_s})$
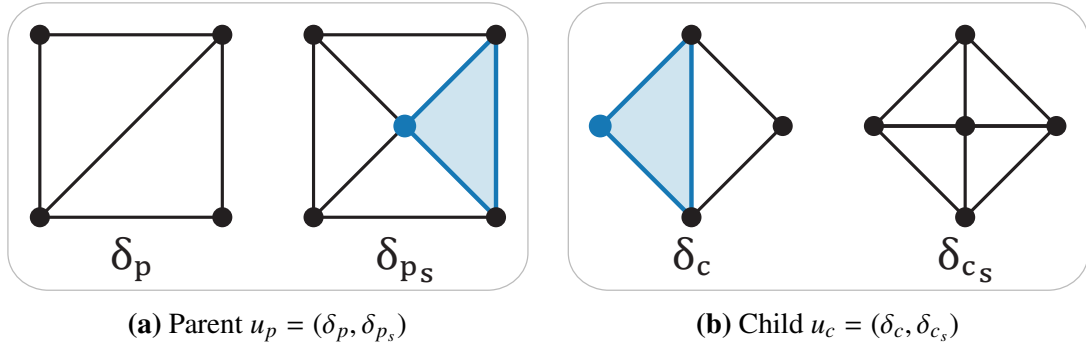
Figure 4.11: Modification u_p is a parent of modification u_c in Δ since subdivided diamond δ_{p_s} and unsubdivided diamond δ_c have a triangle in common (light blue).

For a diamond δ, we denote the set of its children diamonds as *Children*(δ), and the set of its parent diamonds as *Parents*(δ). Although each parent diamond $\delta_p \in$ *Parents*(δ) may only partially cover the domain of δ, the set, *Parents*(δ), collectively covers δ's domain (and similarly for *Children*(δ)). Figure 4.12 illustrates the diamonds in a 2D hierarchy of diamonds at four successive depths (two levels). Each diamond is identified with its spine (a colored edges) and with its central vertex (filled circles at the midpoint of its spine).

4.4.3 Parent-child duets

By definition, a subdivided parent diamond δ_{p_s} and its (unsubdivided) child δ_c always have at least one cell in common. We refer to the set of simplices shared by a subdivided parent and one of its children as a *parent-child duet* or, simply, a *duet*. Duets define the unique contribution of simplices from a parent diamond to one of its children, and thus, they are in one-to-one correspondence with the arcs of the dependency graph of Δ (see Figure 4.13). Observe that a duet between subdivided parent δ_{p_s} and child δ_c always contains the central vertex of δ_{p_s} as well as the spine of δ_c. In 2D, each duet consists of a single triangle: the central vertex of the parent and the spine of the child. In 3D, we observe that δ_{p_s} and δ_c consists of a pair of face-adjacent tetrahedra whose shared face is defined by the spine of δ_c and by the central vertex of δ_{p_s}.

(a) A hierarchy of diamonds   **(b)** Diamond dependency relation
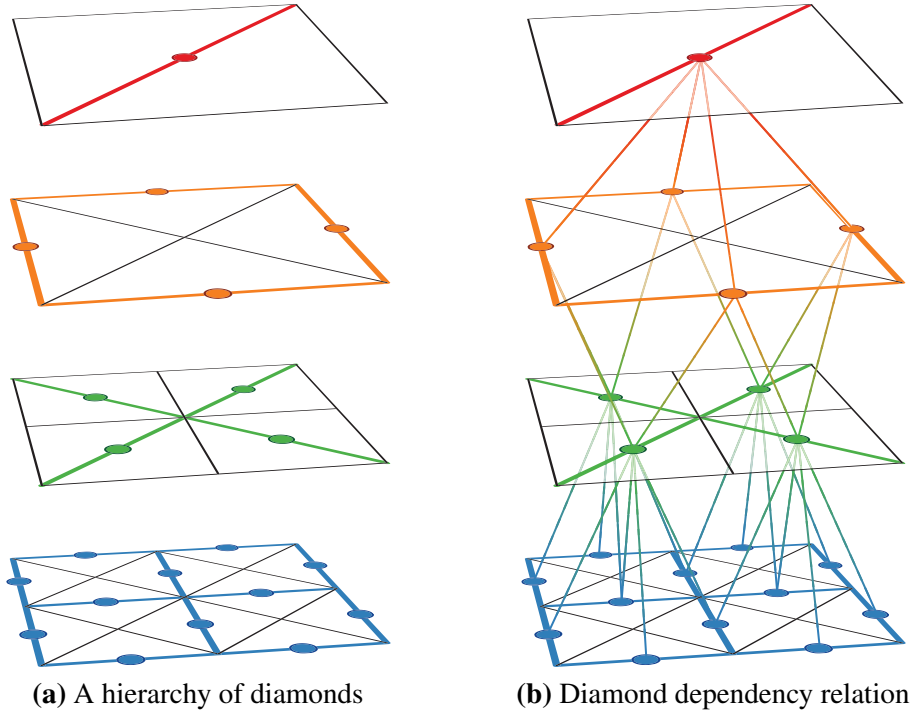
Figure 4.12: A hierarchy of diamonds in 2D. (a) Diamonds at four depths (two levels) of the hierarchy. Each diamond is uniquely identified by its spine (colored edge) and its central vertex (filled circle). (b) The diamond dependency relation defines a partial order on the diamonds and can be encoded as a rooted DAG.

4.5 Properties of a hierarchy of diamonds

We now focus on the combinatorial structure of an arbitrary i-diamond δ of dimension d. This leads to the derivation of closed-form equations for the number of simplices and vertices in δ as well as the number and location of its parents and children.

Theorem 4.5.1. *An i-diamond δ in dimension d is the cross-complex defined by $\mathcal{K}(h_k)$, a Kuhn subdivided $(d - i)$-cube, h_k, and $\mathcal{B}_F(h_i)$, the boundary of a fully subdivided i-cube, h_i, i.e.,*

$$\delta = \mathcal{K}(h_k) \otimes \mathcal{B}_F(h_i)$$

such that h_k and the facets of h_i are in affinely independent subspaces of $\mathbb{R}^d$, and the center of h_k and of h_i coincide.

Proof. Consider the vertices of an arbitrary d-simplex $\sigma \in \delta$

$$\sigma = (\underbrace{\mathbf{v}_0, v_1, \ldots, v_{k-1}, \mathbf{v}_k}_{(d-i+1) \text{ vertices}}, \underbrace{v_{p_1}, v_{p_2}, \ldots, v_{p_j}, \ldots, v_d}_{i \text{ vertices}}),$$

where $k = d - i$. Since δ is defined by its spine $\psi = (\mathbf{v}_0, \mathbf{v}_k)$, which is the diagonal of a $(d - i)$-cube, the vertices in position 0 in all d-simplices of δ are identical, and similarly for the vertices in position $k = d - i$. Furthermore, the midpoint $\mathbf{v}_c = \frac{1}{2}(\mathbf{v}_0 + \mathbf{v}_k)$ of ψ is the
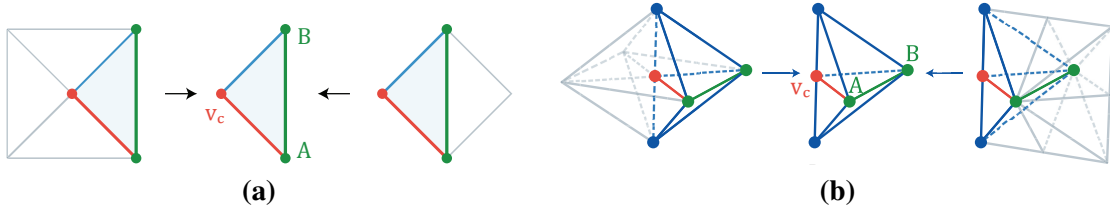
Figure 4.13: Parent-child duets are in one-to-one correspondence with the arcs of the dependency graph and always contain the central vertex $\mathbf{v}_c$ of the parent (red vertex) as well as the spine vertices **A** and **B** of the child diamond (green vertices). (a) 2D duet between a 0-diamond and a 1-diamond (b) 3D duet between a 1-diamond and a 2-diamond.

central vertex of δ and is the vertex that will be inserted in position k for all d-simplices generated during the subdivision of δ.

Due to the use of Maubach's bisection scheme, vertex v_{p_1} at position $(d - i + 1)$ of σ, where $i > 0$, is the center of a $(d - i + 1)$-cube h_p. Also, v_{p_1} is the central vertex of the diamond δ_p whose subdivision generated σ. Similarly, for $j \leq i$, the vertex v_{p_j} at position $(d - i + j)$ of σ is the center of a $(d - i + j)$-cube and v_{p_j} is the central vertex of the level-j ancestor diamond of δ.

The proof is split into two parts. We first show the $(d - i)$-dimensional Kuhn-subdivided component of δ, $\mathcal{K}(h_k)$, whose vertices are in the initial $(d - i + 1)$ positions of any d-simplex $\sigma \in \delta$. Next, we show the fully subdivided i-cube boundary component, $\mathcal{B}_F(h_i)$, whose vertices are in the final i positions of σ. Since σ is a d-simplex, all of its vertices must be in affinely independent subspaces of $\mathbb{R}^d$, and thus σ is a cross-simplex of a $(d - i)$-simplex from $\mathcal{K}(h_k)$ and an $(i - 1)$-simplex from $\mathcal{B}_F(h_i)$.

Kuhn component. Consider the set of d-simplices within δ whose final i vertices are the same, i.e. if σ_a and σ_b are two such d-simplices, then the vertex at position $(k + j)$ of σ_a is equal to the vertex at position $(k + j)$ of σ_b, for $0 < j \leq i$. Since we use Maubach's ordering for the simplices, the subspace of $\mathbb{R}^d$ spanned by these simplices is a $(d - i)$-cube h_k, whose diagonal is ψ. Furthermore, since our hierarchy began with a Kuhn subdivision of h and all i-faces of a Kuhn subdivided d-cube are Kuhn subdivided (Theorem 4.2.1), these simplices comprise a Kuhn subdivision of h_k, i.e. $\mathcal{K}(h_k)$, and there are $(d - i)!$ such simplices.

Fully subdivided component. This proof involves a grid that is *dual* to the one we have been using (i.e. the *primal grid*). A vertex of the dual grid corresponds to the center of a d-cube of the primal grid, and, in general, a j-cube of the dual grid corresponds to a $(d - j)$-cube of the primal grid (see Figure 4.14). Observe that the vertices of this dual grid are offset from those of the primal grid by one half unit in each axis-aligned direction.

Recall that, on the primal grid, the vertex v_{p_j} at position $(d - i + j)$ of σ is the center of a $(d - (i - j))$-cube whose center coincides with the central vertex of a parent of δ. Then, on the dual grid, vertex v_{p_j} of σ is the center of an $(i - j)$-cube, $1 \leq j \leq i$. Note that the central vertex of δ (which is not a vertex of δ until after it subdivides) is the center of an i-cube, h_i on the dual grid.

In the following, consider the collection of d-simplices within δ whose initial $(d - i + 1)$ vertices are the same, i.e. if σ_a and σ_b are two such d-simplices, then the vertex at position j of σ_a is equal to the vertex at position j of σ_b, for $0 \leq j \leq (d - i)$. We can thus project these d-simplices into an $(i - 1)$-dimensional subspace of $\mathbb{R}^d$.

Our claim, which we prove through induction on i, is that these $(i - 1)$-simplices decompose the boundary of a fully subdivided i-cube h_i, e.g. $\mathcal{B}_F(h_i)$. In the base case, $i = 0$, and $\mathcal{B}_F(h_i)$ is empty and is therefore the boundary of a fully subdivided 0-cube.

For the inductive step, assume that in an $(i - 1)$-diamond δ_p, the final $(i - 1)$ vertices of each simplex correspond to the boundary of a fully-subdivided $(i - 1)$-cube $\mathcal{B}_F(h_p)$, whose simplices therefore have dimension $i - 2$. When δ_p is subdivided, its central vertex $\mathbf{v}'$ coinciding with the center of h_p is inserted. In addition, edges are created from $\mathbf{v}'$ to all vertices of δ_p, including the vertices of $\mathcal{B}_F(h_p)$. This increases the dimension of each of $\mathcal{B}_F(h_p)$'s simplices and generates $\mathcal{F}(h_p)$ (recall from Section 4.2.3 that $\mathcal{F}(h_p)$ is defined as the cross-complex of $\mathcal{B}_F(h_p)$ and the vertex at its center). All simplices generated during this subdivision contain vertex $\mathbf{v}'$ in position $d - i$. The final i vertices of each such d-simplex defines an $(i - 1)$-simplex, and together these $(i - 1)$-simplices form the fully subdivided $(i - 1)$-cube $\mathcal{F}(h_p)$.

Now, consider the subset of these simplices that get contributed to an i-diamond δ_c that is a child of δ_p. These are characterized by having the same spine vertices (e.g. $\mathbf{v}_0$ and $\mathbf{v}_k$). Since δ_c is an i-diamond its central vertex is the midpoint of an i-cube h_i in the dual grid. Among the $2 \cdot i$ facets of h_i, one is the fully subdivided $(i - 1)$-cube $\mathcal{F}(h_p)$. By symmetry, each of the other facets are subdivided similarly, and, thus, the boundary of h_i is subdivided as a fully subdivided i-cube, i.e. $\mathcal{B}_F(h_i)$. $\qquad\square$
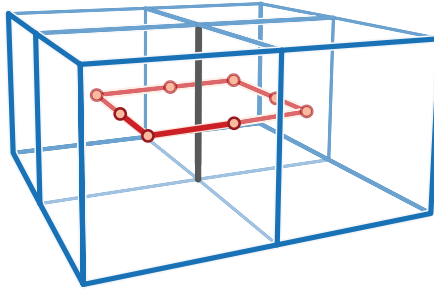
Figure 4.14: The midpoint of a 1-cube (black edge) in the primal grid in 3D is the midpoint of a 2-cube (red square) in the dual grid.

Figure 4.15 illustrates how a three dimensional 1-diamond (Figure 4.15a) can be decomposed into a Kuhn-subdivided 2-cube (Figure 4.15b) and the boundary of a fully subdivided 1-cube (Figure 4.15c).

Let δ be an i-diamond of dimension d, $\mathcal{K}(h_k)$ be the $(d - i)$-dimensional Kuhn-subdivided component of δ and $\mathcal{B}_F(h_i)$ be the fully subdivided i-cube boundary component of δ. The decomposition of Theorem 4.5.1 suggests the following closed-form equations for the number of d-simplices, vertices, parents and children of any diamond δ.

Simplices. The number of d-simplices in an i-diamond is $(d - i)!(2i)!!$ This follows from the fact that δ is defined by the cross complex of $\mathcal{K}(h_k)$ which contains $(d - i)!$ cells

(a) 1-diamond ($d = 3$) **(b)** $\mathcal{K}(h_2)$ **(c)** $\mathcal{B}_F(h_1)$
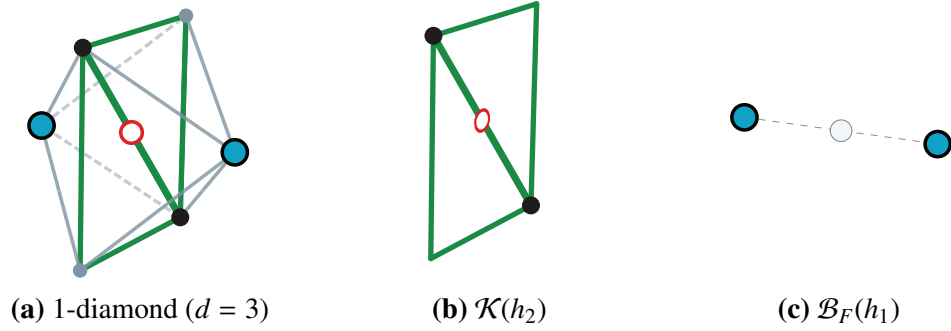
Figure 4.15: A three dimensional 1-diamond (a) can be decomposed into a Kuhn-subdivided 2-cube h_2 (b) and the boundary of a fully subdivided 1-cube h_1 (c). In general, an i-diamond in d dimensions can be decomposed into a Kuhn-subdivided $(d - i)$-cube and the boundary of a fully subdivided i-cube.

and $\mathcal{B}_F(h_i)$ which contains $(2i)!!$ cells. The d-simplices of δ are cross simplices of those from $\mathcal{K}(h_k)$ and $\mathcal{B}_F(h_i)$. Thus, a diamond contains $O(d!)$ cells. Table 4.1 summarizes these properties for $d \leq 5$.

Vertices. The number of vertices in an i-diamond is $(2^{d-i} + 3^i - 1)$. Since $\mathcal{K}(h_k)$ contains 2^{d-i} vertices, and $\mathcal{B}_F(h_i)$ contains $3^i - 1$ vertices and they are both in (pairwise) affinely-independent subspaces, the number of vertices in δ is just their sum. Table 4.2 summarizes these properties for $d \leq 5$.

Children. The number of children of an i-diamond is $2 \cdot (d - i)$ if $i < (d - 1)$ and 2^d if $i = (d - 1)$. The spines of children of an i-diamond, $i < (d - 1)$, coincide with diagonals of the $2 \cdot (d - i)$ facets of the Kuhn-subdivided cube h_k. When $i = (d - 1)$, h_k is a 1-cube (an axis-aligned edge), and h_i is a $(d - 1)$-cube. The spine vertices of δ's children are located at positions 0 and d of each d-simplex, corresponding to one of the two vertices of h_k and one of the 2^{d-1} vertices of h_i. There are thus, 2^d such children. Table 4.3 summarizes these properties for $d \leq 5$.

Parents. The number of parents of an i-diamond is $2 \cdot i$, if $i > 0$ and d if $i = 0$. The central vertex of each parent of an i-diamond, $i > 0$, coincides with the midpoint of one of the $2 \cdot i$ facets of h_i. When $i = 0$, h_i is a 0-cube coinciding with the central vertex of δ. Let σ denote one d-simplex of δ and let (v_0, v_x) denote the spine vertices of its parents, where v_0 is the vertex at position 0 of σ and v_x is the spine vertex of the parent δ_p that generated σ (at position 1 of σ). Then, since δ_p is a $(d - 1)$-diamond, its spine is aligned with a coordinate axis of $\mathbb{R}^d$, and δ has d parents. Furthermore, if $\psi = (v_0, v_d)$ is the spine of δ, then let $\mathbf{v} = v_d - v_0$ be the difference between these vertices. The spine of the jth parent of δ is defined by v_0 and $v_x = v_0 + 2(\mathbf{v} \cdot e_j)$ (where the ($\cdot$) indicates the dot product and e_j is the jth unit vector). Table 4.4 summarizes these properties for $d \leq 5$.

Duets. Since each simplex in the hierarchy is associated with a single diamond, and is generated during the subdivision of a single parent diamond, we can determine

Table 4.1: Number of *d-simplices* in an *i*-diamond of dimension d is $(2i)!!(d-i)!$. The $(d-i)!$ factor comes from the Kuhn-subdivided component $\mathcal{K}(h_k)$ (rows), while the $(2i)!!$ factor comes from the fully-subdivided *i*-cube boundary component $\mathcal{B}_F(h_i)$ (columns).

	0	1	2	3	4	i
1	1					
2	2	2				
3	6	4	8			
4	24	12	16	48		
5	120	48	48	96	384	
d	$d!$	$2(d-1)!$	$8(d-2)!$	$48(d-3)!$	$384(d-3)!$	$(2i)!!(d-i)!$

Table 4.2: Number of *vertices* in an *i*-diamond of dimension d is $2^{d-i} + (3^i - 1)$. The 2^{d-i} term comes from the Kuhn-subdivided component $\mathcal{K}(h_k)$ (rows), while the $(3^i - 1)$ term comes from the fully-subdivided *i*-cube boundary component $\mathcal{B}_F(h_i)$ (columns).

	0	1	2	3	4	i
1	2					
2	4	4				
3	8	6	10			
4	16	10	12	28		
5	32	18	16	30	82	
d	2^d	$2^{d-1}+2$	$2^{d-2} + 8$	$2^{d-3} + 26$	$2^{d-4} + 80$	$2^{d-i} + (3^i - 1)$

the number of *d*-simplices in a parent-child duet as the quotient of the number of simplices and the number of parents. Since the former is $O(d!)$ and the latter is $O(d)$, each duet contains $O(d!)$ simplices that are generated simultaneously. Table 4.5 summarizes these properties for $d \leq 5$

Figure 4.16 illustrates the decomposition of all classes of diamonds for $d \leq 4$. Note that given the decompositions from dimension d, only two new hypercube decompositions are necessary for dimension $(d + 1)$: a Kuhn-subdivided $(d + 1)$-cube and the boundary of a fully subdivided *d*-cube.

4.6   Querying an RSB hierarchy

Observe that the hierarchy of simplices and the hierarchy of diamonds are different hierarchical models over the same family of nested RSB meshes, which we refer to as an *RSB hierarchy*. Whereas the former is focused on the containment relationship among RSB simplices, the latter is focused on the dependency relation required for conforming refinements to an RSB mesh.

We are often interested in extracting conforming meshes from an RSB hierarchy since cracks in non-conforming meshes correspond to discontinuities in functions defined
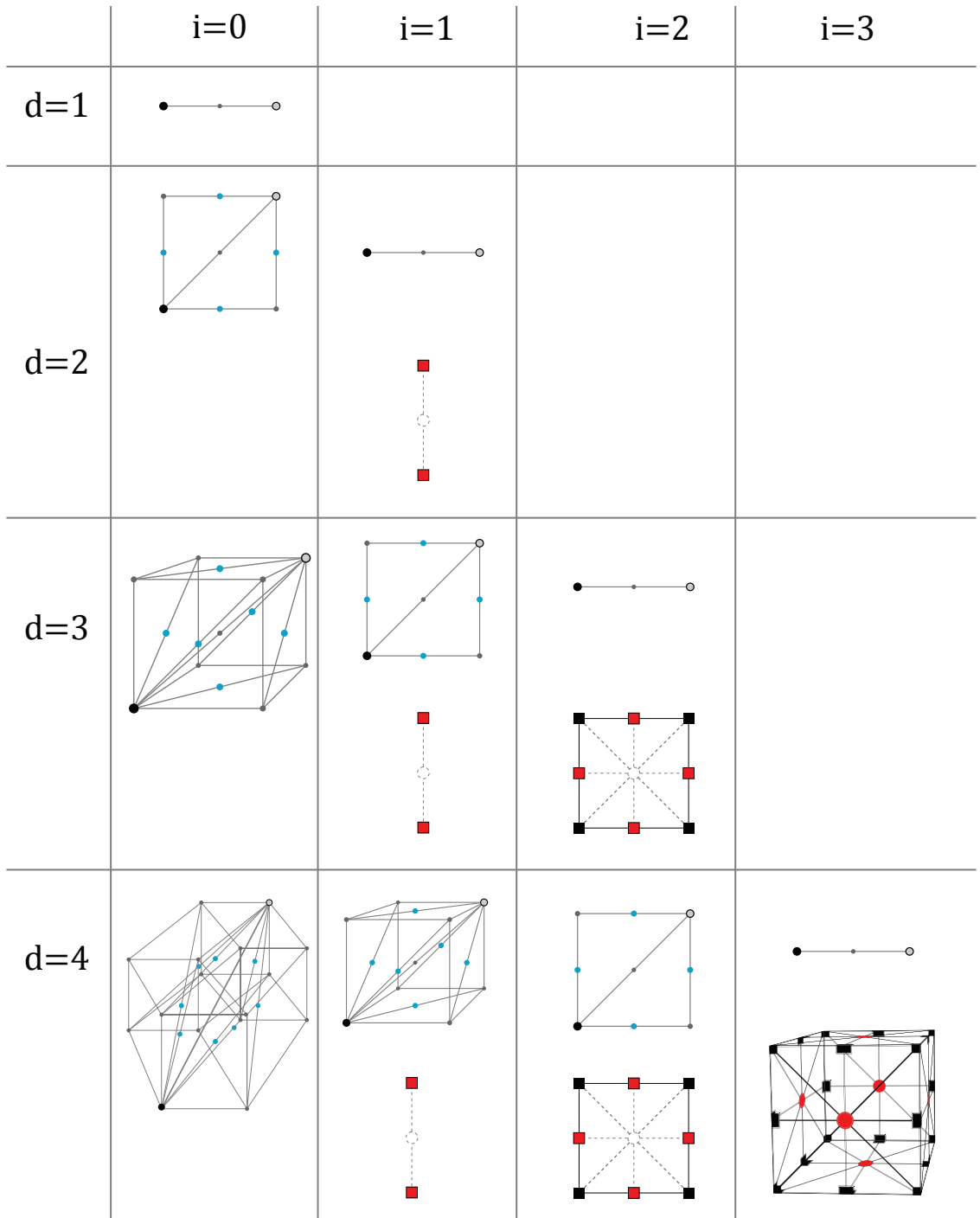
Figure 4.16: An i-diamond δ of dimension d is the cross-complex of $\mathcal{K}(h_k)$, a Kuhn subdivided $(d-i)$-cube, h_k (top cube in each cell) and $\mathcal{B}_F(h_i)$, the boundary of a fully subdivided i-cube, h_i (bottom cube in each cell). For $i < d - 1$, the central vertices of children of δ are located at the midpoints of each $(d-i-1)$-face of h_k (blue vertices). For $i > 0$, the central vertices of parents of δ are located at the midpoints of each $(i-1)$-face of h_i (red vertices). Figures 4.1b and 4.15a illustrate the decompositions for a 2-diamond and a 1-diamond in 3D, respectively.

58

Table 4.3: Number of *children* of an i-diamond in dimension d is 2^d when $i = (d-1)$ and $2(d-i)$, otherwise.

	0	1	2	3	4	i
1	2					
2	4	4				
3	6	4	8			
4	8	6	4	16		
5	10	8	6	4	32	
d	$2d$	$2(d-1)$	$2(d-2)$	$2(d-3)$	$2(d-4)$	$2(d-i) \,\big|\, 2^d$

Table 4.4: Number of *parents* of an i-diamond in dimension d is d when $i = 0$ and $2i$, otherwise.

	0	1	2	3	4	i
1	1					
2	2	2				
3	3	2	4			
4	4	2	4	6		
5	5	2	4	6	8	
d	d	2	4	6	8	$d \,\big|\, 2i$

Table 4.5: Number of d-simplices in a *parent-child duet* from a parent diamond to a child i-diamond is $(2(i-1))!!(d-i)!$ if $i > 0$ and $(d-1)!$ otherwise. Entries are derived as the quotient of corresponding entries from Tables 4.1 and 4.4.

	0	1	2	3	4	i
1	1					
2	1	1				
3	2	2	2			
4	6	6	4	8		
5	24	24	12	16	48	
d	$(d-1)!$	$(d-1)!$	$2(d-2)!$	$8(d-3)!$	$48(d-2)!$	$(d-1)! \,\big|\, (2(i-1))!!(d-i)!$

Algorithm 4.1 ADAPTIVEREFINE(σ)

Require: σ is an RSB simplex in a nested RSB mesh Σ
Require: μ is a selection criterion
  1: **if** $\mu(\sigma)$ fails **then**
  2:        BISECTSIMPLEX(σ)
  3:        ADAPTIVEREFINE(CHILD$_0(\sigma)$)
  4:        ADAPTIVEREFINE(CHILD$_1(\sigma)$)

---

on those meshes. We use a selective refinement process to extract a conforming RSB mesh from an RSB hierarchy. This extracts the mesh with the fewest possible elements satisfying an application-dependent predicate μ, referred to as the selection criterion.

Recall from Section 2.4.1 that selective refinement is performed by traversing the graph describing the dependency relation either top-down by starting from a coarse approximation, bottom-up by starting with the mesh at full resolution or incrementally by modifying an already extracted mesh. In this process, a conforming RSB mesh Σ, referred to as the *current mesh*, is extracted from the hierarchy. The status of a query is described by a cut C of the hierarchy's dependency graph, called the *active front*, separating the set of modifications that have been applied from those that have not.

A conforming RSB mesh corresponds to a set of modifications that is *closed* with respect to the diamond dependency relation. Conforming refinements to an RSB mesh correspond to the subdivision of *complete* diamonds, i.e. where all simplices are present in the current mesh.

Alternatively, a nested RSB mesh corresponds to a set of modifications that is *closed* with respect to the containment hierarchy induced by bisections, and does not need to be conforming. Such meshes are extracted from a hierarchy of simplices using a simpler *adaptive refinement* query, outlined in Algorithm 4.1, which does not involve backtracking or neighbor finding. An adaptive refinement query is initialized with the $d!$ bintree roots. Nodes that fail the selection criterion μ are bisected and their children are tested recursively.

When the hierarchy is modeled as a hierarchy of simplices, a conforming mesh is extracted through the use of a *saturated* selection criterion (see Section 3.1.4.2), which incorporates the diamond dependency relation into the selection criterion, in conjunction with an adaptive refinement query, or through a selective refinement query [DL04] in conjunction with a neighbor-finding algorithm.

In the former case, saturation of the selection criterion, ensures that the ancestors of simplices along the bisection edge refine before it does, so tests against a node's ancestors are not necessary to guarantee the extraction of conforming meshes. In the latter case, the neighbor-finding algorithm recursively bisects simplices that are at shallower depths than the bisecting node. In either case, since the number of simplices in a diamond is factorial in the dimension d, the complexity of conforming updates to a simplex-based hierarchy is $O(d!)$.

In contrast, when the RSB hierarchy is modeled as a hierarchy of diamonds Δ, diamonds are complete once all of their parent diamonds have subdivided. This completion

---
Algorithm 4.2 SELECTIVEREFINE(δ, ForceRefine)

Require: δ is a diamond in a nested RSB mesh Σ
Require: ForceRefine is a boolean
Require: μ is a selection criterion
1: **if** ForceRefine is **true or** $\mu(\delta)$ fails **then**
2:     // *Ensure diamond is complete*
3:     **for all** $\delta_p \in$ *Parents*(δ) **do**
4:         **if** δ_p is not subdivided **then**
5:             SELECTIVEREFINE(δ_p, **true**)
6:     // *Bisect all simplices of δ*
7:     SUBDIVIDEDIAMOND(δ)
8:     // *Check all children*
9:     **if** ForceRefine is **false then**
10:        **for all** $\delta_c \in$ *Children*(δ) **do**
11:            SELECTIVEREFINE(δ_c, **false**)
---

process is carried out by (recursively) forcing all parents of δ to refine, thereby satisfying the transitive closure of the dependency graph. The active front of the query consists of the arcs of the dependency graph connecting subdivided diamonds to their unsubdivided children. Thus, the current mesh Σ consists of the simplices associated with the parent-child duets corresponding to these arcs (see Figure 4.17).

Algorithm 4.2 outlines a top-down selective refinement query for a hierarchy of diamonds, which is initialized using the root diamond of the hierarchy. Most diamonds are checked against the selection criterion μ. However, forced refinements short-circuit the selection criterion using the boolean ForceRefine (Line 1). The algorithm consists of three steps. First, we complete the diamond by recursively subdividing its parents (Lines 3–5). Next, the diamond is subdivided (Line 7). Finally, the children of diamonds that are not forcibly refined are checked for refinement (Lines 9–11). Since a diamond δ has $O(d)$ parents, and each diamond is refined only once, diamond refinements in SELECTIVEREFINE have an amortized complexity of $O(d)$.

Figure 4.17 illustrates the results of a selective refinement query on a 2D hierarchy of diamonds after subdividing the root diamond (filled red circle), three of its children (filled orange circles), and one of its grandchildren (filled green circle).

4.7 Discussion

We have generalized the notion of a diamond to arbitrary dimensions as a cross-complex of two related simplicial decompositions of lower-dimensional hypercubes. This has enabled us to analyze the properties of diamonds and to derive closed-form equations for the number of d-simplices, vertices, parents and children of all types of diamonds in arbitrary dimensions.

In particular, we proved that an i-diamond in d-dimensions contains $(d-i)!(2i)!!$ d-simplices. Thus, representations in which the primitives are d-simplices become very

61

(a) Closed set of modifications   **(b)** Active front   **(c)** Current mesh Σ
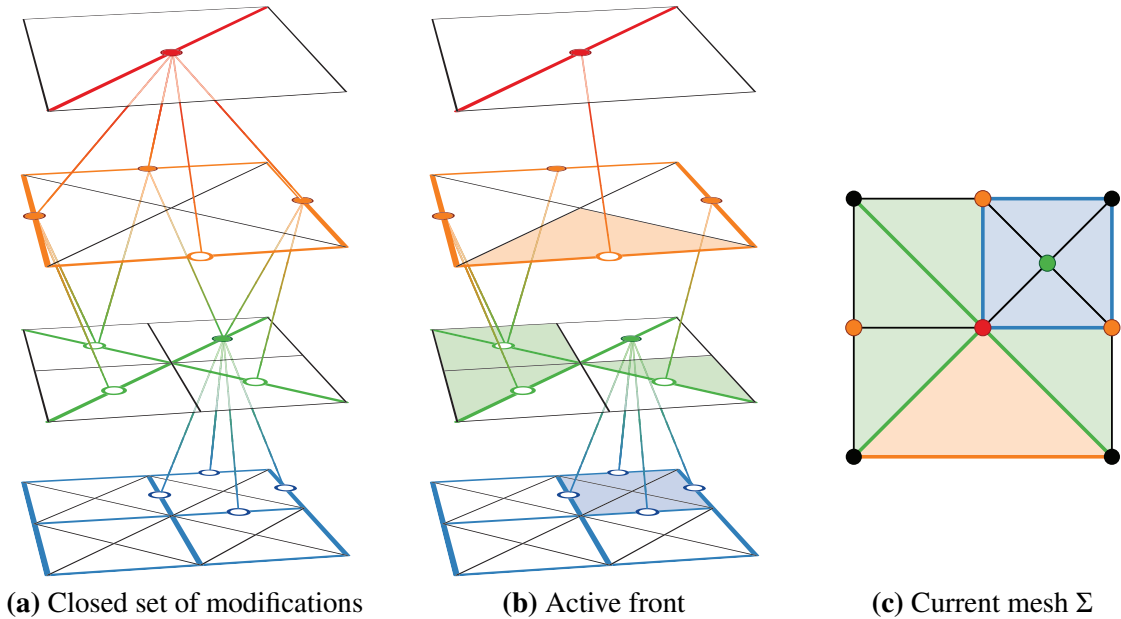
Figure 4.17: During *selective refinement*, a conforming RSB mesh Σ (c) is extracted from a hierarchy of diamonds Δ through a traversal of its dependency graph (a). Σ corresponds to the *active front* (b), a *closed* cut of the DAG describing Δ's dependency relation (a). Simplices in the *current mesh* (c) belong to duets in the active front and correspond to subdivided parents (filled circles) of unsubdivided diamonds (unfilled circles).

expensive to store as the dimension d of the problem domain increases. Specifically, since neighbor-finding operations are required for extracting conforming meshes before any bisection operation, extracting conforming modifications to a simplicial complex is a problem with $O(d!)$ complexity.

We also identified *parent-child duets*, which are in one-to-one correspondence with the arcs of the dependency graph of a diamond hierarchy, as the atomic building block of conforming RSB meshes. This is useful for our efficient selective refinement query (Algorithm 4.2) for compact encodings of RSB meshes (Chapter 6), for efficiently traversing these meshes (Chapter 8) and for transmitting information down the hierarchy during refinement (Chapter 9). Since a diamond has $O(d)$ parents, a diamond-based approach enables conforming updates to an RSB mesh in (amortized) linear time and space.

Although saturated metrics are typically applied to simplex-based representations, they subsume the diamond hierarchy into the selection criterion. Thus, our analysis of the diamond-dependency relation implies that a diamond-based generation for saturated metrics have $O(d)$ complexity, while the typical simplex-based generation, in which errors are passed up through the containment relation, have $O(d!)$ complexity.

Since diamond hierarchies satisfy the MultiTessellation (MT) model [DPM97], and are defined by stellar refinement along an edge of the mesh, we can view a diamond hierarchy as a restricted version of a multiresolution model based on half-edge collapses [DDM$^+$05]. Consider the $(d \cdot N)^{\text{th}}$ Maubach complex $\mathcal{M}_{d \cdot N}$, for some $n \in \mathbb{N}$. Then, if we restrict our edge collapse operations to subdivided diamonds (with all of their

simplices), and collapse the central vertex of such a diamond into one of its spine vertices, we obtain a partial order on these half-edge collapses that is equivalent to that of the diamond dependency relation.

Finally, we observe that the hierarchy of simplices defines a containment hierarchy over the domain, but nested bisections alone are not sufficient to guarantee the extraction of a conforming mesh, which requires a saturated selection criterion or a neighbor finding algorithm. On the other hand, the diamond dependency relation ensures the extraction of conforming meshes but does not define a containment hierarchy. Thus, conforming refinements require the subdivision of a diamond's parents before it can subdivide, which can propagate refinements up the hierarchy. Thus, a *nested refinement domain* would enable a simple top-down adaptive refinement algorithm that guarantees the extraction of conforming meshes. In 2D, the octagonal *descendant domain* of a diamond [Tan95, BLV03, Ger03b] (see Figure 3.4) defines a nested hierarchy of octagons satisfying these conditions.

A promising initial result towards this aim for three dimensional diamond hierarchies [WD10c] builds on the 2D octagonal *descendant domains* to define nested refinement domains in 3D. We observe that the descendant domain of a diamond has a fractal boundary (see Figure 4.18b) which would be difficult to work with in an interactive setting, and introduce two new nested refinement domains that would be easier to work with. The *convex descendant domain* of a diamond is the convex hull of its descendant domain (see Figure 4.18c), and the bounding box descendant domain is the axis-aligned bounding box of its descendant domain (see Figure 4.18d). All three nested refinement domains extend the domain under consideration by at most a factor of three. The original descendant domain is typically the region under consideration for saturated error metrics [Paj98, GP00], but can be used to precompute the range of field values covered by all descendants (i.e. a saturated analogue of the Min/Max octree [WVG92]). This can also be conservatively estimated for a diamond δ from an unsaturated Min/Max range by considering the range within a constant number of diamonds that cover the convex descendant domain or bounding box domain of δ. The bounding box domain can be used to quickly determine if a diamond intersects a geometric object in the domain, for example, in hierarchical frustum culling algorithms.
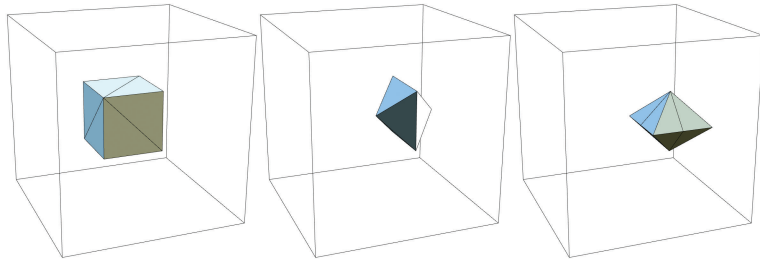
(a) Three classes of diamonds in 3D
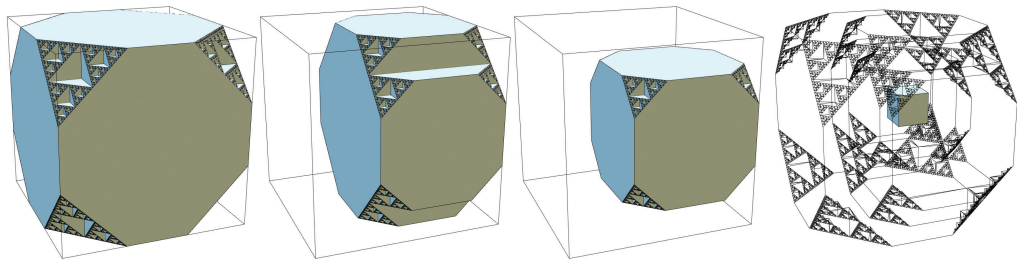
(b) Descendant domains
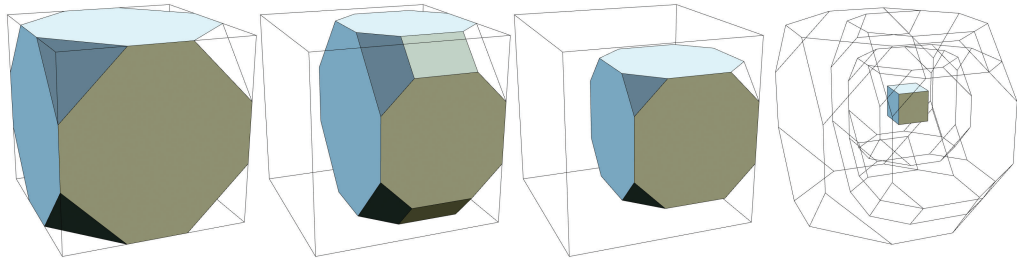
(c) Convex descendant domains
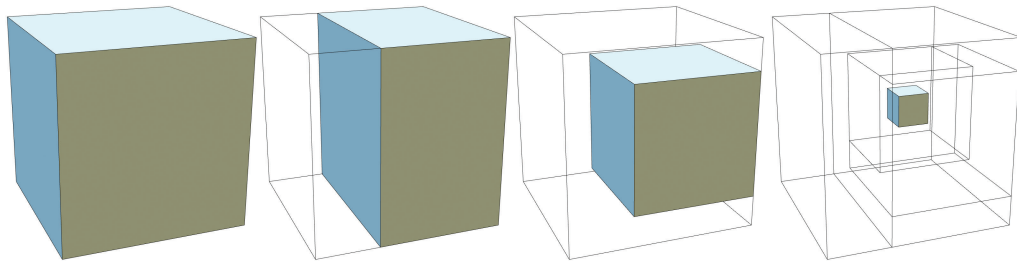
(d) Bounding box descendant domains

Figure 4.18: The three classes of diamonds (a) and their corresponding nested refinement domains (b-d) in 3D. In each case, the corresponding refinement domain of one of the diamond's parents, grandparents and great-grandparents (right column) illustrates the nested nature of these shapes.

Chapter 5

Supercubes: A high-level primitive for RSB hierarchies

We are often interested in associating information with *coherent* subsets of the entities of an RSB hierarchy. This includes its geometric entities, such as its vertices, edges, simplices and diamonds; its hierarchical entities, such as the parents or children of an element; and the connectivity among neighboring elements. A common approach is to associate information with only the desired elements, and to index these in a spatial data structure. However, this can impose significant overhead when an element's index requires more storage than the data we would like to associate with it.

Due to the way RSB meshes are generated, there is typically a great deal of coherence among its elements. For example, when dealing with a variable-resolution diamond mesh extracted from the hierarchy, we have a sparse subset of the total dataset, but the presence of a diamond in the mesh strongly implies the presence of its neighbors or those of its parents or children, in the mesh.

In particular, since diamond hierarchies are defined in terms of the subdivision of diamonds along their spines, and vertices are inserted at the midpoints of these edges, it is useful to consider the spatial and hierarchical relationships among the edges of a nested RSB mesh.

An analysis of this structure reveals a higher level of symmetry within the hierarchy than that which is apparent at the level of diamonds. Specifically, each level of resolution is tiled by a repeating pattern of edges arranged in a cubic domain, which we call a *supercube* and derive from the fully subdivided hypercubes of Section 4.2.3.

In this chapter, we analyze fully subdivided hypercubes to better understand the structure of nested RSB meshes (Section 5.1). We then propose supercubes as the basis for a higher-level primitive within RSB hierarchies, and analyze their properties in Section 5.2. In Section 5.3, we discuss some implications of supercubes. We present our encoding of the elements within a supercube and of collections of supercubes in Chapter 6.

5.1 Tiling space with Kuhn cubes

As discussed in Section 4.2.1, one of the interesting properties of the Kuhn subdivision $\mathcal{K}(h)$ of a hypercube h is that it provides a Kuhn subdivision to the faces of h (Theorem 4.2.1). Furthermore, opposite faces of $\mathcal{K}(h)$ are compatibly decomposed, so a regularly sampled domain can be tiled by Kuhn-subdivided cubes [Kuh60, Tod76] (see triangulations in Figure 5.3).

When a hyper-rectangular domain is tiled by Kuhn cubes using only translation, this tiling is referred to as *Freudenthal's triangulation* [Tod76] and is typically denoted as K_1 (see Figure 5.1a). Alternatively, the *Tucker-Whitney triangulation* [Tuc45, Whi57], typically denoted as J_1, is obtained over the same grid by reflecting adjacent Kuhn cubes

(a) K_1 triangulation   **(b)** J_1 triangulation
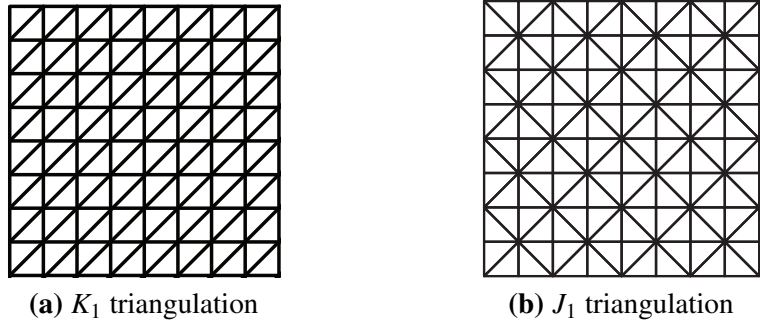
Figure 5.1: Tiling the plane with Kuhn squares. (a) Translating adjacent tiles leads to *Freudenthal's triangulation* K_1. (b) Reflecting adjacent tiles leads to the *Tucker-Whitney triangulation* J_1.

(a) Level ℓ **(b)** Level $\ell + 1$ **(c)** Level $\ell + 2$
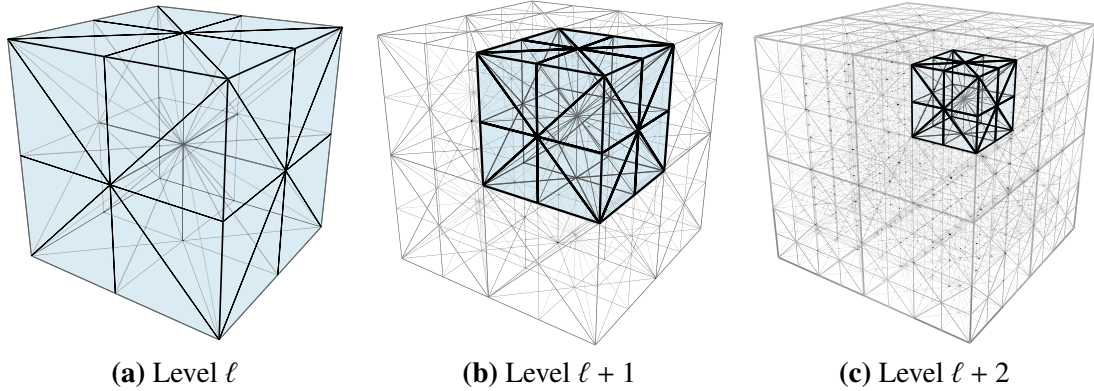
Figure 5.2: The J_1 triangulation is the canonical tiling for RSB hierarchies. It tiles each level of resolution of d-dimensional space with scaled copies of fully subdivided d-cubes. Three consecutive levels of resolution covering the same 3D domain are shown, containing 1, 8 and 64 cubes, respectively. One fully subdivided cube at each level or resolution is highlighted in blue.

across cube facets. Due to the reflectional symmetry, J_1 can be viewed as a tiling of an integer lattice by clusters of 2^d oriented Kuhn-cubes, which define the *fully subdivided hypercubes* $\mathcal{F}$ (see Figures 5.1b and 5.2 for examples in 2D and 3D).

Given a Kuhn-subdivision $\mathcal{K}(\Omega)$ of a hypercubic domain Ω, Freudenthal's triangulation [Fre42, Bey00], generates K_1 by applying *regular refinement* to its simplices (blue arrows in Figure 5.3). Alternatively, this decomposition can be obtained by applying Kuhn's subdivision to the leaves of a complete 2^d-tree (red arrows in Figure 5.3). Since K_1 is generated using regular refinement of hypercubes or of simplices these meshes are referred to as *triangle quadtrees* in 2D [LS00], *tetrahedral octrees* in 3D and generally as *simplicial 2^d-trees* [MW95].[*] K_1 is therefore the canonical decomposition for red/green refinement meshes.

In contrast, the Tucker-Whitney triangulation J_1 can be obtained by applying RSB operations (a multiple of d times) to all elements of $\mathcal{K}(\Omega)$ [Mau96] (green arrows in Fig-

---

[*]Actually, Moore and Warren [MW95] refer to the d-dimensional variant as a *simplicial quadtree*.

ure 5.3), or by reflecting the Kuhn-triangulations in facet-adjacent hypercubes (red arrows in Figure 5.3). Thus, J_1 is the canonical decomposition for RSB meshes.

Properties of fully subdivided hypercubes. Let us consider the number of edges in a fully subdivided hypercube $\mathcal{F}(h)$ of dimension d. Recall that the number of i-faces of a d-cube is $\binom{d}{i}2^{d-i}$. The subdivision $\mathcal{F}(h_i)$ of each i-face $h_i \subseteq h$ induced by $\mathcal{F}(h)$ has 3^i vertices, and there is an edge from the midpoint of $\mathcal{F}(h_i)$ to each of the $3^i - 1$ vertices on its boundary (see Figures 4.5c and 4.6d). Since each of these edges is internal to h_i, and not to any other face of h, we can count all of the unique edges within $\mathcal{F}(h)$ as:

$$\sum_{i=0}^{d} \binom{d}{i}2^{d-i}(3^i - 1) = \sum_{i=0}^{d} \binom{d}{i}2^{d-i} \cdot 3^i - \sum_{i=0}^{d} \binom{d}{i}2^{d-i} \tag{5.1}$$
$$= 5^d - 3^d.$$

The latter equality follows from the binomial theorem (see Appendix C).

Thus, there are $5^d - 3^d$ edges in a fully subdivided d-cube.

5.2 Supercubes

Although fully-subdivided cubes are the underlying symmetry unit within RSB hierarchies, and tile each level of resolution of J_1, we observe that simplices on their boundaries can be incident to several fully-subdivided cubes. We therefore introduce the *supercube* as a high-level primitive for RSB hierarchies.

Supercubes provide a unique mapping from each element within the hierarchy to a single entity through the use of the *half-open interval* convention [Sam06]. That is, each supercube is associated with the simplices at a given level of resolution within the RSB hierarchy that are within its domain or that coincide with its lower boundaries but not those that coincide with its upper boundaries. For example, we consider any edge of a fully subdivided cube $\mathcal{F}$ whose endpoints are both on an upper boundary of $\mathcal{F}$, to belong to a neighboring supercube. Figure 5.5b illustrates the edges that remain in two-dimensional and three-dimensional supercubes after applying the half-open interval convention to their corresponding fully subdivided cubes. Solid lines indicate edges that remain in the supercube, and dashed lines indicate the edges whose endpoints both lie on an upper boundary of the supercube, and are thus indexed by a neighboring supercube. Figure 5.4 illustrates supercubes covering the same domain at three levels of resolution. Note that each edge is only associated with a single supercube.

Entities within the hierarchy that are not simplices can be associated with supercubes through a representative *proxy* simplex. For example, due to the one-to-one correspondence between diamonds and their spines, a diamond can be associated with the same supercube as its spine.

Let the *origin* of a hypercube h be the vertex of h that is at its lower interval in all dimensions. For example, on the unit cube, the origin is the vertex 0^d, and its opposite vertex along a diagonal is 1^d.

We first consider the number of i-faces of h that remain when utilizing the half-open
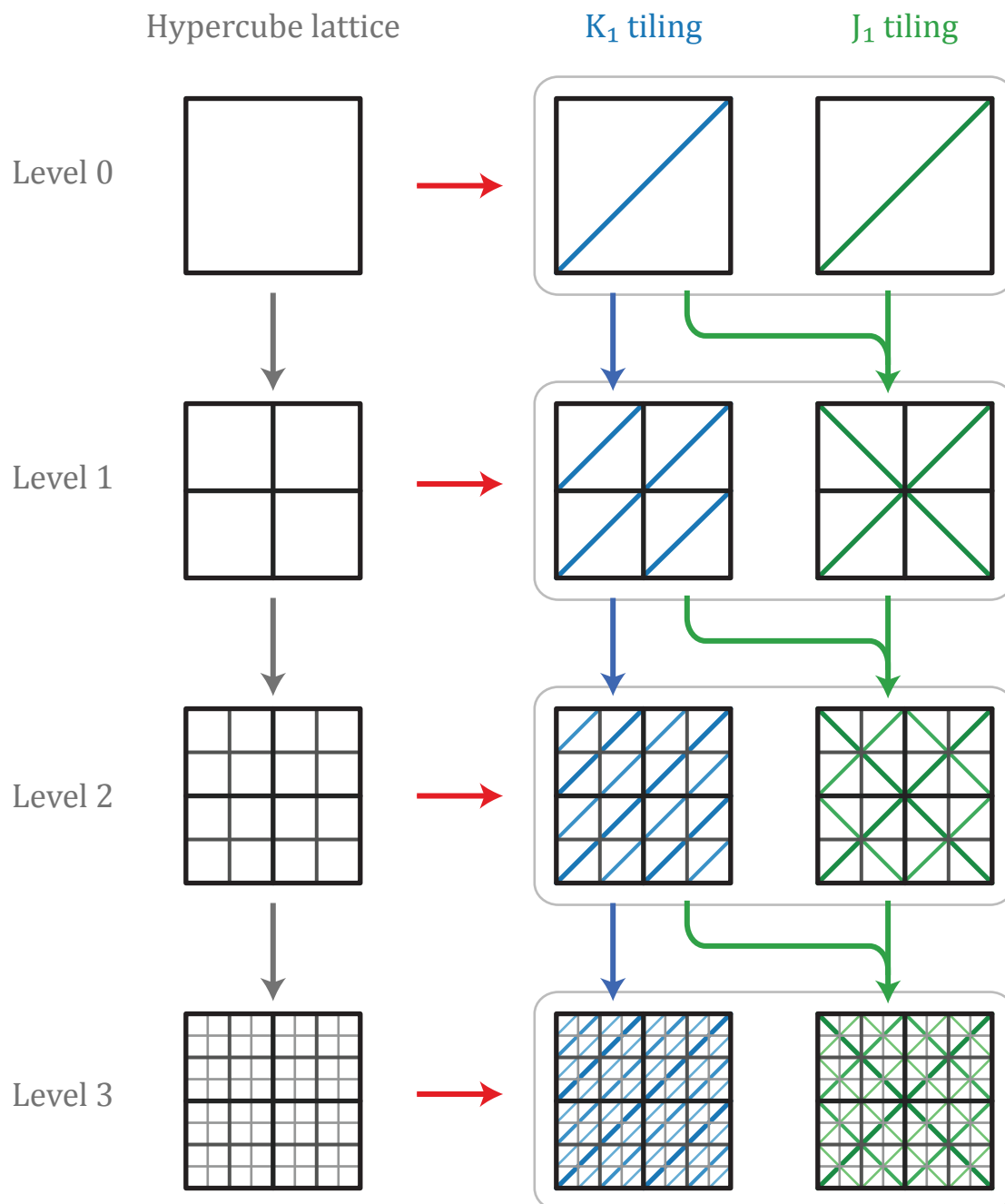
Figure 5.3: Relationship between regular refinement of hypercubes, the *Freudenthal triangulation* K_1 and the *Tucker-Whitney triangulation* J_1. Regular refinement of hypercubes produces a regular hypercubic lattice, i.e. a complete quadtree, octree or 2^d tree (gray arrows). A complete 2^d-tree can be triangulated through a Kuhn-subdivision of its leaves (red arrows). Tiling by translation across hypercube facets produces the Freudenthal triangulation K_1, while tiling by reflection across facets produces the Tucker-Whitney triangulation J_1. Applying regular refinement to each simplex in K_1 at level ℓ produces K_1 at level $\ell + 1$ (blue arrows). Alternatively, applying d steps of RSB to each simplex in K_1 or J_1 at level ℓ produces J_1 at level $\ell + 1$ (green arrows).

|(a) Level ℓ|(b) Level $\ell + 1$|(c) Level $\ell + 2$|
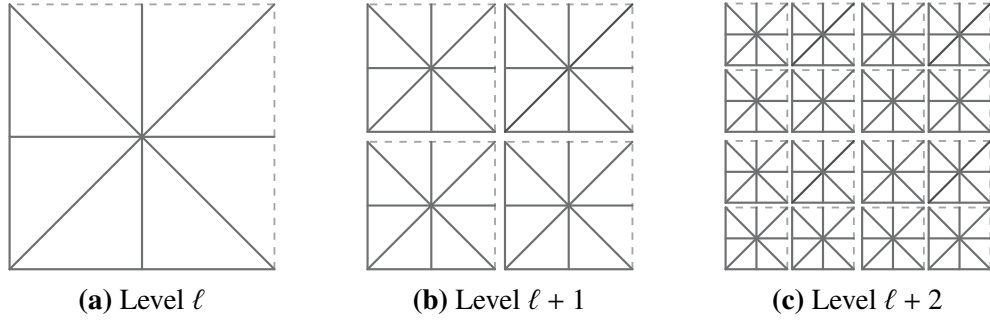
Figure 5.4: Supercubes (in 2D) are structured sets of edges tiling each level of resolution within an RSB hierarchy. Three consecutive levels of resolution covering the same domain are shown, containing 1, 4 and 16 supercubes, respectively. Dashed edges are excluded due to the half-open interval rule.

interval convention. We do so through a mapping of h to the unit cube, where the origin is mapped to 0^d and its opposite vertex is mapped to 1^d. Then an i-face of h that is incident to the origin will have a diagonal along i of the d coordinate axes, but not the remaining $(d - i)$ coordinate axes, i.e. its opposite vertex will have i coordinates at position 1 and $(d - i)$ coordinates at position 0 along its axis. The number of i-faces within the half-open interval is thus $\binom{d}{i}$, which leaves a total of $\sum \binom{d}{i} = 2^d$ faces of the original 3^d faces remaining after applying the half-open interval convention. Note that each such face can be mapped to a vertex of h defined by the unique diagonal from the origin to this vertex (see Figure 5.5).

Number of edges. We are now able to determine the number of edges in a supercube **s** as the number of edges in a fully subdivided cube $\mathcal{F}$ that remain after applying the half-open interval condition. Since the number of i-faces is $\binom{d}{i}$, and each i-face contributes $(3^i - 1)$ edges, there are a total of

$$\sum_{i=0}^{d} \binom{d}{i}(3^i - 1) = \sum_{i=0}^{d} \binom{d}{i}3^i - \sum_{i=0}^{d} \binom{d}{i} \tag{5.2}$$
$$= 4^d - 2^d$$

edges remaining in a supercube (see Figure 5.5b).[†]

Number of vertices. To determine the number of vertices in a supercube, we rearrange

---

[†]This equality follows from the binomial theorem, see Appendix C.
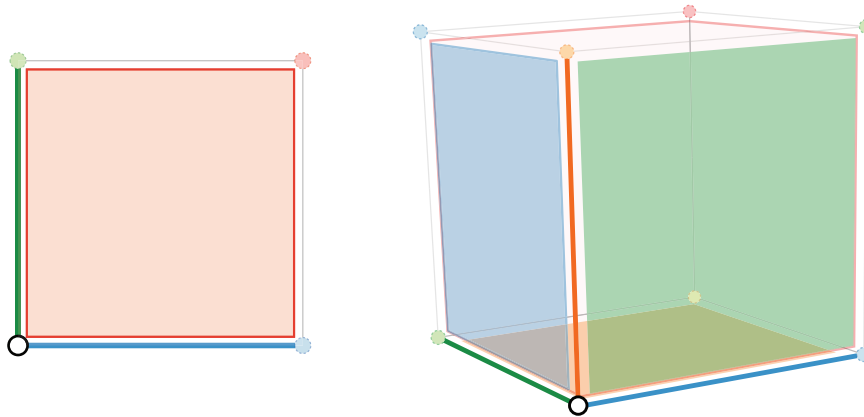
(a) Faces of a half-open *d*-cube
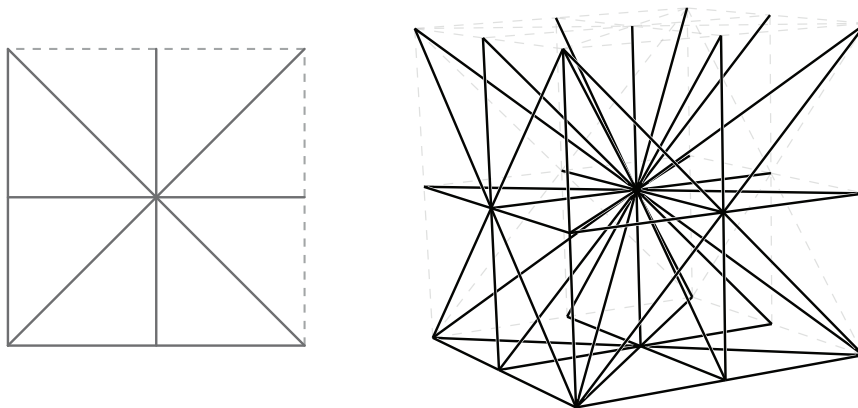
(b) Edges of a supercube

Figure 5.5: (a) A half-open *d*-cube retains the $\binom{d}{i}$ *i*-faces incident to its origin. In particular it has a single *d*-face (red) and a single vertex corresponding to its *origin* (white). (b) Each *i*-face of a half-open cube contributes its $3^i - 1$ internal edges to a supercube.
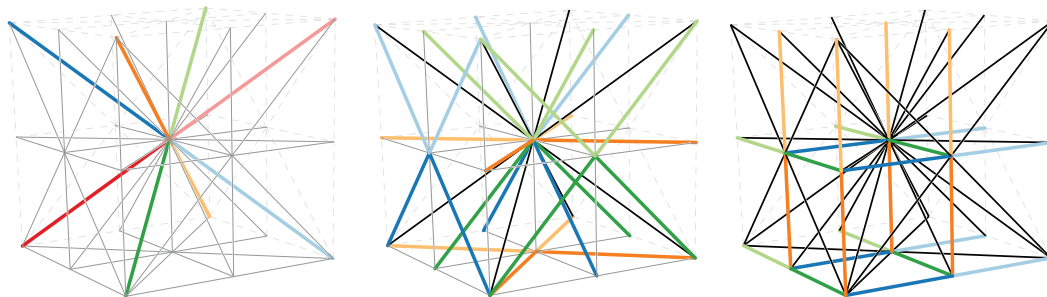
(a) Cube diagonals   **(b)** Face diagonals   **(c)** Cube edges

Figure 5.6: Edges of a three-dimensional supercube **s** that remain after applying the half-open interval to its corresponding fully subdivided cube. The edge colors highlight the 2^d copies of $\binom{d}{i}$ *types* of *i*-diamonds within **s**. (a) Eight edges aligned with a cube diagonal. (b) Eight groups of three edges aligned with a face diagonal of a cube. (b) Eight groups of three edges aligned with an edge of a cube.

the final term of Equation 5.2 as:

$$4^d - 2^d = 2^d \cdot (2^d - 1)$$
$$= 2^d \cdot \left[\sum_{i=0}^{d} \binom{d}{i} - 1 \right] \tag{5.3}$$
$$= 2^d \cdot \sum_{i=1}^{d} \binom{d}{i}$$

which better highlights the geometry of a fully subdivided hypercube.[‡] That is, a supercube **s** is defined by 2^d hypercubes, each locally satisfying the half-open interval convention. The single face of each such such hypercube that does not correspond to an edge in **s** is a vertex of **s** (i.e. a 0-face). So there are 2^d vertices remaining, which correspond to the origins of the 2^d Kuhn cubes that comprise the supercube **s**.

However, these vertices do not uniquely correspond to a single supercube. Consider the origin of a supercube at level ℓ, as in Figure 5.4a. Then this point belongs to a supercube at each successive level within the hierarchy (e.g. the lower left corners in Figures 5.4(b) and (c)).

Instead of mapping vertices of the hierarchy to supercube vertices, we can use the correspondence between edges in the hierarchy and their unique midpoints to uniquely map the vertices. So, each supercube uniquely indexes $4^d - 2^d$ vertices of the hierarchy.

Number of simplices. Supercubes also provide a mapping to the d-simplices within the hierarchy, as considered by Hebert in 2D [Heb98] and 3D [Heb94] and generalized to arbitrary dimensions by Pascucci [Pas00] and by Atalay and Mount [AM07] under the term *subtree of simplices*.

Consider a d-simplex σ of class-i at depth $m = (d \cdot \ell + i)$ in a d-dimensional RSB hierarchy. Then we can index σ according to its containing Kuhn-cube $\mathcal{K}_\sigma$ at depth $d \cdot \ell$ in the hierarchy (see Figure 5.7a), its class-0 ancestor simplex σ_0 within $\mathcal{K}_\sigma$ (see Figure 5.7b) and a traversal of the local bintree from σ_0 to σ (see Figure 5.7c).

Since its containing Kuhn cube $\mathcal{K}_\sigma$ is one of the 2^d subcubes within a supercube **s**, we refer to this as its *reflection number* $i_r \in [0, 2^d)$. Its class-0 ancestor σ_0 is one of the $d!$ class-0 simplices within $\mathcal{K}_\sigma$, which we refer to as its *permutation number* $i_p \in [0, d!)$. Finally, the bintree rooted at σ_0 is a complete binary tree of depth $d - 1$. This provides σ with a unique *descendant number* $i_d \in [0, 2^d - 1)$. Thus, a supercube uniquely indexes $2^d \cdot (2^d - 1) \cdot d!$ distinct RSB simplices.

Number of diamonds. The one-to-one correspondence between edges of an RSB hierarchy and the spines of diamonds provides a unique association from each diamond to a single supercube. Specifically, each supercube indexes $4^d - 2^d = 2^d(2^d - 1)$ diamonds.

Furthermore, we can break this down by diamond class to see how many copies of each class of diamond are in a supercube. Recall that the spine of an i-diamond is aligned with the diagonal of an axis-aligned $(d - i)$-cube, which is an i-face of a d-cube.

[‡]The latter term is achieved by incorporating the -1 into the sum's index using the identity $\binom{d}{0} = 1$.

(a) Reflection i_r **(b)** Permutation i_p **(c)** Descendant i_d
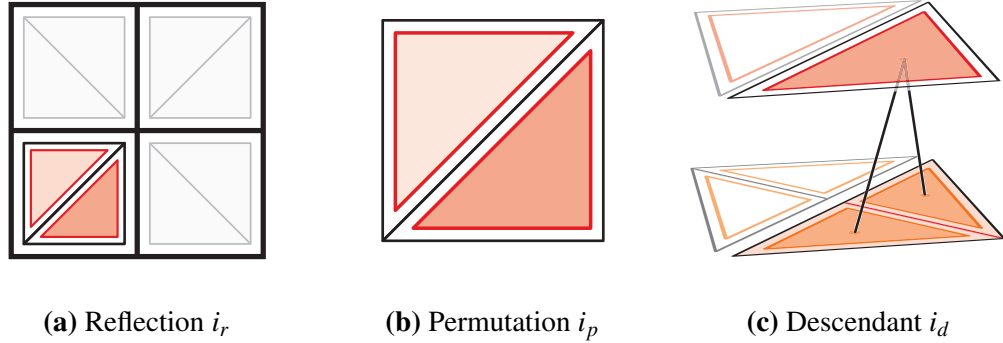
Figure 5.7: Each RSB simplex σ within a supercube (shown in 2D) is uniquely indexed by the *reflection number* i_r of its containing Kuhn cube $\mathcal{K}_\sigma$ (a), the *permutation number* i_p of its containing class 0 simplex σ_0 within $\mathcal{K}_\sigma$ (b) and the *descendant number* i_d of σ from σ_0 (c).

Returning to Equation 5.3, we see that there are $\binom{d}{i}$ i-diamonds associated with each of the 2^d Kuhn-cubes within a supercube **s** (see Figure 5.6).[§]

Figure 5.8 illustrates the four 0-diamonds and twelve 1-diamonds that map to each 2D supercube. Note that the spines of the diamonds coincide with supercube edges and that the midpoint of each edge coincides with the diamond's central vertex. Similarly, Figure 5.9 illustrates the eight 0-diamonds, twenty-four 1-diamonds and twenty-four 2-diamonds that map to each 3D supercube. Observe that there are 2^d copies of each of the following (a) one 0-diamond in Figure 5.9a (blue, green, orange or red); (b) three 1-diamonds in Figure 5.9b (orange, green and blue); and (c) three 2-diamonds in Figure 5.9b (orange, green and blue).

By comparing the edges of a supercube (Figures 5.6) to its diamonds (Figures 5.8 and 5.9), we see that there is a directional bias to the alignment of diamond spines due to the directional bias within reflected Kuhn-cubes. However, when considered at the level of a supercube, the diamonds at the same relative positions are identical. We therefore refer to a diamond at a particular location within a supercube as a unique diamond *type*, of which there are $4^d - 2^d$. Table 5.1 lists the number of i-diamonds in a d-dimensional supercube, $d \leq 5$.

Number of parents. We can also consider a mapping from the set of diamond parents to a supercube. For this, we can associate the set of parents of a given diamond to the edge of the supercube corresponding to its spine. This is an especially interesting property due to the one-to-one correspondences between parents of a diamond, arcs of the the dependency graph and duets in the hierarchy.

Recall that a 0-diamond has d parents, and an i-diamond has $2i$ parents otherwise. Then, by multiplying the number of diamond classes by the number of parents per dia-

---

[§]For this, we used the binomial identity $\binom{d}{i} = \binom{d}{d-i}$ to change the meaning of the terms to i-diamonds which correspond to the $(d-i)$-faces

Table 5.1: Number of i-diamonds in a d-dimensional supercube is $\binom{d}{i}2^d$.

	0	1	2	3	4	i	Total
1	2						2
2	4	8					12
3	8	24	24				56
4	16	64	96	64			240
5	32	160	320	320	160		992
d	$\binom{d}{0}2^d$	$\binom{d}{1}2^d$	$\binom{d}{2}2^d$	$\binom{d}{3}2^d$	$\binom{d}{4}2^d$	$\binom{d}{i}2^d$	$4^d - 2^d$

(a) Twelve vertices   **(b)** Four 0-diamonds   **(c)** Eight 1-diamonds
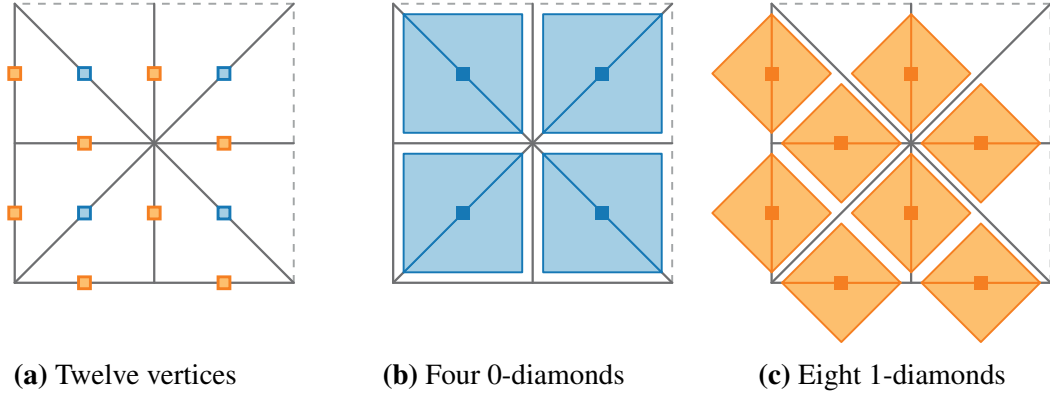
Figure 5.8: A 2D supercube **s** contains four 0-diamonds (b) and eight 1-diamonds (c), whose central vertices coincide with the midpoints of its twelve edges (a).

mond, we find the number of parents in a supercube to be:[¶]

$$
2^d \cdot d + 2^d \sum_{i=1}^{d-1} \binom{d}{i} \cdot 2i = 2^d \cdot \left(d + \sum_{i=1}^{d-1} \binom{d}{i} \cdot 2i \right)
$$
$$
= 2^d \cdot \left(\sum_{i=0}^{d} \binom{d}{i} \cdot 2i - d \right) \tag{5.4}
$$
$$
= 2^d \cdot \left(2^d \cdot d - d \right)
$$
$$
= 2^d \cdot (2^d - 1) \cdot d.
$$

Since there are $2^d \cdot (2^d - 1)$ diamonds in a supercube, the average number of parents per diamond in a supercube is d.

[¶]The inductive proof that $\sum_{i=0}^{d} \binom{d}{i} \cdot 2i = 2^d d$ is presented in Theorem C.2.1 of Appendix C.

(a) Eight 0-diamonds   **(b)** Twenty-four 1-diamonds   **(c)** Twenty-four 2-diamonds
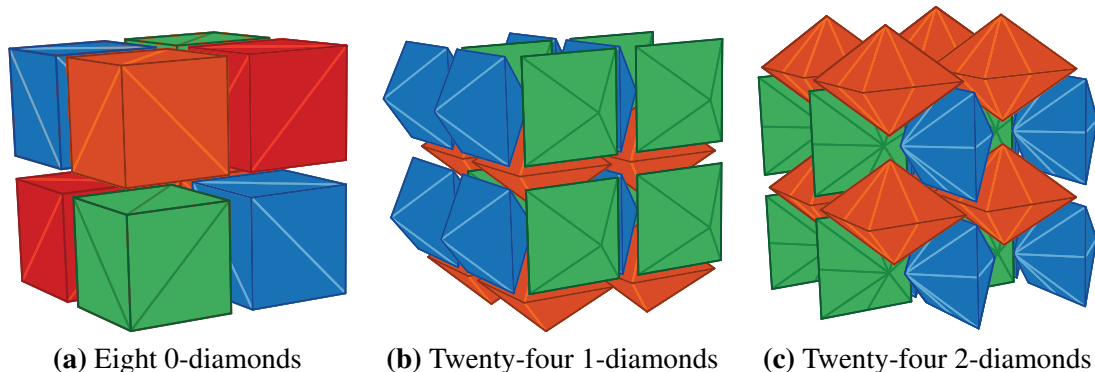
Figure 5.9: A supercube in 3D is composed of eight 0-diamonds (a) twenty-four 1-diamonds (b) and twenty-four 2-diamonds (c). Alternatively, it contains 2^d copies of a single 0-diamond, three 1-diamonds and three 2-diamonds.

5.3 Discussion

Supercubes capture the symmetry of the decomposition structure within the RSB hierarchy. In contrast to diamonds, which have d distinct similarity classes and different spine orientations within each class, or to RSB simplices which also vary along the various vertex permutations, all supercubes are identical up to translation and uniform scaling by a factor or 2.

This suggests the utility of supercubes as an *algorithmic primitive* for processing RSB hierarchies and for reasoning about their relationships. When designing algorithms for RSB hierarchies, we can guarantee that all possible cases are handled by accounting for the distinct relationships *within* and *between* supercubes.

The supercube perspective of RSB hierarchies reinforces our analysis from Chapter 4 of the relative complexities of the simplex-based and diamond-based approaches. There are:

- $2^d \cdot (2^d - 1)$ distinct diamond types (also, unique edges and vertices),

- $2^d \cdot (2^d - 1) \cdot d$ distinct parents of diamonds (also, unique parent-child duets), and

- $2^d \cdot (2^d - 1) \cdot d!$ distinct RSB simplices of order d.

Thus, for each of the $2^d(2^d - 1)$ diamond types in the hierarchy, there are an average of $d!$ RSB simplices and an average of d parents.

Similarly, supercubes provide a memory-less means of iterating through the elements in an RSB hierarchy, leading to streaming algorithms for RSB hierarchies. As can be seen in the right column of Figure 5.3, the origins of supercubes form a hypercubic lattice that tiles each level of resolution within the hierarchy. Since it is trivial to iterate through a regular grid, all that remains is an efficient method of iterating through the elements of a given supercube. We provide this in Chapter 6 by considering the edge midpoints of a supercube as offsets from its origin. In contrast, previous approaches [ZCK97, BPSC04] process the entire hierarchy through a breadth-first traversal of the DAG, in which memory

is required for each element. Since the number of simplices double at each successive depth in the hierarchy, this can become prohibitive after a few levels of resolution within the hierarchy. We exploit these properties to generate the approximation errors of a multiresolution scalar field in Section 7.1.1, and in our GPU-based parallel framework for multiresolution terrain processing [YWD11].

Supercubes also provide a formal approach to processing collections of diamonds that are 'outside the cube,' that is, the base domain is no longer restricted to a hypercube of resolution $(2^N + 1)^d$. This is useful from a practical point of view, since many datasets of interest are not hypercubic, and embedding the dataset in its smallest enclosing hypercube can impose a significant amount of unused storage space. A similar type of decomposition is used by Maubach [Mau95] and by Tanaka et al. [Tan95,TTW03], where a rectangular (or cuboid-shaped) domain is initialized with Freudenthal's K_1 triangulation, and is converted into an RSB decomposition after one level of refinement (as in the arrows leading from K_1 to J_1 at the next level of resolution in Figure 5.3).

We explore the use of supercubes as a *clustering primitive* for coherent subsets of an RSB hierarchy in Chapter 7 to encode closed subsets of the vertices of a multiresolution scalar field as well as its extracted RSB meshes. This is useful in reducing the geometric and sampling overhead when representing scalar fields that are oversampled and for scalar fields in which values for some regions are not defined or are not available. This can also be useful in dynamic datasets in which we would like to be able to locally increase the resolution without requiring the entire field to have uniform high resolution.

Finally, the supercube can be viewed as an intermediate primitive between spatial decompositions based on nested hypercubes, such as 2^d-trees, and those based on nested RSB hierarchies. Compared to 2^d-trees, where each subdivision adds 2^d cubes to a mesh, RSB hierarchies provide significantly more adaptivity by spreading this growth over the course of d refinements. The 2^d children of a 2^d-tree node can be seen as corresponding to the 2^d 0-diamonds of a supercube σ covering the same domain as a 0-diamond one level higher in the hierarchy, while the intermediary i-diamonds in a supercube enable the increased adaptability achieved by RSB hierarchies compared to 2^d-trees. In Chapter 10, we exploit the connection between hypercubes and 0-diamonds to define a multiresolution model over hypercubes generated by regular refinement defining a dependency relation for balanced hypercubic refinement. We also propose a supercube-based pointerless encoding for nested hypercubic meshes extracted from a balanced 2^d-tree.

Chapter 6

Encoding diamond hierarchies

In this chapter, we introduce a novel interpretation of the binary representation of a diamond's central vertex, $\mathbf{v}_c$ in terms of three components: its scale, encoding its level of resolution in the hierarchy; its embedding supercube at this level of resolution; and its diamond type, encoding its *class* and position within its embedding supercube. This provides all the required information to compute the local mesh topology of a d-dimensional diamond, including the location of its vertices and simplices, as well as the central vertices of its parents and children in terms of scaled offsets from $\mathbf{v}_c$. This leads to the development of an efficient pointerless representation for d-dimensional hierarchies of diamonds as well as for simplicial complexes extracted from such hierarchies. This encoding is derived from the diamond decomposition of Chapter 4 as well as the analysis of supercubes from Chapter 5.

A key distinction needs to be made between a hierarchy of diamonds Δ, which is a *multiresolution* model over the domain induced by the diamond dependency relation and is encoded as a DAG of diamonds, and a conforming *variable resolution* RSB mesh Σ that we extract from Δ, which corresponds to a cut of the arcs of the DAG, separating the modifications that have been applied from those that have not.

In the following, we assume that a d-dimensional hierarchy of diamonds Δ covers a regularly sampled hypercubic domain Ω containing $(2^N + 1)^d$ samples, where N is the maximum level of resolution Level_{Max} and vertices of diamonds in Δ have integer coordinates in the range $[0, 2^N]$.

Due to the one-to-one correspondence between diamonds and grid points, a diamond $\delta \in \Delta$ is uniquely defined by its spine ψ, or alternatively, by its central vertex $\mathbf{v}_c$, the unique midpoint of ψ.

We first provide our implicit encoding for diamonds in Sections 6.1– 6.1.4, and illustrate this encoding through an example in Section 6.1.5. We then discuss how we encode supercubes and collections of supercubes in Section 6.2. In Section 6.3, we discuss simplex-based, diamond-based and supercube-based encodings for nested RSB meshes.

6.1 Encoding diamonds

The regularity of the vertex distribution in Δ and the subdivision operation, enables us to derive all geometric and hierarchical relationships of a diamond δ directly from the binary representation of the coordinates $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d)$ of its central vertex $\mathbf{v}_c$.

Let

$$\mathbf{v}_c = \begin{bmatrix} \mathbf{x}_1 & = & \underbrace{x_1^1 \, x_1^2 \ldots x_1^m} & \underbrace{\tau_1^1 \, \tau_1^2} & \underbrace{00 \ldots 0} \\ \mathbf{x}_2 & = & x_2^1 \, x_2^2 \ldots x_2^m & \tau_2^1 \, \tau_2^2 & 00 \ldots 0 \\ & & \vdots & & \\ \mathbf{x}_d & = & \underbrace{x_d^1 \, x_d^2 \ldots x_d^m}_{\mathbf{s}} & \underbrace{\tau_d^1 \, \tau_d^2}_{\boldsymbol{\tau}} & \underbrace{00 \ldots 0}_{\gamma} \end{bmatrix}^T \tag{6.1}$$

be the binary representation of $\mathbf{v}_c$. Our encoding depends on three quantities which can be efficiently extracted from the binary representation of the central vertex of a diamond through bit shifting operations: the *scale* γ, the *type* τ and the *supercube origin* $\mathbf{s}$ of δ.

6.1.1 Diamond scale

Let $\textsc{Trailing}(\mathbf{x}_i)$ denote the number of trailing zeros in the binary representation of a coordinate $\mathbf{x}_i$ of $\mathbf{v}_c$. Then, the minimum of the number of trailing zeros among each of the d coordinates of $\mathbf{v}_c$ encodes the *scale* γ of δ, e.g.

$$\gamma = \min_{i \leq d}(\textsc{Trailing}(\mathbf{x}_i)).$$

Then, for a diamond δ at scale γ, the rightmost γ bits in any coordinate of $\mathbf{v}_c$ are zero, but at least one of the bits in position $\gamma + 1$ (e.g. τ^2) is nonzero.

Recall from Section 4.4 that the *depth* of an i-diamond δ is the length of a path from the root diamond to δ in the dependency graph of Δ, i.e. the number of subdivisions required to obtain δ. The *level* of an i-diamond δ is the number of i-diamonds above δ along any path to the root diamond. Since the *class* of a diamond cycles with every d subdivisions, its depth is thus $\textsc{Level}(\delta) * d + i$.

The level and scale are related through $\textsc{Level}_{Max}$ as

$$\textsc{Level}(\delta) = \textsc{Level}_{Max} - \gamma.$$

6.1.2 Diamond type

The two bits at position $\gamma + 1$ and $\gamma + 2$ of each coordinate $\mathbf{x}_i$, which we denote as τ_i^2 and τ_i^1, respectively, uniquely encode the *type* τ of δ. Since vertices in Δ have d coordinates, and τ has two bits for each coordinate, there are $(2^2)^d = 4^d$ possible values for τ. However, the definition of γ precludes the 2^d cases where all bits of τ^2 are zero. This gives us the $4^d - 2^d$ diamond types of Section 5.2.

The similarity *class* of δ is encoded within τ as the number of zeros in position τ^2, e.g. an i-diamond has $(d - i)$ nonzero bits at this position, and there are $\binom{d}{i}$ variations in this encoding. Since there is no similar restriction on the bits in position τ^1, we have 2^d possibilities for τ^1, corresponding to the 2^d subcubes within its embedding supercube, for the total $\binom{d}{i}2^d$ distinct *types* of i-diamonds.

We can now interpret the diamond types $4^d - 2^d$ in terms of midpoints of edges of the supercube as defining a local grid of 4^d points, where each of the d coordinates

indexes into this grid using two bits (with four values). Furthermore, the 2^d vertices of the supercube (i.e. where all τ^2 bits are zero) are not valid midpoints of edges on this grid (see Figure 6.1).

The oriented *direction* of δ's spine ψ can be computed from τ using the following encoding. First, initialize a sign variable sgn to $+1$. Component u_i of the direction vector $\vec{u}$ is then:

$$u_i = \begin{cases} 1, & \text{if } \tau_i^1 = 0 \text{ and } \tau_i^2 = 1 \\ -1, & \text{if } \tau_i^1 = 1 \text{ and } \tau_i^2 = 1 \\ 0, & \text{if } \tau_i^1 = 0 \text{ and } \tau_i^2 = 0 \\ 0, & \text{if } \tau_i^1 = 1 \text{ and } \tau_i^2 = 0. \end{cases} \qquad (6.2)$$

In the fourth case, where $\tau_i^1 = 1$ and $\tau_i^2 = 0$, we multiply the sign sgn by -1. This global negation corresponds to a reflection across the median plane of **s** corresponding to coordinate axis i.

The orientation of spine ψ is then:

$$\text{orient}(\psi) = \text{sgn} * \vec{u}. \qquad (6.3)$$

Note that Equation 6.3 maps diamond types into axis aligned vectors with $3^d - 1$ possible values ($\vec{u}_i \in \{-1, 0, +1\}$, but $\vec{u} \neq 0^d$), and relates our diamond type to the oriented spine direction of Gregorski et al. [GDL$^+$02].

6.1.3 Supercube origin

The final $m = N - (\gamma + 2)$ bits in each of the coordinates of $\mathbf{v}_c$ encode the *origin* of the supercube **s** containing δ. Since there are no restrictions on the values of these bits, the origins of supercubes at a fixed level of resolution $\ell = m + 2$ are points on a regular d-dimensional grid that has been scaled by a factor of $2^{\gamma+2}$.

The encoding of Equation (6.1) provides an alternate interpretation for the diamond type τ as the scaled offset of its central vertex $\mathbf{v}_c$ from its containing supercube's origin (see Figure 6.1), i.e.

$$\mathbf{v}_c = \mathbf{s} \cdot 2^{\gamma+2} + \tau \cdot 2^{\gamma}. \qquad (6.4)$$

Since the type τ of a diamond δ is determined by the supercube edge with which its spine coincides, each supercube can be seen as containing a single copy of each unique type of diamond within the hierarchy.

6.1.4 Diamond components

The geometric and hierarchical *components* of a diamond δ such as the location of its vertices and of the central vertex of its parent and children diamonds can be calculated as scaled offsets from its central vertex $\mathbf{v}_c$. These offsets are scaled d-dimensional vectors $\vec{g} = 2^{\gamma} \cdot \vec{f}$ such that $f_i \in \{-1, 0, 1\}$ and γ is the scale of δ. Specifically, a component of diamond δ, at scale γ, whose center is **p** and whose unscaled offset from $\mathbf{v}_c$ is $\vec{f}$ can be computed as:

$$\mathbf{p} = \mathbf{v}_c + 2^{\gamma} \cdot \vec{f}. \qquad (6.5)$$

79
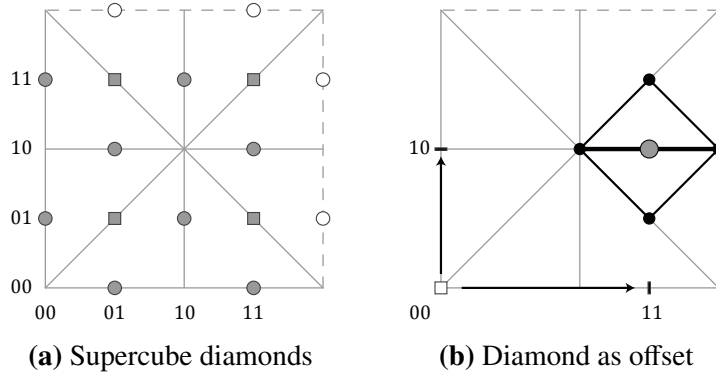
Figure 6.1: A diamond's spine coincides with an edge (solid lines) of a single supercube (a). Alternatively, a diamond's central vertex lies at the midpoint of a supercube edge (b). The midpoints of edges of a supercube implicitly define a local grid of 4^d locations (indexed by two bits), of which its 2^d vertices are not the midpoints of edges.

These offsets can be derived directly from the *type* τ and the *scale* σ of δ using the diamond decomposition from Theorem 4.5.1 or can be precomputed and accessed at runtime from a lookup table.

6.1.4.1 Kuhn-subdivided component

Since the class i of δ is encoded in the number of zeros in the rightmost bits of τ, the subspace of $\mathbb{R}^d$ spanned by the Kuhn subdivided $(d-i)$-cube $\mathcal{K}(h_k)$ is defined by the $(d-i)$ coordinates of τ^2 with value $\tau_j^2 = 1$.

Specifically, δ's spine $\psi = (\mathbf{v}_0, \mathbf{v}_k)$ can be calculated using the oriented spine direction $\vec{u}$. That is, $\mathbf{v}_0$ has offset vector $\vec{f} = -\vec{u}$ and $\mathbf{v}_k$ has offset vector $\vec{f} = \vec{u}$. Their coordinates can be obtained by plugging $\vec{f}$ into Equation 6.5. The remaining vertices and cells can be found through an affine mapping to the canonical subdivision of Section 4.2.1, or, if d is reasonably small, through precomputed lookup tables.

Since the $2 \cdot (d-i)$ children of δ have central vertices that coincide with the facets of h_k (for $i < (d-1)$), offsets from $\mathbf{v}_c$ to their central vertices can be computed as $\vec{f} = \pm e_j$, in all coordinates that $\tau_j^2 = 1$. When $i = (d-1)$, the 2^d children of δ are at offsets $\vec{f} = (\pm 1, \pm 1, \ldots, \pm 1)$, and at scale $\gamma - 1$.

6.1.4.2 Fully-subdivided component

The i-dimensional subspace spanned by $\mathcal{B}_F(h_i)$ is along the coordinates in which $\tau_j^2 = 0$. Consequently, for $i > 0$, offsets to the parents of δ are $\vec{f} = \pm e_j$, in all coordinates that $\tau_j^2 = 0$. For $i = 0$, we can use the oriented spine direction to find vertices v_0 and v_d of the spine. Then, as in Section 4.5, $\mathbf{v} = v_d - v_0$ and the central vertices of a diamond's d parents are located at offset $\vec{f} = -u + \mathbf{v} \cdot e_j$.

Since the vertices and d-simplices of $\mathcal{B}_F(h_i)$ are defined along all directions spanned by h_i, they can be found by incrementally traversing in a direction within h_i orthogonal to

the directions that have already been traversed. That is, since the ancestor of a diamond at the center of an i-cube of $\mathcal{B}_F(h_i)$ is the center of one of its facets, the traversal is only along a single dimension.

6.1.5 Example

Consider the two dimensional diamond δ whose central vertex $\mathbf{v}_c$ has coordinates $(72, 20)$. Using Equation (6.1) on the binary representation of $\mathbf{v}_c$, we can determine its scale γ, its type τ, and its supercube origin $\mathbf{s}$ as follows:

$$\mathbf{v}_c = \begin{bmatrix} 72 \\ 20 \end{bmatrix} = \begin{bmatrix} 100\,10\,00_2 \\ 001\,01\,00_2 \end{bmatrix} = \begin{bmatrix} \underbrace{100}_{\mathbf{s}} & \underbrace{10}_{\tau} & \underbrace{00}_{\gamma} \\ 001 & 01 & 00 \end{bmatrix}.$$

Thus, $\gamma = 2$, $\tau = (10_2, 01_2) = (2, 1)$ and $\mathbf{s} = (100_2, 001_2) = (4, 1)$. Since only one of the bits in τ^2 is zero, δ is a 1-diamond. Also, we can scale $\mathbf{s}$ to obtain the location of its supercube's origin within the grid as $(4, 1) \cdot 2^4 = (64, 16)$

The vertices of δ's spine $\psi = (\mathbf{v}_0, \mathbf{v}_k)$ can be determined using Equation (6.2) as $\vec{u} = (0, -1)$. This vector points down the y-axis, thus, we can find the coordinates of its spine vertices $\mathbf{v}_0$ and $\mathbf{v}_k$ by subtracting, and adding, respectively, the scaled spine orientation vector $\vec{u}$ to $\mathbf{v}_c$:

$$\mathbf{v}_0 = \mathbf{v}_c - 2^\gamma \cdot \vec{u} = (72, 20) - 4 \cdot (0, -1)) = (72, 24).$$

By adding the scaled offset $\vec{u}$ to $\mathbf{v}_c$, we obtain, $\mathbf{v}_k = (72, 16)$.

Since δ is a 1-diamond, both its Kuhn and fully-subdivided components are 1-dimensional. From the rightmost bits of τ, we can determine that its Kuhn component h_k is aligned with the y axis and its fully-subdivided component is aligned with the x axis (see Figure 6.2). Thus, for example, we can use Equation (6.5) to find the central vertex of its parent δ_p (in the negative x direction) at unscaled offset $\vec{f} = (-1, 0)$. We first scale $\vec{f}$ by 2^γ and then add the result to $\mathbf{v}_c$, so

$$\mathbf{v}_c(\delta_p) = (72, 20) + 2^2 \times (-1, 0) = (68, 20).$$

Figure 4.16 illustrates the components of all diamond classes up to dimension $d = 4$. Each cell corresponds to a diamond δ of class i (columns) in dimension d (rows). The top hypercube in each cell is a Kuhn subdivided $(d - i)$-cube containing the oriented spine, whose vertices are colored black and gray, respectively, and the children, whose central vertices (blue) are located at the center of its facets, at an offset $\vec{f} = \pm e_j$, in all coordinates that $\tau_j^2 = 1$. The bottom hypercube in each cell is the boundary of a fully subdivided i-cube. The central vertices of the parents (red) of δ are located at the center of its facets at offset $\vec{f} = \pm e_j$, in all coordinates that $\tau_j^2 = 0$. The two hypercubes intersect at their midpoints.
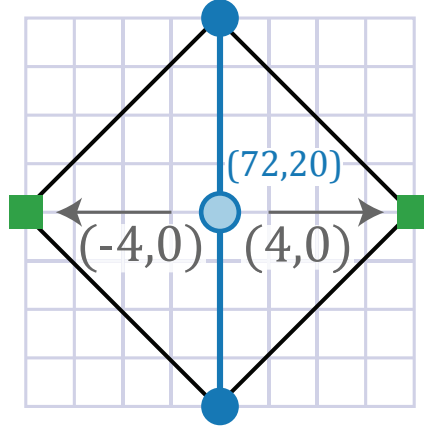
Figure 6.2: The vertices, parents and children of a diamond δ are located at scaled offsets from its central vertex. These offsets are computed from the type τ and scale γ of δ. Here, the parents of a diamond with central vertex (72,20) are located four units away on the x-axis.

6.1.6 Domain corners

Although they are not diamonds, we can apply Equation (6.1) to the 2^d corner vertices of the domain boundary. The lower left corner maps to a supercube with coordinates 0^d whose scale is determined by the number of bits used by the machine to represent coordinates (e.g. 32 for 4 byte integers). The remaining $2^d - 1$ corners map to a supercube at location $(0, 0)$ at a scale of $\text{LEVEL}_{Max} + 1$.

6.2 Encoding supercubes

Since one of the primary applications of supercubes is to efficiently encode information about a subset of the entities of an RSB hierarchy, supercubes require some form of bookkeeping to index their encoded entities.

For example, when encoding a supercube's diamonds, edges or vertices (edge midpoints), a simple approach would be to encode this data as an array with one entry for each of the $4^d - 2^d$ possible entities in the supercube. The data associated with each entity can then be indexed by its type τ, and unencoded entities can be marked in place as missing. However, this can waste a considerable amount of storage for supercubes with many unencoded entities. We refer to this approach as *array-based supercube indexing*.

On the other extreme, if we expect a low occupancy rate, we can use a map data structure (a tree or a hash map) to index the encoded entities. This requires $\lg(\lceil 4^d - 2^d \rceil) \approx 2 \cdot d$ bits of indexing data per encoded entity. We refer to this approach as *map-based supercube indexing*.

A more efficient approach for static representations in relatively low dimensional domains, in which we expect a *moderate* number of entities to be encoded can be obtained by indexing the encoded entities using a bitflag of $4^d - 2^d$ bits (e.g. 2 bytes in 2D, 7 bytes in 3D) along with a corresponding array with storage only for the encoded entities. The data associated with a entities with associated type τ can then be indexed in the array

by the *prefix sum* of τ's position in the bitflag, i.e. since each set bit for positions before τ in the bitflag correspond to occupied locations, and each unset bit corresponds to an unoccupied location, the position of τ's data is in the position corresponding to its prefix sum in the bitflag. Since prefix sum computations can be performed efficiently in hardware, the processing overhead of this representation compared to that of the simpler encoding above is negligible. Each supercube in this representation incurs a fixed storage overhead, regardless of the number of entities encoded. Thus, the overhead of this representation is reduced as the average number of encoded entities per supercube increases. We refer to this approach as *bitflag-based supercube indexing*.

Since this bitflag representation requires a reorganization of the array data every time a diamond is added or removed, we typically use the array representation during the dynamic operations (such as initial generation), and convert to the bitflag representation when the values become static.

6.2.1 Encoding collections of supercubes

We observe that within a given level of resolution ℓ, supercubes can be uniquely indexed by their origin. However, supercubes from different levels can map to the same origin (see Figure 5.4).

We propose a two step access structure, where supercubes are first indexed by their level of resolution, and then by their origin. This also enables the level ℓ of a supercube to be implicitly encoded within the access structure.

The encoded supercubes at a given level of resolution ℓ belong to a uniform grid that has been scaled by a factor of $2^{\gamma+2}$. Thus, depending on the data distribution, we have several options for access structures to the supercubes. When the majority of the data within a given level is present, the supercubes can be indexed using a full array. However, most of the time, this will not be the case, and we can organize the supercubes into an auxiliary data structure, such as a B-tree (for efficient out-of-core access) or a hash table (for $O(1)$ in-core access).

Thus, the data associated with a diamond δ (or its spine ψ or midpoint $\mathbf{v}_c$) with supercube origin $\mathbf{s}$, type τ and scale γ can be accessed in three steps. First, the set of supercubes at level $\ell = N - (\gamma + 2)$ is located. Next, the supercube $\mathbf{s}$ within this set is found. Finally, the data at location τ within $\mathbf{s}$ is found using the internal map of $\mathbf{s}$.

6.3 Encoding RSB meshes

Recall from Section 4.6 that variable-resolution simplicial meshes (not necessarily conforming) are extracted from an RSB hierarchy using an adaptive refinement process, while variable-resolution simplicial complexes are extracted from an RSB hierarchy in a selective refinement process. The former corresponds to a closed set of refinements with respect to the simplex bisection operation, while the latter corresponds to a closed set of refinements with respect to the diamond dependency relation. In either event, the current RSB mesh Σ corresponds to the active front of the query that is closed with respect to the given relation.

To ensure that extracted meshes do not contain cracks, selective refinement requires all of a diamond δ's simplices to be present in the current mesh before δ can subdivide, i.e. *delta* must be *complete*. Although, in the worst case, this requirement can trigger subdivisions recursively up to the root of the hierarchy, the cost of such non-local subdivisions is amortized over the set of elements in the local neighborhood. Consequently, an efficient implementation of selective refinement requires fast access to the simplices or diamonds in the active front as well as their ancestors and descendants.

Below, we first describe the pointerless simplex-based representations for RSB meshes. We then introduce our pointerless diamond-based approach, and finally, our pointerless supercube-based approach for diamond meshes.

6.3.1 Simplex-based representation

Pointerless encodings for a simplex-based simplicial complex Σ_σ extracted from a forest of simplices typically use location codes to index the encoded simplices, but vary in their adaptation of 2^d-tree location codes [Gar82a, Gar82b, Sch92, Sam06] to nested RSB meshes.

The *bintree location code* [EKT01, LDS01, LDS04] for a simplex σ at depth m is encoded as the index $i_p \in [0, d!)$ of the bintree containing σ followed by an m-bit binary string indicating the traversal path from the root of bintree i_p to σ. Each such simplex can be indexed using $(m + \lceil \lg(d!) \rceil)$ bits. Note that, in this encoding, the vertices of a simplex are not stored, and can be determined from those of its bintree root in $O(m)$ time by descending the tree.

Alternatively, the *subtree location codes* [Heb98, Heb94, Pas00, AM07] are based on the simplex encoding scheme for supercubes outlined in Section 5.2 (see Figure 5.7). A simplex σ at depth $m = d \cdot \ell + i$ requires an encoding for its *reflection number* $i_r \in [0, 2^d)$, its *permutation number* $i_p \in [0, d!)$ and its *descendant number* $i_d \in [0, 2^d - 1)$, in addition to the path in the 2^d-tree to its containing supercube, requiring $\ell_\sigma = O(d \cdot \ell)$ bits. The *subtree location code* for an RSB simplex σ at level ℓ can be encoded using $(d \cdot \ell + d \cdot 2 + \lceil \lg(d!) \rceil)$ bits per simplex in the mesh,

By substituting the level ℓ and class i for the depth m of the simplex, i.e. $m = d \cdot \ell + i$, we observe that both types of location codes have the same complexity $O(d \cdot \ell + \lceil \lg(d!) \rceil)$. Furthermore, by substituting the maximum level of resolution N for ℓ, and using the identity $\lg(n!) = \Theta(n \lg n)$ [CLRS01], we can determine the space complexity of simplex-based location codes to be $O(dN + d \lg d)$ bits.

Additional information can also be required for efficient (i.e. constant time) neighbor finding. For example, in [LDS01, LDS04], an additional 1 byte *neighbor mask* is required to efficiently determine the appropriate bisection-edge neighbors of a simplex.

An advantage of *subtree location codes* over *bintree location codes* is that the vertices of a simplex can be directly computed without requiring a tree traversal [Heb94, AM07].

Besides the storage savings achieved by diamond-based encodings, the simplex-based encodings cannot efficiently cache the status of its subdivided neighbors, and thus each simplex bisection necessarily requires $O(d!)$ neighbor-finding operations before its $O(d!)$ bisections.

6.3.2 Diamond-based representation

The encoding presented in Section 6.1 leads to an efficient pointerless representation for a diamond-based simplicial complex Σ_d extracted from a hierarchy of diamonds. Σ_d can be encoded as a collection of diamonds, each of which contains a set of d-simplices, such that, the collection of simplices from all diamonds in Σ_d forms a simplicial complex covering the domain Ω. Since we can reconstruct the location of all vertices, simplices, parents and children of a diamond δ from the coordinates of its central vertex, each diamond can be entirely indexed by the d coordinates of its central vertex. Thus, encoding the coordinates of a diamond's central vertex in a hierarchy with maximum level of resolution N, requires $d \cdot N$ bits.

In general, not all d-simplices of a diamond will belong to the complex Σ (see Figure 6.3a, where, e.g., the blue diamonds contain only one of their two triangles). Thus, each diamond δ requires some bookkeeping to track the set of its d-simplices belonging to the complex Σ_d. However, as observed in Section 4.4.3, a diamond's d-simplices are generated concurrently during the during the subdivision of a single parent of δ as a parent-child duet. Thus, a single bit is sufficient to track the presence or absence of each parent-child duet within δ and, consequently, $2 \cdot d$ bits are sufficient to track all $O(d!)$ d-simplices of diamond $\delta \in \Sigma_d$.

An advantage of the correspondence between parents and duets is that, when a diamond contains all of its duets, it is complete, and can refine. Thus, the $O(d)$ bookkeeping bits of a diamond simultaneously cache the subdivision status of each parent of δ. This can be used to further accelerate diamond refinement since it can reduce the number of required spatial accesses.

As a consequence, the cost of encoding diamonds in a diamond-based simplicial complex scales linearly with respect to the dimension, even though the number of simplices scales factorially with respect to the dimension.

The cost of encoding a diamond in this representation is: $(d \cdot N + d \cdot 2)$ bits per diamond in the mesh.

A straightforward representation for encoding an active front utilizes a hash table of diamonds. Each diamond δ is indexed by its central vertex $\mathbf{v}_c$, and contains a set of bitflags tracking its subdivided parents as well as any additional information that must be encoded for the diamond.

In 2D, if each coordinate can be encoded in 15 bits, diamonds can be encoded using 4 bytes of overhead. In 3D, this representation requires 7 bytes of overhead for each encoded diamond: 6 bytes for the coordinates of its central vertex and one additional byte of bookkeeping.

6.3.3 Supercube-based representation

We can encode an RSB mesh Σ_s generated by selective refinement in terms of supercubes as well. There is a considerable amount of coherence among the simplices in Σ that a simplex-based or diamond-based representation of an active front cannot exploit. Due to the RSB rule, neighboring simplices in Σ can differ by at most one level of refinement, so the presence in Σ of a simplex σ from diamond δ often indicates the presence of simplices
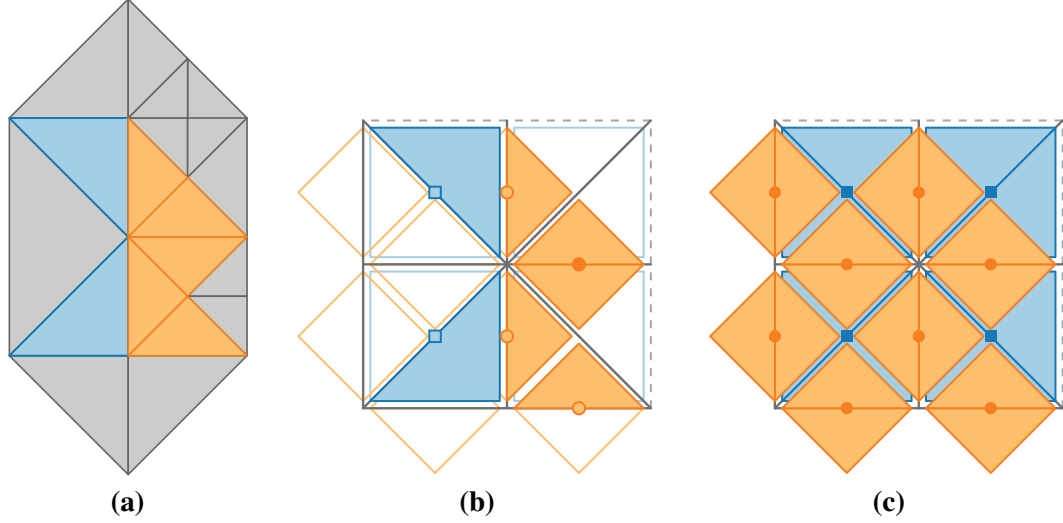
Figure 6.3: (a) A portion of a conforming RSB mesh Σ (in 2D). (b) Highlighted triangles from Σ map to triangles in the supercube **s**, while gray triangles map to other supercubes. (c) The set of all triangles (tetrahedra in 3D) in a supercube overlap.

from neighboring diamonds in the hierarchy. We therefore propose a supercube-based representation Σ_s for simplicial complexes extracted from a hierarchy of diamonds.

Since we are only considering the cost of encoding the presence or absence of simplices in a conforming mesh, our encoding can use a bitflag-based supercube representation on the $2^d(2^d - 1)d$ supercube parents. In addition, we require $d \cdot N$ bits to encode the origin of each supercube. Thus the cost of this representation is: $d \cdot N + 2^d(2^d - 1)d$ bits per supercube in the active front.

Assuming that each coordinate requires 2 bytes, then in 2D, each supercube in Σ_s requires 7 bytes: 3 bytes to represent the bitflags (e.g. 1 bit for each of the 24 possible duets in the supercube) in addition to the 4 bytes for the coordinates of its origin.

Alternatively, in 3D, each supercube in Σ requires 27 bytes: 21 bytes to represent the bitflags (e.g. 1 bit for each of the 168 possible duets in the supercube) in addition to the 6 bytes for the coordinates of its origin.

Note, however, that when considered collectively, the duets in a supercube overlap (see Figure 6.3c for an illustration in 2D), but simplices in a simplicial complex cannot overlap. Due to the containment relation among the simplices in the hierarchy, the presence of a simplex σ in Σ precludes the presence of its parent simplex as well as both of its children simplices from Σ. Thus, in practice, a supercube in Σ contains significantly fewer than the $2^d(2^d - 1)d$ possible duets (see Figure 6.3b, where the 2D supercube contains 7 of the 24 possible triangles).

Chapter 7

Diamond-based multiresolution scalar fields

One of the primary applications of diamond hierarchies has been as a multiresolution model for scalar fields such as terrain and volumetric datasets defined at the vertices of a regularly sampled rectilinear grid. We call this model a ***Diamond-based Multiresolution Sscalar Field (DMSF)***.

7.1 DMSF Model

A hierarchy of diamonds is the basis for a multiresolution model of a scalar field. In the case of a DMSF, the base mesh is a coarse RSB mesh and the modifications correspond to the diamond subdivisions. Since each diamond has a one-to-one correspondence with its central vertex, the vertices are ordered according to the dependency relation of a hierarchy of diamonds Δ. Thus, the spatial decomposition and dependency relation of a DMSF are obtained from the implicit relationships among the diamonds in Δ, and only the modifications need to be explicitly encoded.

The minimal information encoded in a diamond δ is given by the scalar value $F(\mathbf{v}_c)$, associated with the central vertex $\mathbf{v}_c$ of δ. In addition to encoding $F(\mathbf{v}_c)$, it is often useful to encode aggregate information about the field values within the domain of δ, which can be used to accelerate mesh extraction. This typically includes an approximation error for the diamond to guide the refinement, as well as the range of values within the domain of δ. which can be used to cull irrelevant regions during field-based queries [WVG92]. The field gradient can also be maintained to accelerate visualization and analysis of the dataset. All such information is associated with a diamond, or its central vertex, and thus a DMSF can be efficiently encoded as a d-dimensional array indexed by $\mathbf{v}_c$.

7.1.1 Generating a DMSF

To generate the model, we need to find the approximation error for each diamond $\delta \in \Delta$.

The error $\epsilon(\delta)$ associated with diamond δ encodes the maximum approximation error for any point within the domain of δ, i.e.,

$$\epsilon(\delta) = \underset{p \in \delta}{Max}\,(\epsilon(p)), \tag{7.1}$$

where $\epsilon(p) = \left| F(p) - \hat{F}(p) \right|$ is the absolute difference between the field value at point p and the approximated value obtained through barycentric interpolation of the field values at the vertices of δ.

The points of a diamond δ can be efficiently enumerated using a recursive scheme based on [MDM04], where, in 3D, each tetrahedron is split into triangular slices that are

aligned with a coordinate plane, and each triangle is in turn sliced into axis aligned line segments.

The final component in generating a DMSF relates to enumerating the diamonds. A straightforward approach is to perform a top-down breath first enumeration of the DAG. However, this approach requires an amount of memory proportional to the number of diamonds at the current level. Since this roughly doubles with each increasing subdivision level, this method can exhaust the memory store or cause thrashing when processing deeper levels of the hierarchy.

We use supercubes as an algorithmic primitive to directly enumerate all diamonds on a level by level basis. Since the origins of supercubes at level ℓ of the hierarchy coincide with the vertices of a scaled regular grid, these points can be easily generated without incurring any memory overhead. Similarly, the diamonds within each supercube can be enumerated using successive diamond types. Thus, this approach can be run on the levels of the hierarchy in any order, i.e. top-down manner from the root of Δ, bottom-up from the leaves of Δ, or in an arbitrary order. As such, this method admits highly parallel implementations for processing each level of the hierarchy, as well as individual supercubes, diamonds or simplices within those supercubes.

7.2 Full DMSF

A *full DMSF*, which we denote as Δ_f, contains diamonds corresponding to all vertices of a scalar field of resolution $(2^N + 1)^d$. The base mesh of Δ_f is a single 0-diamond δ_0 covering the entire hypercubic domain Ω. The 2^d corner points of Ω (i.e. the vertices of δ_0) are the only points within Δ_f that do not correspond to diamonds.

A full DMSF Δ_f can be encoded as a d-dimensional array whose entries represent the information associated with each diamond and can be indexed using a C-style row major ordering, or a more complicated indexing scheme such as a hierarchical space-filling curve [PF01, GDL⁺02].

Alternatively, Δ_f can be encoded by a supercube-based representation without incurring any storage overhead. Since all vertices of Δ_f are present, the internal map within each supercube can be encoded in an array with $4^d - 2^d$ elements. Furthermore, since all supercubes at each level of resolution are present, each level can be encoded as an array of supercubes, indexed by their origin. The advantage of this representation is that diamonds are clustered near their spatial and hierarchical neighbors. Thus, these coherent diamonds can be loaded in the cache at the same time.

7.3 Partial DMSF

However, when some of the vertices of a full DMSF Δ_f are unavailable or irrelevant for an intended application, a *partial DMSF*, which we denote as Δ_p, can be much more efficient to encode than Δ_f.

The base mesh of a partial DMSF Δ_p is a coarse RSB mesh consisting of diamonds from a corresponding hierarchy of diamonds Δ, whose vertices are tagged with values from the scalar field F. The diamonds in Δ_p are a subset of the diamonds of Δ_f subject to

the *transitive closure* constraint that if a diamond δ belongs to Δ_p then all ancestors of δ belong to Δ_p as well. Finally, the dependency relation of Δ_p is the dependency relation of Δ restricted to the diamonds in Δ_p.

Such sparse representations are important, for example, when not all the data in a volume data set are available and instead of having a full grid, we have the data points at a subset of the vertices of the domain, or when the portion of data of interest is small compared to the full dataset, as, for instance, when only certain portions of the dataset are available at a higher resolution. Furthermore, when the dataset is locally oversampled, e.g., samples covering a large body of water in a terrain dataset, we can accurately interpolate these values from samples at a higher resolution. Figure 7.1 shows a zero approximation error sparse representation of a 6000×4800 sample tile from the gtopo30 dataset [Sur]. For this dataset, a partial DMSF requires less than 1/6 of the original samples since flat regions do not need to be subdivided to the highest resolution to obtain an accurate approximation.

(a) GTOPO30 Tile   **(b)** Partial DMSF
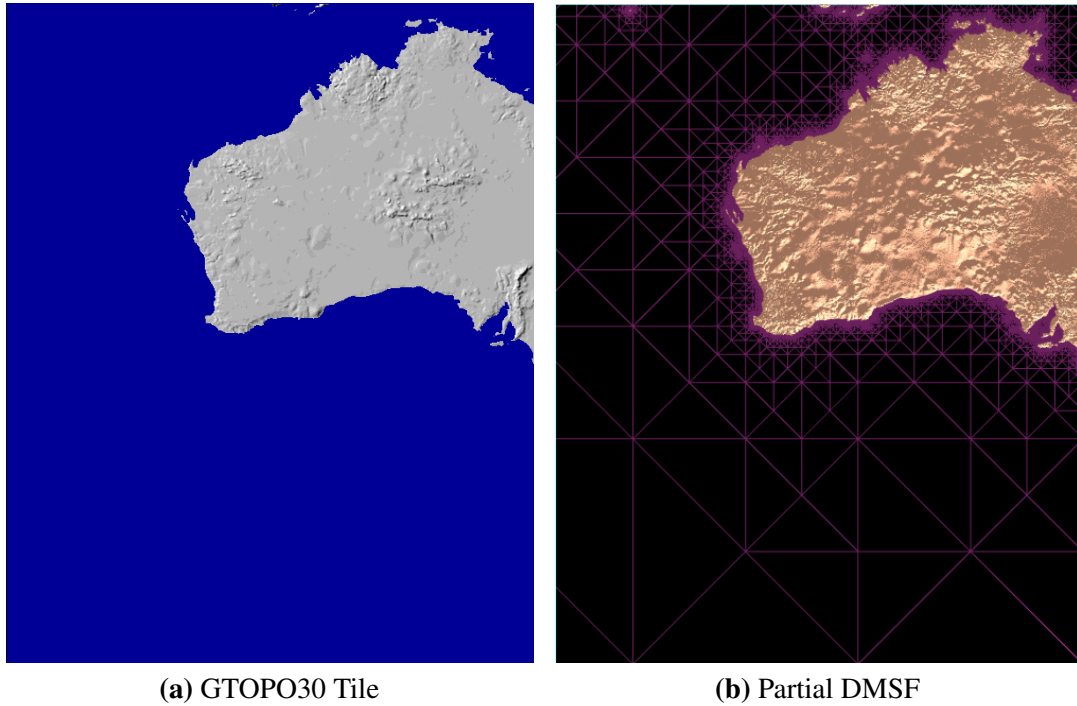
Figure 7.1: Terrains covering large flat regions such as oceans are oversampled by a regular grid (a). A zero error sparse representation of this terrain (b) requires less than 1/6 of the samples from the original dataset. Image (a) courtesy of USGS [Sur].

The main challenge in representing such sparse datasets relates to the efficient encoding of the coordinates of its diamonds. Whereas the coordinates of diamonds in a full DMSF Δ_f can be implicitly determined e.g. by representing Δ_f as an array, such a representation is impractical for a partial hierarchy Δ_p where much of the data is non-existent or redundant.

A straightforward representation for Δ_p is to explicitly encode the d coordinates of each diamond's central vertex in addition to the scalar field data. We denote such a *diamond-based partial DMSF* as Δ_d.

However, due to the transitive closure constraint of the partial DMSF model, the encoded diamonds exhibit both a spatial and a hierarchical coherence which can be exploited by clustering the diamonds into supercubes. We denote such a *supercube-based partial DMSF* as Δ_s. Since Δ_p is static, and typically sparse with respect to a corresponding full DMSF Δ_f we can represent the internal map within supercubes using the *bitflag-based* encoding of Section 6.2. The supercubes at each level are indexed by the coordinates of their origin.

7.4 Theoretical evaluation

In this section, we consider when it is appropriate to represent a DMSF using a partial representation. We measure this in terms of the *density* of the dataset, i.e. the percentage of samples from a full DMSF that are retained in the partial representation. Next, we analyze when supercube-based representations of a partial DMSF are appropriate. For this, we consider the *concentration* of the clustering, that is, the average number of diamonds encoded per supercube. A supercube-based representation for a partial hierarchy of diamonds provides the maximum benefit when the desired dataset is sparse with respect to the full dataset and concentrated with respect to the supercube clustering.

We begin by introducing some notation. Let Δ_f denote the full DMSF, containing $n_f = (2^N + 1)^d$ diamonds and let Δ_p denote the desired partial DMSF, containing n_p diamonds. Δ_p can be encoded using a diamond-based partial DMSF Δ_d or a supercube-based representation Δ_s, whose n_p diamonds are clustered into n_s supercubes. Finally, let b_δ denote the number of bytes required to encode the data associated with each diamond, b_v the number of bytes required to encode the coordinates of the central vertex of each diamond and b_s the number of bytes required to encode the indexing and bookkeeping information associated with each supercube.

We compare the costs of these representations in Table 7.1 with respect to an ideal representation Δ_p, which only represents the n_p diamonds. This representation is not practical since it has no way of indexing the encoded diamonds, but we use it to compare the remaining representations. Δ_f must encode all n_f samples but the indexing of its elements is implicit. However, it encodes $n_f - n_p$ extraneous diamonds. In contrast, Δ_d encodes only the n_p diamonds but must also explicitly encode the spatial coordinates of each diamond. Finally, the overhead in Δ_s can be attributed entirely to the n_s supercubes.

Using this notation, we define the *density* $\mathbf{D} = n_p/n_f$ of the dataset as the ratio of retained diamonds in Δ_p compared to Δ_f. Also, we define the *concentration* $\mathbf{C} = n_p/n_s$ of the dataset as the average number of diamonds per supercube. We note that $\mathbf{C} \in [1, 4^d - 2^d]$ since we only encode supercubes that contain at least one diamond.

By rearranging the equations in Table 7.1 and substituting terms for $\mathbf{D}$ and $\mathbf{C}$, we can compare the representations.

The supercube-based partial DMSF Δ_s is more compact than the full DMSF Δ_f when

$$\mathbf{D} < \frac{b_\delta}{b_\delta + (b_s/\mathbf{C})}. \tag{7.2}$$

Table 7.1: Storage requirements and overhead, in bytes, for the full DMSF Δ_f containing n_f samples and the partial DMSFs Δ_p containing n_p samples – the diamond-based Δ_d and the supercube-based Δ_s. Costs and overhead are with respect to the number of bytes required to encode the coordinates of vertices (b_v), the data associated with each sample (b_δ), and for indexing and book-keeping of supercubes (b_s). Overhead is relative to the theoretically optimal Δ_p.

DMSF Representation	Storage cost	Overhead
Δ_p	$n_p \cdot b_\delta$	0
Δ_f	$n_f \cdot b_\delta$	$(n_f - n_p) \cdot b_\delta$
Δ_d	$n_p \cdot b_\delta + n_p \cdot b_v$	$n_p \cdot b_v$
Δ_s	$n_p \cdot b_\delta + n_s \cdot b_s$	$n_s \cdot b_s$

Table 7.2: Comparison between a full DMSF Δ_f and a supercube-based partial DMSF Δ_s in terms of density $\mathbf{D} = n_p/n_\delta$ and supercube concentration $\mathbf{C} = n_d/n_s$ for our terrain DMSF model, where $b_\delta = 4$, $b_v = 4$ and $b_s = 6$. Values indicate maximum density for which the cost of Δ_s is less than that of Δ_f.

$\mathbf{C}$	1	2	3	4	5	6	7	8	9	10	11	12
$\frac{4}{(4+6/\mathbf{C})}$	40%	57%	67%	73%	77%	80%	82%	84%	86%	87%	88%	89%

The diamond-based partial DMSF Δ_d is more compact than the full DMSF Δ_f when

$$\mathbf{D} < \frac{b_\delta}{b_\delta + b_v}. \tag{7.3}$$

Finally, the supercube-based partial DMSF Δ_s is more compact than the diamond-based partial DMSF Δ_d when

$$\mathbf{C} > b_s/b_v. \tag{7.4}$$

However, since all representations must encode the n_p diamonds, a more relevant measure of the effectiveness of each representation is related to its overhead with respect to Δ_p (third column of Table 7.1). While Δ_d has a constant overhead of b_v bytes per diamond, the overhead in Δ_s is related to $\mathbf{C}$ as ($b_s/\mathbf{C}$) bytes per diamond.

As a first example, consider a terrain dataset where each elevation is encoded using 2 bytes and the approximation error is also encoded using 2 bytes. Then $b_\delta = 4$ bytes. Also, assume that coordinates are encoded using unsigned shorts, so $b_v = 4$ bytes. Finally, let $b_s = 6$ bytes consisting of: the origin of the supercube (4 bytes) and 2 bytes for the 12 bitflags to indicate encoded diamonds. Then, in terms of density and concentration, Δ_s is more compact than Δ_f and Δ_d, respectively, when $\mathbf{D} < \frac{4}{(4+6/\mathbf{C})}$, and when $\mathbf{C} > 6/4 = 1.5$. Table 7.2 lists the maximum density $\mathbf{D}$ at which Δ_s is more compact than Δ_f for integer values of $\mathbf{C}$.

For three-dimensional volumetric datasets, let the size of each refinement be $b_\delta = 4$ bytes as in [GDL⁺02]. Further, assume vertices are encoded in 6 bytes as three unsigned shorts, then $b_v = 6$ bytes. Finally, let $b_s = 17$ bytes consisting of: the origin of the supercube (6 bytes), bitflags to indicate the encoded diamonds (7 bytes) and a pointer to

an array containing the data (4 bytes). Then, in terms of density and concentration, Δ_s is more compact than Δ_f and Δ_d, respectively, when $\mathbf{D} < \frac{4}{(4+17/\mathbf{C})}$, and when $\mathbf{C} > 17/6$. The curves in Figure 7.2 separate the half-spaces in which Δ_s is more compact than Δ_f by the constant to its right. For example, when $\mathbf{C} = 17$ and $\mathbf{D} = .2$, Δ_f requires four times as much space to encode as Δ_s.
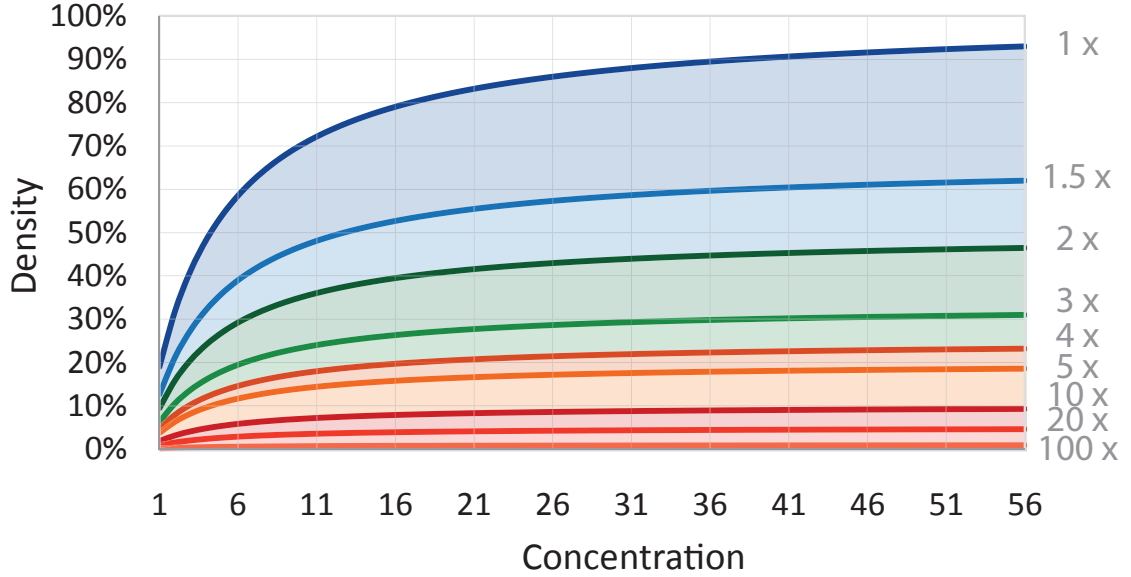
Figure 7.2: Comparison between storage costs of a full DMSF Δ_f and a supercube-based partial DMSF Δ_s in terms of density $\mathbf{D}$ and concentration $\mathbf{C}$ for $b_\delta = 4$, $b_v = 6$ and $b_s = 17$. Curves highlight the factor by which Δ_s is more compact than Δ_f.

7.5 Applications

In this section, we discuss several applications in which it can be beneficial to generate a partial DMSF Δ_p from a full DMSF Δ_f.

7.5.1 Error-based generation

As a first application, consider a partial DMSF Δ_p generated from a full DMSF Δ_f by retaining all diamonds whose error is greater than a given threshold ϵ. When $\epsilon = 0$, this generates a lossless encoding of Δ_f, i.e. Δ_f can be reconstructed from Δ_p without any error.

Since Δ_p is a partial DMSF, it must also retain all ancestors of the retained samples. This ensures the transitive closure of the diamond dependency relation (as described in Section 7.3).

Due to the transitive closure requirement, and the definition of the error metric there is a high degree of hierarchical coherence in addition to the spatial coherence among samples associated with a given supercube. Namely, if a diamond is required to satisfy a given selection criterion, it is likely that its neighbors, parents and children are also

Table 7.3: Number of supercubes (n_s) and diamonds (n_p) as well as disk size (in KB = 1024 Bytes or MB = 1024^2 Bytes) for supercube-based partial DMSFs extracted from full DMSF terrain datasets with different uniform errors. Note that the values for *Puget Sound 4k* dataset for 0% error are actually for 0.03% error rather than 0% error.

Dataset	Dims	10% Error			1% Error			0.1% Error			0% Error		
		n_s	n_p	size	n_s	n_p	size	n_s	n_p	size	n_s	n_p	size
San Bernadino	128^2	222	867	4.7K	1.2K	8K	39K	1.4K	14K	63K	1.4K	16K	71 K
Devil's Peak	165×301	275	1.5K	7.3K	2.8K	19K	89K	3.3K	32K	145K	3.3K	32K	145K
Grand Canyon	1280×640	3.5K	16K	83K	28K	172K	836K	62K	556K	2.5M	62K	556K	2.5M
Mt. Marcy	1201^2	1.3K	6K	32K	13K	68K	340K	63K	426K	2M	101K	939K	4.2M
Puget Sound 1k	1025^2	710	3.3K	17K	41K	219K	1.1M	82K	761K	3.4M	86K	1M	4.3M
Puget Sound 4k	4097^2	754	3.5K	18K	75K	354K	1.8M	880K	5.5M	26M	1.2M	9.7M	44M
Australia	4800×6000	18K	99K	490K	20.1K	110K	546K	50.1K	262K	1.3M	528K	4.78M	21M

necessary. The non-full supercubes are typically those that are close to the boundary of the domain or those containing leaf nodes.

7.5.1.1 Terrain modeling

An important example in the field of GIS, is the use of a supercube-based partial DMSF for terrain datasets. A partial DMSF provides a solution for a long-standing problem of representing subsets of a regular grid. For example, elevation data for surfaces and coastlines are required to be sampled at a high resolution, but regions covered by water are considered to be flat and are often highly oversampled. This is especially relevant for global datasets since approximately 70% of the earth's surface is covered by water.

We performed our experiments on several regular datasets, including *Digital Elevation Models (DEM)s* of Mt. Marcy, the Grand Canyon, Devils Peak, San Bernardino, two versions of the Puget Sound at different resolutions, and a tile of the gTopo30 covering a portion of Australia (see Figure 7.1). Datasets whose dimensions are not $(2^N + 1)^2$ were embedded into the smallest containing virtual grid of dimensions $(2^N + 1)^2$. All experiments were run on a 2 GHz Intel Core 2 Duo laptop with 4 GB of RAM.

Table 7.3 summarizes the sizes of partial DMSFs extracted from the dataset testbed with a range of uniform errors.

Figure 7.3 shows the average density of the various datasets from Table 7.3. As can be seen from the table, when encoding a supercube-based partial DMSFs of uniform error less than one percent, there are between 5 and 12 diamonds per supercube on average, and for errors less than 0.1 percent, the average concentration is between 9 and 11 diamonds for most datasets. Note also, that the average density of supercubes increases as the error threshold decreases.

Since the overhead per supercube b_s is 6 bytes, our supercube based partial DMSF representation has an overhead of less than 1 byte per sample when the average supercube density is greater than 6, an approaches 0.5 bytes per vertex as the average density approaches 12. Compared to the average of 3 Bytes of overhead per vertex in [Ger03a], our method is 3-6 times more space efficient. Furthermore, the stack-based method of [Ger03a] does not provide random access to its vertices and requires up to $O(\sqrt{n_p})$ extra storage in memory.
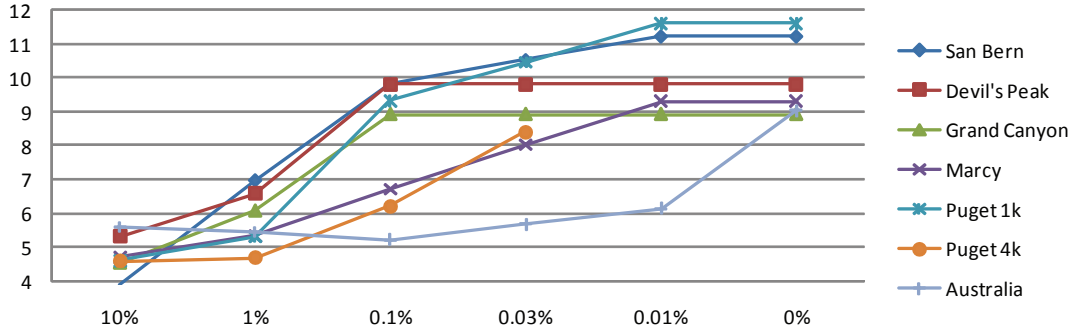
Figure 7.3: Average density of supercubes (vertical axis) with uniform error (horizontal axis). Derived from Table 7.3 as (n_d/n_s).

7.5.1.2 Modeling volume data

Next, we consider the problem of modeling three-dimensional volume datasets. We generated lossless ($\epsilon = 0\%$) and lossy ($\epsilon = 1\%$) partial DMSFs from a testbed of volumetric datasets of resolution up to 512^3 from the Volume Visualization repository [Vol].

Table 7.4 lists the number of elements, and storage costs in a zero-error partial DMSF Δ_p for the various datasets as well as their density **D** and concentration **C**. These datasets are plotted on Figure 7.4 for $\epsilon = 0\%$ (red) and $\epsilon = 1\%$ (orange). We observe that some datasets, such as Fuel and Aneurysm are extremely sparse, and achieve a 12.2 times and 17.3 times reduction in storage requirements, respectively, compared to the full DMSF dataset Δ_f. In contrast, other datasets such as Plasma and Buckyball are quite dense, and thus, a partial representation does not yield a significant savings compared to Δ_f. However, even for these datasets, the size of Δ_s is close to that of Δ_f (requiring 4% more and 19% less space, respectively), whereas Δ_d is much larger (requiring 2.4 times and 1.9 times more space, respectively). Most of the remaining datasets achieve around three times savings for Δ_s compared to Δ_f (see penultimate column in Table 7.4).

Since b_v (6 bytes) is 1.5 times as large as b_δ (4 bytes), the overhead associated with Δ_d compared to the ideal representation Δ_p is 150%. In contrast, the overhead of Δ_s (i.e. Δ_s/Δ_p) is related to the concentration of the supercube clustering, and averages around 12% across all datasets. Thus, the 2.25 times savings achieved by Δ_s compared to Δ_d is entirely due to the difference in geometric overhead.

7.5.2 Range-based generation

Partial DMSFs can also be used to reduce the storage requirements and mesh extraction times required for isosurface extraction when the (set of) isovalue(s) can be determined in advance. In isosurfacing applications, the *active* cells, i.e. those that intersect the isosurface, typically occupy a sparse but spatially widespread subset of the domain. Since isosurfaces are continuous, there is a great deal of spatial and hierarchical coherence among the active cells.

We can thus generate an isovalue-based partial DMSF Δ_p from Δ_f, where all diamonds whose range intersects the predetermined isovalue(s) are retained, while those not

Table 7.4: DMSFs generated based on uniform field error with $\epsilon = 0$ from volumetric datasets with maximum level of resolution N, containing $(2^N + 1)^3$ samples. File sizes for the full DMSF (Δ_f), the diamond-based partial DMSF (Δ_d) and the supercube-based partial DMSF (Δ_s) are listed in MB (1 MB = 1024^2 B). The density ($D = n_p/n_f$) values are in the range $[0, 1]$ and concentration ($C = n_p/n_s$) values are in the range $[1, 56]$. All datasets are plotted on Figure 7.4 (red circles).

Dataset	N	n_f	n_p	n_s	D	C	Δ_f	Δ_d	Δ_s	Δ_f/Δ_s	Δ_s/Δ_p
Fuel	6	275 K	19.9 K	620	.07	32.2	1.05	.19	.09	12.2 x	1.10 x
Neghip	6	275 K	129 K	3.46 K	.47	37.2	1.05	1.23	.55	1.91 x	1.09 x
Plasma	6	275 K	265 K	4.98 K	.97	53.2	1.05	2.53	1.09	.96 x	1.06 x
Hydrogen	7	2.15 M	545 K	16.0 K	.25	34.0	8.19	5.19	2.34	3.50 x	1.10 x
Buckyball	7	2.15 M	1.65 M	38.3 K	.77	43.0	8.19	15.7	6.90	1.19 x	1.08 x
Aneurysm	8	17.0 M	791 K	44.6 K	.05	17.7	64.8	7.54	3.74	17.3 x	1.18 x
Tooth	8	17.0 M	5.23 M	104 K	.31	50.1	64.8	49.9	21.7	2.99 x	1.06 x
Engine	8	17.0 M	5.34 M	112 K	.31	47.6	64.8	50.9	22.2	2.92 x	1.07 x
Head	8	17.0 M	5.47 M	139 K	.32	39.4	64.8	52.1	23,1	2.80 x	1.08 x
Bonsai	8	17.0 M	5.00 M	147 K	.29	34.1	64.8	47.7	21.5	3.02 x	1.10 x
Foot	8	17.0 M	5.90 M	151 K	.35	39.2	64.8	56.3	25.0	2.59 x	1.08 x
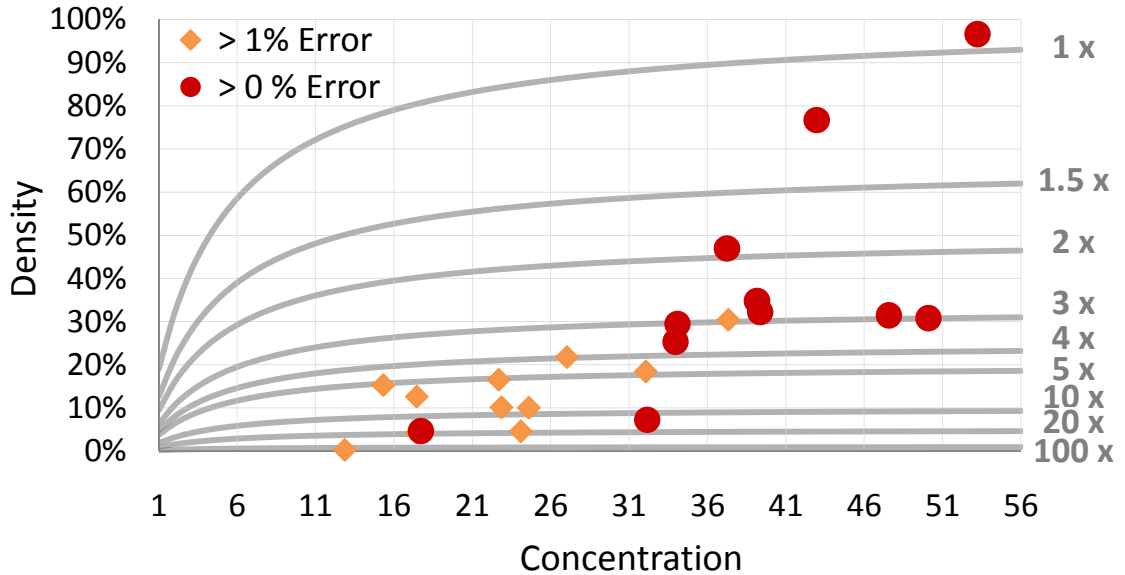
Figure 7.4: Density and concentration of partial DMSF datasets of Table 7.4 extracted from a complete DMSF with uniform error greater than 0% (red) and 1% (orange). Gray curves indicate the factor by which a supercube-based partial DMSF Δ_s is more compact than a full DMSF Δ_f.

intersecting the isovalue(s) are only retained if they are ancestors of the required diamonds. Δ_p can then be queried using selective refinement to extract adaptive simplicial complexes. This model thus trades fidelity in regions away from the desired isosurface for storage and extraction efficiency of the desired isosurface(s). We present results on volumetric datasets, but since contour plots are a common metaphor in GIS, a range-based DMSF consisting of samples pertaining to a series of significant isocontours can be a useful representation as well.

Table 7.5 lists the number of elements and the storage requirements for the three DMSF representations for each isovalue-based volumetric dataset (the values of N and n_f can be found in Table 7.4). The density and concentration of these datasets are plotted in Figure 7.5. We observe that these extracted partial DMSFs are indeed sparse with respect to Δ_f, averaging around 5% of the samples and often much less. They are also quite concentrated with respect to the supercube clustering, with an average concentration of 26 out of a possible 56 diamonds per supercube. Thus, supercube-based partial DMSFs of these datasets require an average of 25 times less storage than their corresponding full DMSFs. In fact, the largest dataset $\mathrm{Xmas}_{\{868\}}$ (orange square in Figure 7.5) requires only 1.3% of the samples of Δ_f and is over 65 times more compact.

As in the error-based partial DMSFs, the supercube-based encodings are approximately 2.3 times smaller than a corresponding diamond-based DMSF, and have very low overhead (around 13%) compared to the ideal representation Δ_p.

We note that, when more than one isovalue is desired (as in the set of Engine datasets on the bottom of Table 7.5, and the corresponding colored rhombuses in Figure 7.5), there is also a significant amount of coherence among the active cells of distant isovalues (e.g. $\kappa = 58$ and $\kappa = 170$). Thus, the supercube-based representation for the Engine dataset with two isovalues, $\mathrm{Engine}_{\{58,100\}}$, requires only 15% more storage space than either of the individual datasets $\mathrm{Engine}_{\{58\}}$ or $\mathrm{Engine}_{\{100\}}$, and has a higher concentration than either of them. This advantage is increased in $\mathrm{Engine}_{\{58,100,170\}}$ as the samples from a third isovalue are added, where the density only increases by 3% and the supercube concentration increases by 0.8.
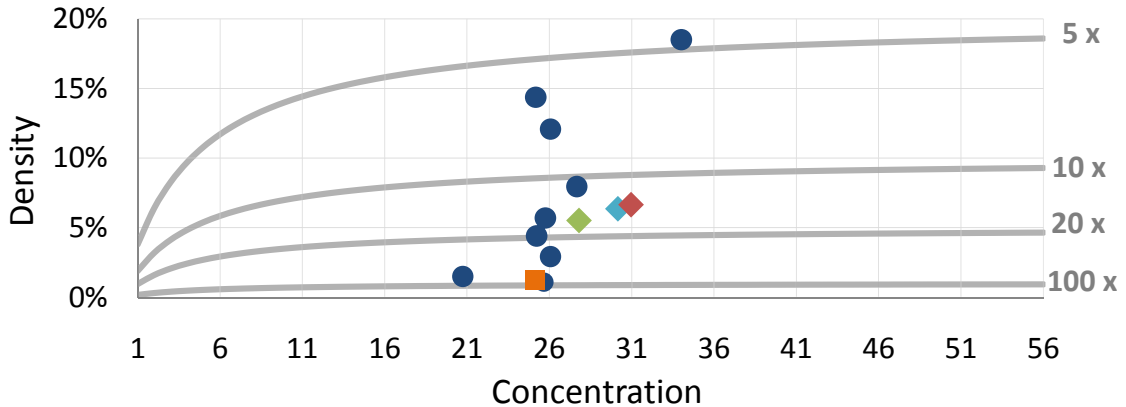
Figure 7.5: Density and concentration of partial DMSF datasets of Table 7.5 containing all diamonds intersected by the specified isovalue(s).

Table 7.5: DMSFs generated based on specific isovalue(s) for three dimensional datasets. File sizes for the full DMSF (Δ_f), the diamond-based partial DMSF (Δ_d) and the supercube-based partial DMSF (Δ_s) are listed in MB (1 MB = 1024^2 B). All datasets are plotted on Figure 7.5.

Dataset$_{\{Isovalue(s)\}}$	n_p	n_s	D	C	Δ_f	Δ_d	Δ_s	Δ_f/Δ_s	Δ_s/Δ_p
Fuel$_{\{7.2\}}$	15.7 K	608	.057	25.8	1.05	.15	.07	15.1 x	1.13 x
Neghip$_{\{868\}}$	39.5 K	1.57 K	.144	25.2	1.05	.38	.18	5.95 x	1.13 x
Hydrogen$_{\{24\}}$	63.1 K	2.42 K	.029	26.1	8.19	.60	.028	29.3 x	1.12 x
Bucky$_{\{128\}}$	2.59 K	9.94 K	.121	26.1	8.19	2.47	1.15	7.12 x	1.12 x
Aneurysm$_{\{128\}}$	255 K	12.3 K	.015	20.8	64.8	2.43	1.17	55.3 x	1.16 x
Tooth$_{\{650\}}$	1.87 K	7.27 K	.011	25.7	64.8	1.78	.83	78.1 x	1.13 x
Bonsai$_{\{35\}}$	1.35 M	48.8 K	.008	27.7	64.8	12.9	5.94	10.9 x	1.12 x
Foot$_{\{23.5\}}$	3.14 M	92.3 K	.185	34.0	64.8	30.0	13.5	4.80 x	1.10 x
Head$_{\{58\}}$	749 K	29.7 K	.044	25.2	64.8	7.14	3.34	19.4 x	1.13 x
Xmas$_{\{868\}}$	1.74 M	69.3 K	.013	25.1	515	16.6	7.76	66.4 x	1.13 x
Engine$_{\{58\}}$	937 K	33.7 K	.055	27.8	64.8	8.94	3.12	15.7 x	1.12 x
Engine$_{\{100\}}$	937 K	33.7 K	.055	27.8	64.8	8.94	4.12	15.7 x	1.12 x
Engine$_{\{58,100\}}$	1.08 M	35.8 K	.064	30.2	64.8	10.3	4.70	13.8 x	1.11 x
Engine$_{\{58,100,170\}}$	1.13 M	36.5 K	.067	31.0	64.8	10.8	4.90	13.2 x	1.10 x

7.5.3 Region Of Interest-based generation

Partial DMSFs are suitable for representing datasets extracted from a full DMSF using arbitrary selective refinement criteria. These include polygonal regions such as squares and circles (see Figure 7.6) as well as polylines. Additional error functions include distance or view dependent criteria as well as samples relevant to specific contour lines or ranges of contour lines within the datasets.

7.5.4 Merging corresponding partial DMSFs

Another interesting means of generating partial DMSFs is to merge two existing partial DMSFs, covering portions of the same domain and possibly at different resolutions. In the latter case, corresponding samples do not necessarily map to the same coordinates, but we assume that they correspond to the same regions of the domain.

We observe that the resolution of a dataset is a bottom-up distinction, that is, it is determined by the minimum distance between samples. However, corresponding datasets are aligned in a top-down rather than a bottom-up manner, i.e. their root diamonds cover the same hypercubic domain. Thus, a reinterpretation of our supercube-based representation in a top-down manner would enable the alignment of datasets of different resolutions. Since a diamond's scale is a bottom-up characteristic and its level is a top-down characteristic, this requires a method to represent supercubes by their *level* rather than by their *scale*.

Recall that the supercube structure clusters together those diamonds whose coordinates agree on all but two bits, i.e. the bits corresponding to their diamond type τ, and that
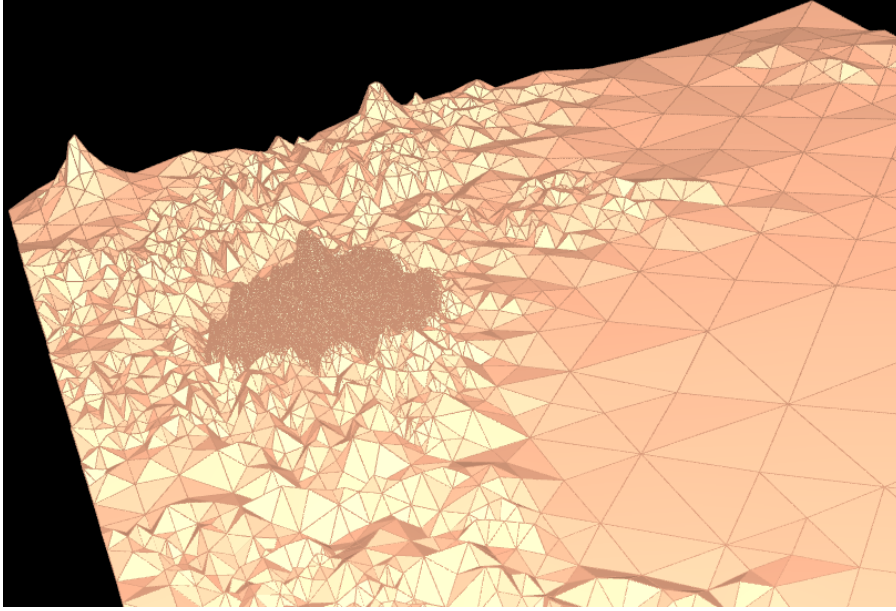
Figure 7.6: Circular region of interest (ROI) from Puget Sound 1k dataset. Values inside the ROI have 0 error while those outside the ROI have an approximation error less than 10%. This partial DMSF has 3600 supercubes with an average concentration of 10.3 diamonds per supercube.

all the bits to the right of these bits are zero. Consequently, the origin of a supercube $\mathbf{s}$ at scale γ has at least $\gamma + 2$ trailing zeros in each of its coordinates, and $\mathbf{s}$ can be unscaled by shifting its coordinates to the right by $\gamma + 2$ bits. Unscaled supercube origins at a particular level are thus a subset of the points of a regular grid.

Since the scale of a supercube is a function of the level of its diamonds as well as the resolution of the dataset, i.e. LEVEL_{Max}, a supercube based partial DMSF can store the unscaled coordinates of the origins of its supercubes and rescale them at runtime. Let $\mathbf{p}$ be the scaled coordinates of a supercube at scale γ whose unscaled origin is located at unscaled point $\mathbf{s}$. Then the central vertex $\mathbf{v}_c$ of a diamond with type τ can be calculated as: $\mathbf{v}_c = \mathbf{s} \cdot 2^{\gamma+2} + \tau \cdot 2^{\gamma}$. Note that since we are scaling by a multiple of 2, this operation can be efficiently executed using hardware bitshift operations. Supercubes can then be partitioned by either their level or their scale, corresponding to top-down or bottom-up representations, respectively.

Thus, aligning two top-down supercube based DMSFs is accomplished by simply setting the maximum level of the lower resolution dataset to that of the higher resolution dataset. Specifically, let $\mathbf{A}$ and $\mathbf{B}$ be two DMSFs where dataset $\mathbf{A}$ has a resolution of $(2^j + 1)^d$ and dataset $\mathbf{B}$ has a resolution of $(2^k + 1)^d$, such that $j < k$. Then $\text{LEVEL}_{Max}(\mathbf{A})$ is j and $\text{LEVEL}_{Max}(\mathbf{B})$ is k. A top-down supercube based DMSF of dataset $\mathbf{A}$ can be aligned with dataset $\mathbf{B}$ by simply increasing $\text{LEVEL}_{Max}(\mathbf{A})$ to k.

An advantage of this unscaled representation is that the dataset is no longer dependent on the resolution of the original grid and can be dynamically rescaled. Furthermore, this unscaled representation requires the same amount of storage as the scaled supercube based DMSF representation. However, since this requires rescaling the supercubes at runtime, the unscaled representation has a slight (but constant) computational overhead to

the scaled representation.

Example. As an example, consider a diamond in 2D with central vertex $\mathbf{v}_c = (44, 108)$ from a dataset of resolution $(2^8 + 1)^2 = 257^2$. Using the encoding of Section 6.1, we find that its scale is 2, its type τ is $(3, 3)$ and its scaled supercube origin is at $(32, 96)$. To unscale the supercube, we divide it by $2^{Scale(\delta)+2} = 2^4$ and obtain the point $(2, 6)$. Since the resolution of the grid is 2^8, the LEVEL$_{Max}$ is 8, and thus the supercube is at level LEVEL$_{Max} - Scale(\delta) = 8 - 4 = 4$. To convert this to a diamond or supercube in a dataset of resolution $(2^{10} + 1)^2 = 1025^2$, we observe that the scale of the supercube in the higher resolution is $10 - 4 = 6$. Consequently, the rescaled supercube needs to be scaled by a factor of 2^6 and will be at location $(2, 6) * 2^6 = (128, 384)$.

After adding the scaled diamond type, the central vertex of δ is located at $(176, 432)$, and the scale of δ is 4. Notice that although the resolution of the dataset increased (by a factor of $2^2 = 4$), the level of the diamond as well as that of the supercube remained constant, and the only change was in the value of LEVEL$_{Max}$, which changed from 8 to 10.

Transitive closure of the dependency relation. When merging two corresponding DMSFs, we typically need to insert additional vertices to maintain the transitive closure of the resultant DMSF. Each of these new vertices requires a scalar value as well as an error value. For the scalar values, we can recursively interpolate the value from the two vertices of its associated diamond's spine. This is guaranteed to terminate at the domain corners (i.e. the vertices of the root diamond), but will typically terminate much earlier. Since we want to ensure that the diamonds from the new dataset are reached, we set the error of the new points to the maximum possible error.

Due to the large number of shared ancestors within the DMSF, the number of points necessary to ensure transitive closure is often quite small relative to the number of diamonds being inserted, but depends on the location to which it is inserted. For example, let Δ_p be an empty two-dimensional DMSF with LEVEL$_{Max} = 30$, e.g. its equivalent full DMSF would have a resolution of $(2^{30}+1)^2$. Adding a 2x2 block of samples to Δ_p requires only a few hundred samples to maintain transitive closure, while adding a 1025^2 block of highest resolution samples to Δ_p has an overhead of less than 1%, e.g. whereas the 1025^2 block contains 1,050,625 samples, an empty partial DMSF of size $(2^{30} + 1)^2$ with this block added has fewer than 1,060,000 diamonds. Furthermore, adding a 4097^2 block from the same DMSF requires fewer than 0.1% additional samples to maintain transitive closure.

We simulate a situation where a higher resolution component is available by using two corresponding DEMs from the Puget Sound dataset, whose resolutions are 1025^2 and 4097^2, respectively. First, we extract a partial DMSF of uniform error less than 1% from the Puget Sound 1k dataset (see Figure 7.7a). This DMSF has 82 K supercubes and 761 K diamonds ($\mathbf{C} = 9.3$). Next, we merge it with a partial DMSF generated from the Puget Sound 4K dataset using a square ROI of side length 580 samples (see Figure 7.7b). This DMSF has 29 K supercubes and 341 K diamonds ($\mathbf{C} = 11.6$). The combined partial DMSF contains 108 K supercubes and 1.08 M diamonds ($\mathbf{C} = 9.95$). The number of shared samples between the two datasets is only 21 K or around 2% of the samples in the

resultant dataset. Figure 7.7 illustrates a terrain extracted from this DMSF.

(a) Puget Sound 1k at 1% error.

(b) Puget Sound 4k with square ROI.

(c) Combination of Puget datasets (shaded)

(d) Combination of Puget datasets (wire)
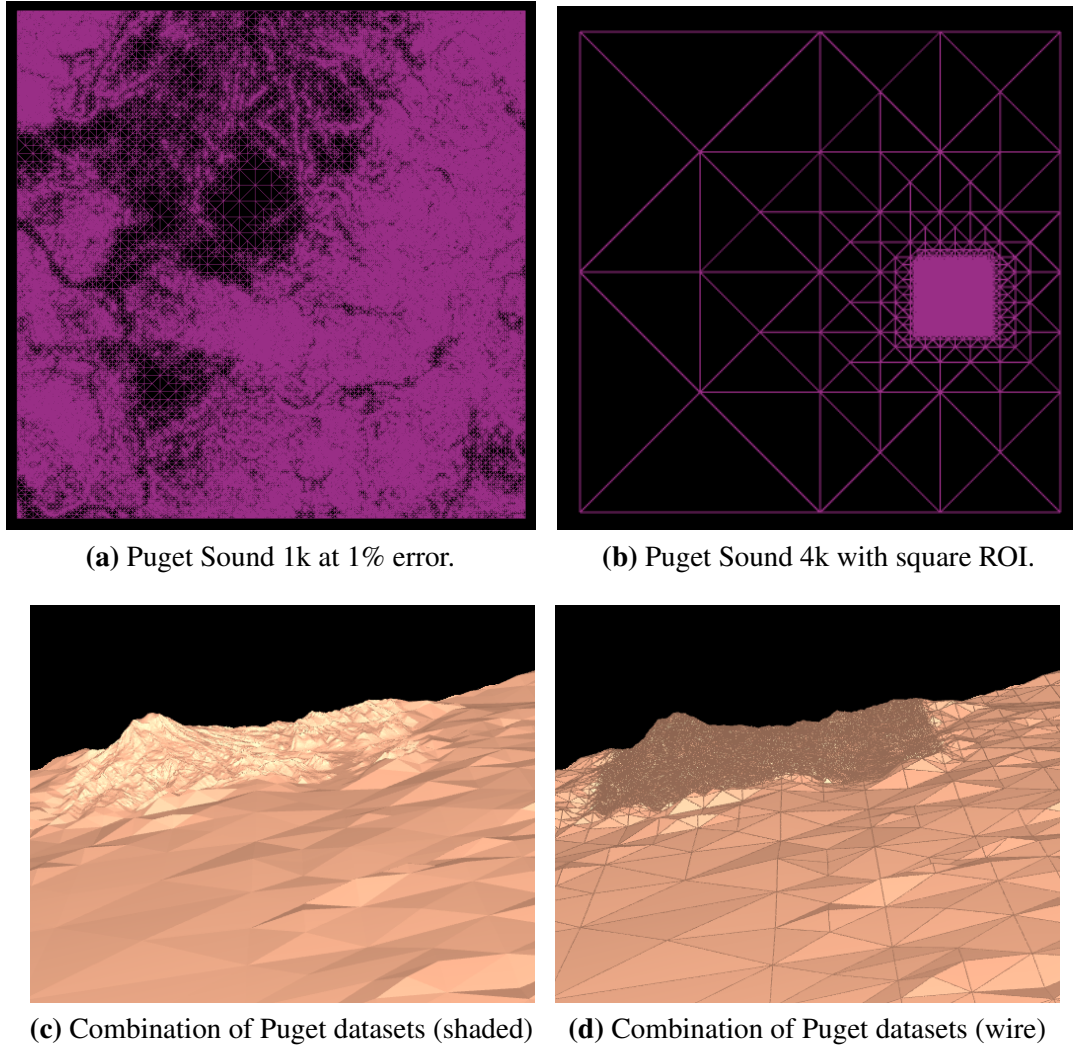
Figure 7.7: Planar projection of (a) the Puget sound 1k dataset with 1% error and (b) the Puget sound 4k dataset with square ROI. (c,d) Merged datasets after upscaling the lower resolution dataset. Shown without (c) and with (d) wireframe to highlight the size of the mesh elements.

7.6 Runtime performance

We can compare the runtime performance of the three DMSF representations, the full DMSF Δ_f, the diamond-based partial DMSF Δ_d and the supercube-based partial DMSF Δ_s, by comparing the rates at which they can process diamonds during selective refinement queries. Since an active front facilitates selective refinement, we evaluate the performance of the two active front representations introduced in Section 6.3. Recall that the active front of a selective refinement query corresponds to a simplicial complex *current mesh* Σ. On a DMSF, Σ is a conforming RSB mesh. This mesh can be represented using a diamond-

Table 7.6: Selective refinement performance of the three DMSF representations using a diamond-based active front representation Σ_d and a supercube-based active front representation Σ_s, in terms of the minimum, maximum and average number of diamonds (in thousands) visited per second.

| | Diamond-Based (Σ_d) | | | Supercube-Based (Σ_s) | | | Σ_s/Σ_d |
	Min	Max	Average	Min	Max	Average	Average
Δ_f	252 K/s	343 K/s	317 K/s	298 K/s	354 K/s	324 K/s	1.019 x
Δ_d	223 K/s	333 K/s	301 K/s	263 K/s	340 K/s	306 K/s	1.019 x
Δ_s	237 K/s	327 K/s	296 K/s	276 K/s	331 K/s	300 K/s	1.014 x

based representation, which we denote as Σ_d, or through a supercube-based representation which we denote as Σ_s.

Since selective refinement queries depend on the specific selection criterion used, we evaluate the performance of each structure in terms of the number of diamonds visited by the selective refinement query per second. In Table 7.6, we present the aggregate results over our testbed of volumetric datasets for an error-based isosurface extraction query and note that we observed the same trends when using different queries, such as region of interest and approximation error queries. The LOD criterion for this query selects all diamonds with approximation error greater than some threshold ϵ and containing a particular isovalue κ. As a partial DMSF for this query, we use the datasets generated in Table 7.5.

For these experiments, we implemented Δ_d using a hash table from the central vertex of each diamond in Δ_d to the data associated with it. This incurs a storage overhead inversely proportional to the *load factor* of the hash table, i.e., the ratio of diamonds in Δ_d to buckets in the hash table. Across all datasets tested, we found the load factor to average 73.5% (with a standard deviation of 16%). Thus, the hash-indexed Δ_d requires an average memory overhead of 36% compared to the values listed in Table 7.5. Similarly, for Δ_s, we used a separate hash table for each level of supercubes, and indexed the data associated with each supercube by its origin (as described in Section 6.2.1). We found the load factor to average 75% (with a standard deviation of 11%) across the datasets, thus, requiring an average memory overhead of 33% compared to the values listed in Table 7.5.

We evaluate the performance of the DMSF representations (i.e. the rows of Table 7.6) by comparing the average number of diamonds processed per second. Recall that due to the query type and the transitive closure of Δ_d and Δ_s, all three representations process the same set of diamonds and yield the same result. We first observe that all three representations yield similar performance results of about 300,000 diamonds per second. Δ_f is the fastest DMSF representation, since it can directly access its diamonds using the array location of their central vertices. Δ_d is approximately 5-6% slower than Δ_f, due to its use of indirect hashing, while Δ_s is around 7.5% slower than Δ_f. Thus, despite its required extra processing, such as extracting the supercube origin and diamond type and the prefix sum calculation, supercube-based Δ_s's performance is within 2% that of diamond-based Δ_d.

Next, we evaluate the relative performance of the two active front representations Σ_s and Σ_d by comparing columns 4 and 7 (AVERAGE) of Table 7.6. Thus, the supercube-based

active front representation Σ_s is, on average, 1-2% more efficient than the diamond-based active front representation Σ_d. Although this is not a significant difference, we note that the addition (removal) of any diamond to (from) Σ_d incurs a memory allocation (deallocation), whereas, due to the supercube clustering, such allocations (deallocations) are rarer for Σ_s.

Finally, we evaluate the sizes of the two active front representations. Recall that the supercube-based active front representation Σ_s requires 27 bytes overhead per supercube. Over the entire test, Σ_s averaged 26.5 tetrahedra per supercube (with a standard deviation of 3.1). Thus, the supercube-based active front Σ_s incurs an overhead of around 1 byte per tetrahedron in the active front. In contrast, the diamond-based active front representation Σ_d requires 7 bytes overhead per diamond. Over the entire test, Σ_d averaged 3.3 tetrahedra per diamond (with a standard deviation of .55). Thus, the diamond-based active front incurs an overhead of around 2.16 bytes per tetrahedron in the active front.

We implemented both Σ_s and Σ_d using hash tables, (analogously to our indexing of the partial DMSFs above). Across all datasets, we achieved an average load factor of 74% for Σ_d (with a standard deviation of 15%), and thus the hash-indexed Σ_d incured a memory overhead of around 36%. The average load factor for Σ_s was 72% (with a standard deviation of 11%), requiring an average overhead of 39%. Thus, a supercube-based active front representation Σ_s can be used to extract an equivalent mesh from a DMSF as a diamond-based representation Σ_d in slightly less time and using less than half the storage.

Figure 7.8 illustrates the clustering of tetrahedra in a supercube-based active front by the color of their isosurface triangles.

7.7   Discussion

We have introduced a dimension-independent compact representation for diamond-based multiresolution scalar fields. Our encoding of diamonds allows the recovery of all local mesh geometry and topology from the coordinates of the central vertex of each diamond.

Supercube clustering provides an efficient means of associating information with a subset of the diamonds within an RSB hierarchy by exploiting the spatial and hierarchical coherence within the dataset.

Our supercube clustering of the retained samples reduces the geometric overhead by compactly indexing up to $4^d - 2^d$ samples. Supercubes also enable an efficient encoding of an active front of a selective refinement query. We demonstrated the effectiveness of this clustering over a wide range of two and three-dimensional datasets and error criteria and discussed situations where a full DMSF would be more appropriate.

As such, supercube-based representations are most effective when encoding data on a subset of the diamonds of an RSB hierarchy that is *sparse* compared to an encoding for all diamonds in the hierarchy and *coherent* with respect to the supercube clustering.

Compared to the sparse representation of Gerstner [Ger03a] (in 2D), our representation supports random access as well as general selective refinement queries and does not require keeping track of an automaton or stacks of scalar values. The overhead of our supercube based representation is less than 1 byte per sample in 2D, compared to 3 bytes per sample in [Ger03a].

(a) Tooth, $\epsilon > 10\%$ **(b)** Fuel, $\epsilon > .03\%$
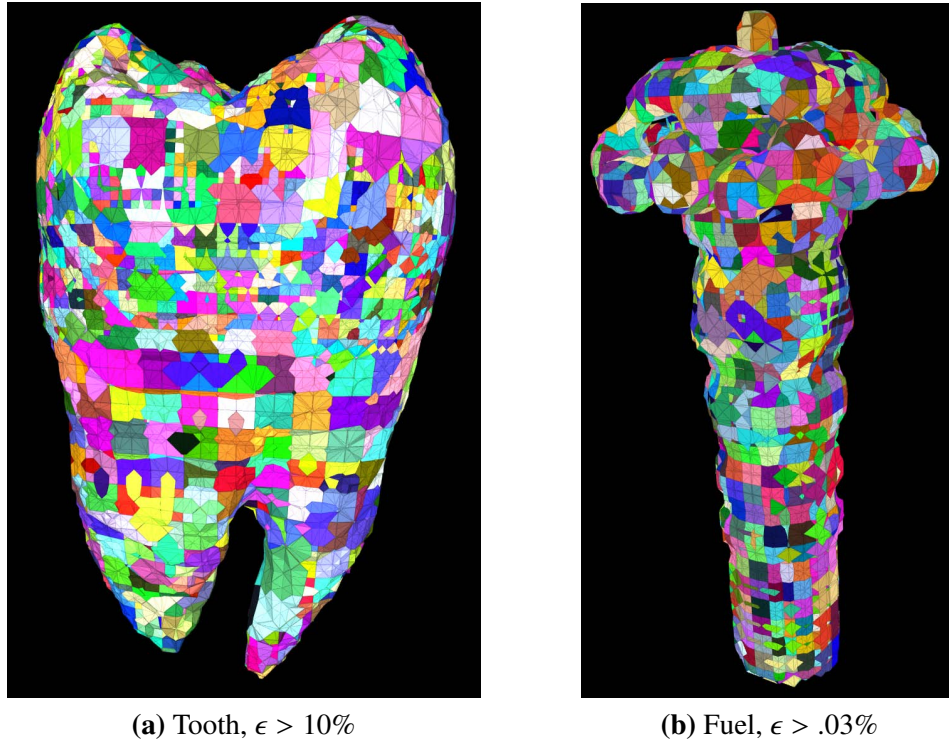
Figure 7.8: Isosurfaces extracted from DMSF models. Triangles are colored by their embedding supercube. (a) Tooth dataset with uniform error $\epsilon > 10\%$. A supercube-based active front representation Σ_s has an average of 26.9 tetrahedra per supercube yielding an average overhead of 1 byte per tetrahedron. The diamond-based active front Σ_d has an average of 2.9 tetrahedra per diamond yielding an overhead of 2.4 bytes per tetrahedron. (b) Fuel dataset with uniform error $\epsilon > .03\%$. Σ_s has an average of 25.6 tetrahedra per supercube yielding an average overhead of 1.05 bytes per tetrahedron. Σ_d has an average of 3.3 tetrahedra per diamond yielding an overhead of 2.12 bytes per tetrahedron.

We have demonstrated that several common volumetric datasets (in 3D) are oversampled by a factor of three or more while at the same time the retained elements have a high degree of coherence with respect to supercube clustering. Thus, a supercube-based partial DMSF is an effective multiresolution representation that efficiently supports selective refinement queries with very little geometric or computational overhead. We have also demonstrated that a supercube-based active front representation can accelerate selective refinement while requiring less than half the storage of a diamond-based active front data structure.

Here, we have focused on the implementation of the internal map between supercubes and their associated data. However, since our indexing structure for the supercubes at each level utilizes a hash table, our partial DMSF representations can be inflated by as much as 40%. This can reduce the benefits of a supercube-based DMSF Δ_s to a full DMSF Δ_f, when their relative differences are less pronounced. However, when the relatives sizes are more significant, a hash-indexed Δ_s still provides a significant advantage over Δ_f. Additionally, we have demonstrated that hash-indexed DMSFs and active front representa-

tions based on diamonds suffer from similar or worse overhead than their supercube-based counterparts. Alternatively, since Δ_d and Δ_s contain static spatial data, a perfect spatial hash [LH06] can yield significantly lower overhead than a standard hash table.

Our discussion of supercubes has focused only on the storage requirements of a nested RSB's spatial decomposition. An interesting extension would be to utilize a supercube-based DMSF for analyzing the dataset's range. For example, we could add a base value or error value to a supercube to enable compression of the associated diamond's scalar and error values. Another compression method for the errors might be to quantize them on a level by level basis as in [GDL$^+$02], where error components are quantized to 6 bits. Alternatively, downsampling the data, rather than subsampling can improve the quality of approximated meshes [HK95, LPD$^+$04].

Another interesting direction relates to the connection between the resolution of an extracted mesh and the *distortion* (a higher-dimensional generalization of curvature) that the retained field values induce on its domain [MDP08]. Our preliminary results [WMD10, DIM$^+$11] indicate that a distortion-guided extraction directs the mesh resolution towards the salient features of the field. This enables accurate analysis of complex datasets using significantly fewer resources.

Chapter 8

Topological navigation on diamond meshes

In shape modeling and analysis applications, we are often interested in computing local properties about elements within a mesh. Such queries are typically posed in terms of local neighborhoods surrounding a region of the mesh and require efficient support for navigating its topological connectivity. Examples in 2D include *visibility queries*, such as computing the *viewshed* or the *horizon* of a point [DM03]; generating compressed representations of shapes [ESV96, Ros99]; computing the local curvature within a region of the mesh [GG06]; and extraction of morphological representations in terms of *critical points* and *integral lines* [Ban70, EHZ03]. When modeling volumetric datasets, efficient topological queries are necessary for applications such as ray casting [BKS97, EHK+06]; morphological analysis [EHNP03]; and and computing the local mesh *distortion* (three dimensional analogue of discrete curvature) [MDP08]. Additionally, in geometry processing applications, the *1-ring* of a vertex (i.e. its adjacent vertices) is often required for computing properties such as the mesh Laplacian [DKT05]. Ideally, we would like to extract such entities in an output sensitive manner, i.e. in time that is linear in the size of the affected neighborhood.

   Due to the increasing sizes of datasets, it is important to reduce the storage requirements associated with the mesh's topological connectivity. Not only does this require additional storage space, but it must also be maintained when modifying the mesh, for example, during mesh simplification or refinement. Thus, many topological data structures have been developed for encoding cell and simplicial meshes (see [DH07] for a recent survey). Such data structures differ in their representation domain, in their support for mesh navigation, and in the subset of the incidence and adjacency relations that they encode.

   Interestingly, when dealing with structured families of meshes, we can often exploit the unique properties induced by the mesh generation process to yield more efficient data structures. For example, when modeling multiresolution tetrahedral meshes using the *half-edge collapse* operator, the entire multiresolution structure can be encoded using less space than the mesh at full resolution, while also providing efficient support for *selective refinement* queries [CDM+04]. Compared to the general case, the loss in representational power can often be made up for by the gains in encoding and processing efficiency obtained by exploiting the structure of the mesh.

   Diamond meshes are a compact representation for conforming RSB meshes and exploit the fact that they are generated through the RSB scheme to achieve a compact and efficient encoding. They do not, however, explicitly encode any topological connectivity information such as the incidence or adjacency among neighboring mesh elements.

   In this chapter, we introduce optimal querying algorithms for adjacency-based and incidence-based topological relations on two- and three-dimensional diamond meshes us-

ing only information about the *existence* of vertices and diamonds in the mesh. In contrast, previous approaches have only considered the boundary and hierarchical relationships of diamonds [GDL+02], or navigation on simplex-based RSB meshes [EKT01, LDS01, AM07], which require significantly more storage than diamond-based representations.

Since diamond meshes are typically extracted from a multiresolution model of the domain, these algorithms enable topological navigation to be performed directly on these extracted diamond meshes without requiring an auxiliary step to compute and store topological information for the entire mesh. As we demonstrate in our experiments, this can inflate the mesh by an order of magnitude (see Table 8.2a).

The remainder of this chapter is organized as follows. We first review relevant background notions in Sections 8.1. In Section 8.2, we analyze properties of diamond hierarchies and diamond meshes, which forms the basis of our navigation algorithms. We provide a general overview of our approach in Section 8.3. We then develop algorithms for extracting the topological relations on two-dimensional diamond meshes in Section 8.4, and generalize these to three-dimensional diamond meshes in Section 8.5. In Section 8.6, we compare the storage and computational complexity against other approaches.

8.1 Topological relations

Let us consider a simplicial d-complex Σ and a p-simplex $\sigma \in \Sigma$, with $0 \leq p \leq d$. The *topological relations*, which we denote as $R_{p,q}$, are defined over Σ in terms of the incidence and adjacency among its simplices:

Boundary relation $R_{p,q}(\sigma)$, with $0 \leq q < p$, consists of the set of q-simplices that are faces of σ.

Co-boundary relation $R_{p,q}(\sigma)$, with $p < q \leq d$, consists of the set of q-simplices incident to σ.

Adjacency relation $R_{p,p}(\sigma)$ consists of the set of p-simplices in Σ that are $(p{-}1)$-adjacent to σ (when $p > 0$), or the set of 0-simplices that are adjacent to σ through an edge (when $p = 0$).

Occasionally, we are interested in a subset of the relation, i.e. a *partial relation*, as a way of initializing a query. We denote this as $R_{p,q}^*$.

We refer to any relation which involves a constant number of entities as *constant*, and to relations involving a variable number of entities as *variable*. In general, the co-boundary and adjacency relations in a simplicial complex are variable, while the boundary relations are constant. A constant relation should be retrieved by a data structure representing Σ in constant time, while variable relations for a simplex σ should be retrieved by examining a local neighborhood of σ. A query is said to support *optimal* retrieval of a topological relation if the required time is linear in the relation's cardinality. If the retrieval of a relation requires examining all the simplices of a specific dimension, then the data structure does not support efficient retrieval of that relation.

(a) Vertices of a diamond mesh   **(b)** Top simplices of a diamond mesh
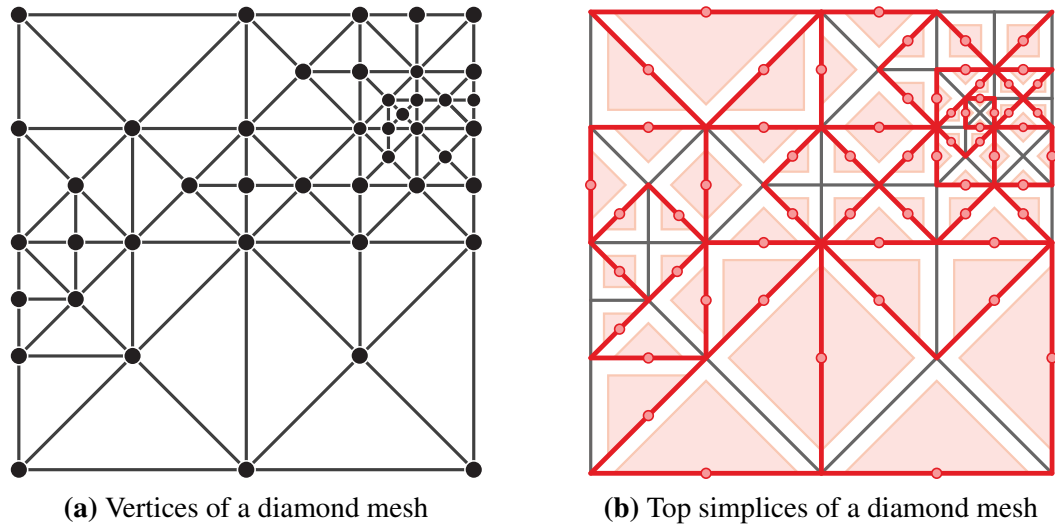
Figure 8.1: A diamond mesh is a conforming RSB mesh extracted from a hierarchy of diamonds and is in correspondence with a *closed* set of vertices from the hierarchy (a). Each top simplex (light pink in (b)) belongs to a single diamond (filled pink circle at midpoint of red edges in (b)).

(a) 2D diamonds   **(b)** 3D diamonds
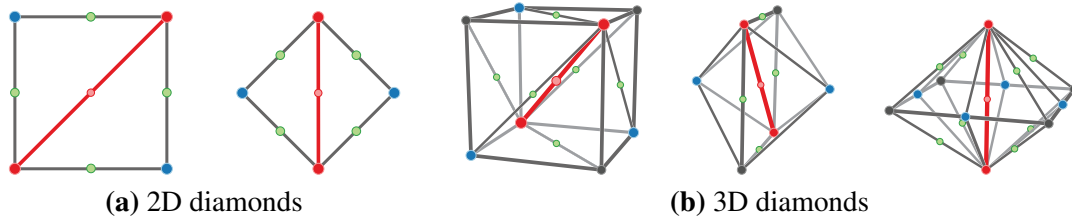
Figure 8.2: The two classes of diamonds in 2D (a) and the three classes of diamonds in 3D (b). A diamond's *central vertex* is at the midpoint of its *spine* (red edge). The blue and green circles coincide with the central vertices of a diamond's *parents* and *children*, respectively.

8.2 Properties of diamond meshes

A *diamond mesh* Σ is a conforming RSB mesh extracted from a hierarchy of diamonds. It consists of a collection of diamonds each with at least one duet containing top simplices in Σ (see Figure 8.1 for an example in 2D).

Recall from Section 4.6 that a diamond is *complete* if all of its duets belong to the diamond mesh Σ, and that *incomplete* diamonds may not subdivide. Since the top simplices in a diamond are generated during the refinement of its parents, a diamond can only be subdivided after all of its parents have been subdivided. As a consequence, diamond meshes are *balanced*, in the sense that facet-adjacent top simplices can differ by at most one refinement depth.

Since vertices are only introduced into a mesh during the subdivision of a diamond and each such vertex coincides with a subdividing diamond's central vertex, the following property holds for diamond meshes:

Property 1. *The existence of vertex **v** in diamond mesh Σ implies that the diamond δ whose central vertex is **v** has been subdivided.*

Thus, as observed in [LKR⁺96, Paj98, Pup98], the diamond dependency relation induces a dependency relation on the vertices of the hierarchy. Also, there is a unique correspondence between a *closed* set of vertices from the hierarchy and a diamond mesh Σ defined on those vertices, where the set is *closed* with respect to the transitive closure of the direct dependency relation (see Figure 4.17).

This provides a framework for analyzing the *longevity* of certain elements of an RSB mesh as it is refined, i.e. the number of depths in the hierarchy in which it can exist in a mesh. Consider a diamond mesh generated through a top-down refinement of a diamond hierarchy of infinite depth (where diamonds are refined, but never coarsened).

Lifespan of a vertex. Vertices are introduced into a diamond mesh at the central vertices of subdividing diamonds. Thus, we can associate a *level* and a *depth* to a vertex in correspondence to the level and depth of its associated diamond. After a vertex is inserted into a diamond mesh, it is never removed.

Lifespan of an edge. Each edge in a diamond hierarchy uniquely corresponds to the spine of a single diamond, which can be determined by considering the midpoint of the edge. Edges are introduced into a diamond mesh as a *subdivision edge* during the subdivision of a diamond δ between the central vertex of δ and the remaining vertices of δ (see Section 4.4.1). An edge **e** is removed from the mesh during the subdivision of the diamond whose spine is **e**. It is replaced by two edges bisecting it at its midpoint (i.e. the one-dimensional RSB operation). Thus, the lifespan of an edge is approximately one level of refinement (i.e. around *d* refinement depths in the hierarchy). In 2D, edges always survive for two refinements (see Figure 8.3). In 3D, edges aligned with cube diagonals survive for three refinements, while edges aligned with face diagonals and cube edges can survive for two, three or four refinements.

Lifespan of a top simplex. Simplices in a diamond mesh survive for exactly one refinement. They are created during the refinement of a diamond δ_p, and are uniquely associated with a single child diamond δ_c of δ_p. Upon the subdivision of δ_c, they are removed from the mesh.

8.3 Retrieving topological relations on diamond meshes

In this section, we provide an overview of our approach to querying the topological connectivity of a diamond mesh. We assume a simple interface to the diamond mesh Σ requiring only two predicates CONTAINSVERTEX(·) and CONTAINSDIAMOND(·) which query a randomly accessible collection of vertices and diamonds in the mesh based on the coordinates of a vertex, or of the central vertex of a diamond, respectively. In contrast to the encoding provided in Section 6.3, we do not assume that a diamond tracks which of its duets are present in the mesh. Since a duet *d* in a diamond δ in diamond mesh Σ

(a) Edge creation **(b)** Edge destruction
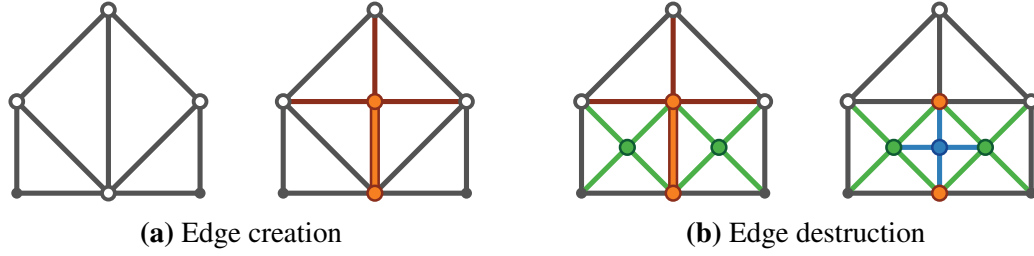
Figure 8.3: Edges in a *d*-dimensional diamond mesh survive for approximately *d* refinement depths. An edge **e** (thick orange line on the right of (a)) is created during the subdivision of a diamond δ, and connects a vertex of δ to its central vertex. It survives during the refinement of any diamonds whose external boundary contains **e** (left of (b)) and is destroyed during the subdivision of the diamond whose spine is **e** (right of (b)).

corresponds to a subdivided parent δ_p of δ, Property 1 implies that δ_p's central vertex $\mathbf{v}_{\delta_p}$ will be in Σ (i.e. CONTAINSVERTEX($\Sigma, \mathbf{v}_{\delta_p}$) will be TRUE) when d is in Σ.

Our topological queries are modeled after the incidence- and adjacency-based topological relations and incorporate diamonds into the query. Thus, in addition to the $(d+1)^2$ topological relations $R_{p,q}$ involving p- and q- simplices, where $0 \leq p, q \leq d$, we introduce $2d+3$ topological relations that operate on diamonds. The $d+1$ *diamond co-boundary relations* for a p-simplex σ in Σ, which we denote as $R_{p,\diamond}(\sigma)$, consist of the set of diamonds in the mesh that are incident to σ. For example, the *Vertex-Diamond* relation of a given vertex v, denoted as $R_{0,\diamond}(v)$, is the set of diamonds incident to v (e.g. the pink diamonds surrounding the blue vertex in Figure 8.4c).

Similarly, the $d+1$ *diamond boundary relations*, which we denote as $R_{\diamond,q}$, consist of the set of q-simplices within Σ that are incident to a given diamond. Note that we always include the spine of a diamond as a member of the *Diamond-Edge* relation $R_{\diamond,1}$ even though it is not on the external boundary of complete diamonds (see Figure 8.2).

Finally, the *Diamond-Diamond* relation, which we denote as $R_{\diamond,\diamond}$, consists of the set of diamonds that are adjacent to a given diamond in the mesh, i.e. the diamonds that contain a top-simplex that is adjacent to a top-simplex in the given diamond.

We also consider *constrained* boundary and adjacency relations on diamonds, which we denote as $R_{\diamond,q|\sigma}$ and $R_{\diamond,\diamond|\sigma}$, respectively. These relations are a subset of the full relation *subject* to also being incident to a given simplex σ in its boundary. For example, when querying the mesh we might be interested in the *Diamond-Diamond* relation subject to incidence on a common vertex v, which we denote as $R_{\diamond,\diamond|v}$. This is useful since it can help find the diamonds containing simplices in the star of the vertex (see Section 8.4.3).

In general, to retrieve relation $R_{p,q}(\sigma)$ of a given p-simplex σ in diamond mesh Σ, we first find a subset of σ's incident diamonds, i.e. $R_{p,\diamond}^*(\sigma)$. If applicable, this is followed by iterating through the constrained diamond adjacency relations $R_{\diamond,\diamond|\sigma}$ subject to incidence with σ. This completes the *Simplex-Diamond* relation $R_{p,\diamond}(\sigma)$. Finally, we apply the constrained diamond boundary relation $R_{\diamond,q|\sigma}$ to each element in $R_{p,\diamond}(\sigma)$, achieving our desired result $R_{p,q}(\sigma)$ (see Figure 8.4). All such queries are optimal: The first step is constant, as is each iteration of the second and third step. Since the latter two steps must

(a) Diamond mesh Σ **(b)** $R_{0,\diamond}^*(\mathbf{v})$ **(c)** $R_{0,\diamond}(\mathbf{v})$ **(d)** $R_{0,1}(\mathbf{v})$
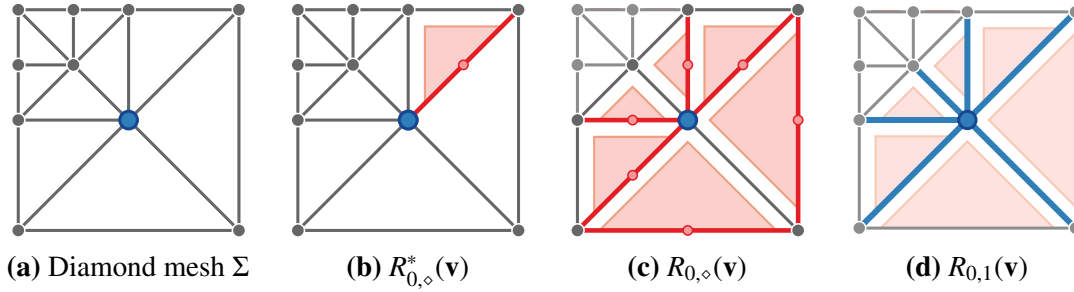
Figure 8.4: Extraction of *Vertex-Edge* relation $R_{0,1}(\mathbf{v})$ for the blue vertex $\mathbf{v}$ in (a). First, we extract the *partial Vertex-Diamond* co-boundary relation $R_{0,\diamond}^*(\mathbf{v})$ by finding a diamond in the co-boundary of v (b). Next, we iterate through the constrained *Diamond-Diamond* adjacency relation $R_{\diamond,\diamond|\mathbf{v}}$ to obtain the full *Vertex-Diamond* relation $R_{0,\diamond}(\mathbf{v})$ (c). Finally, we extract the constrained *Diamond-Edge* boundary relation $R_{\diamond,1|\mathbf{v}}$ to obtain the desired result $R_{0,1}(\mathbf{v})$ (blue edges in (d)).

be applied a number of times that depends linearly on the relation's cardinality, the query is optimal.

8.4 Retrieving topological relations on 2D diamond meshes

8.4.1 Boundary relations involving 2D diamonds

All boundary relations involving diamonds are based on the duet construct (see Figure 4.13a). A two-dimensional duet d has three fields: *d.spineA*, *d.spineB* and *d.parent*, representing the location of the two spine vertices and the corresponding parent's central vertex. Thus, given a diamond δ, with central vertex $\mathbf{v}_c$, we can query its boundary relations by processing each of its two duets in constant time. Note that, for any diamond in Σ, its spine is always in Σ, and at least one of its duets are in Σ. The status of a duet d of diamond δ in mesh Σ is checked via the predicate CONTAINSVERTEX(Σ, *d.parent*). Recall from Section 6.1 that the coordinates of a diamond's vertices can be found using scaled offsets from its central vertex. Figure 8.2a highlights the spine (red edge) and the central vertices of the parents (blue filled circle) for the two classes of diamonds in 2D.

The *Diamond-Vertex* relation $R_{\diamond,0}$ of a diamond δ in Σ always includes both spine vertices of δ, and, for each duet d of δ that is in Σ, its associated *parent*'s vertex is also in $R_{\diamond,0}$ (see Figure 4.13a). The result of a Diamond-Vertex query can be expressed as a set of vertices (of cardinality at most four).

Similarly, the *Diamond-Edge* relation $R_{\diamond,1}$ of a diamond δ in Σ always contains the spine of δ, and for each duet d of δ in Σ, the edges from *d.parent* to each spine vertex of δ belong to $R_{\diamond,1}$ as well. The result of a Diamond-Edge query can be expressed as a set of vertex pairs or as a set of edge midpoints (of cardinality at most five).

The *Diamond-Triangle* relation $R_{\diamond,2}$ contains the triangle associated with each duet of δ that is in the mesh Σ. We can express $R_{\diamond,2}$ in terms of a *simplex bit flag*, consisting of two bits, where each bit correspond to one of the triangles in δ, or as a set of vertex triples, e.g. for rendering applications.

The diamond boundary relations can be easily specialized to the incidence of a given

---
Algorithm 8.1 Diamond-DiamondRelation(δ)

Require: Σ is a valid diamond mesh, i.e. a conforming RSB mesh
Require: δ is a diamond in Σ
Require: Colors reference Figures 8.5 and 8.9.
Ensure: *dSet* is the set of diamonds in Σ that are adjacent to δ
  1: $dSet \leftarrow \emptyset$
  2: **for all** Duets d in δ **do**
  3:   **if** ContainsVertex(Σ, *d.parent*) **then**
  4:     **for all** *childPt* $\in$ *ChildPointSet* **do**
  5:       **if** ContainsDiamond(Σ, *childPt*) **then**
  6:         Insert *childPt* into *dSet*
  7:       **else**
  8:         **if** ContainsDiamond(Σ, *neighborPt*) **then**
  9:           Insert *neighborPt* into *dSet*
 10:   **else**
 11:     Insert *d.Parent* into *dSet*
 12:
 13: **return** *dSet*

---

vertex. For example, if σ is a spine vertex of δ, then the *constrained Diamond-Triangle* relation $R_{\diamond,2|\sigma}$ is unaffected, but the *constrained Diamond-Edge* relation $R_{\diamond,1|\sigma}$ will omit the edges between the duet's parent vertices and its opposite spine vertex. Similarly, if σ is a parent vertex, then $R_{\diamond,2|\sigma}$ will omit the triangle from the other duet, regardless of its presence in Σ, and $R_{\diamond,1|\sigma}$ will omit the spine as well as both edges from the other duet.

The remaining boundary relations $R_{2,1}$, $R_{2,0}$ and $R_{1,0}$ can be obtained directly if the given edge or triangle is expressed as a tuple of vertices, or it can be queried in linear time by finding the incidence relations $R_{2,\diamond}(\sigma)$ or $R_{1,\diamond}(\sigma)$, where σ is a 2-simplex or 1-simplex, respectively (see Section 8.4.3), and then applying the appropriate diamond boundary query (subject to incidence with σ) as defined above.

8.4.2 Adjacency relations involving 2D diamonds

As noted in Section 8.2, one of the key properties of diamond meshes is that their edge-adjacent triangles are guaranteed to be within one subdivision of each other. Consider the duet d belonging to diamond δ, whose triangle is defined by the spine edge **e** := $(d.spineA, d.spineB)$, and whose third vertex is $d.parent$ (for example, the red triangle in Figure 8.5d). Then a triangle t adjacent to d along the spine e of δ is either at the same depth or it is one refinement closer to the root. In the former case, t is the triangle belonging to the other duet in δ, and δ is a *complete* diamond. In the latter case, t belongs to diamond δ_p, the parent diamond of δ corresponding to the other duet of δ (e.g. the blue triangle in Figure 8.5d).

On the other hand, if t is adjacent to d along a non-spine edge, then t is either one subdivision depth further from the root than δ, in which case, t is a triangle of a child diamond of δ (e.g. the green triangle in Figure 8.5d), or t is at the same subdivision depth
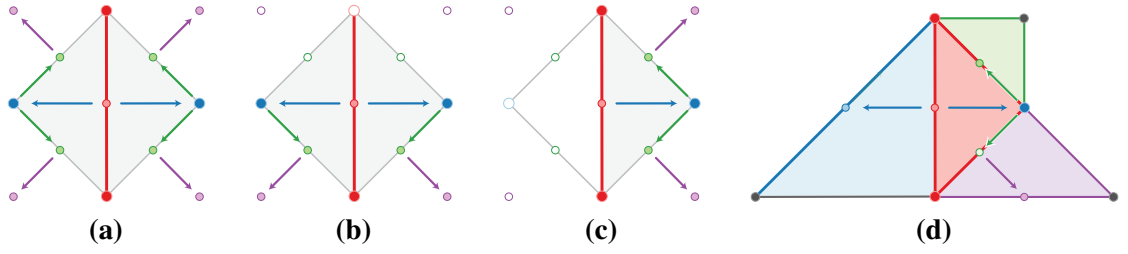
Figure 8.5: Tests for the diamonds adjacent to a given diamond (with red spine) in the general case (a), and subject to an incident spine vertex (b) or an incident parent vertex (c). An example $R_{\diamond,\diamond}$ relation for this diamond is shown in (d).

as δ, in which case, its associated diamond is a neighboring diamond of δ (e.g. the purple triangle in Figure 8.5d).

Since diamonds uniquely correspond to grid points within the hierarchy that are not vertices in the mesh (via their central vertices), we can satisfy adjacency queries using the ContainsVertex and ContainsDiamond predicates on a few easily calculated grid point near δ. These relationships are derived entirely from the position of the duet's vertices, as shown in Figure 8.5a and in Algorithm 8.1. Given a duet d, our first test involves d.parent. If the vertex is present (i.e. its associated diamond has subdivided), we check the two childPt vertices, coinciding with the midpoints of the edges between the d.parent and the spine vertices, i.e.

$$ChildPointSet = \begin{cases} \frac{1}{2}(d.parent + d.spineA) \\ \frac{1}{2}(d.parent + d.spineB). \end{cases}$$

In either case, if the corresponding diamond is not present, we check

$$neighborPt = \mathbf{v}_c + 2\left(childPt - \mathbf{v}_c\right).$$

Note that the set of neighbor points is a subset of the parents of the children of δ and also of the children of the parents of δ. That is, if δ_n is the diamond whose central vertex is *neighborPt*, then $\delta_n \in Children(Parents(\delta))$ and also $\delta_n \in Parents(Children(\delta))$.

As with the boundary relations, this algorithm can be constrained to accommodate an incident vertex or edge σ, which we denote as $R_{\diamond,\diamond|\sigma}$. Figures 8.5(b) and (c) illustrate the required changes to Algorithm 8.1 when the incident vertex is a spine vertex of δ, or if it is a parent vertex of a duet of δ.

8.4.3 Co-boundary relations involving 2D diamonds

The co-boundary relations are defined in terms of their diamond counterparts $R_{p,\diamond}$. We first consider the *Triangle-Diamond* relation $R_{2,\diamond}$, then the *Edge-Diamond* relation $R_{1,\diamond}$ and, finally, the *Vertex-Diamond* relation $R_{0,\diamond}$.

Recall that each triangle in Σ is uniquely contained by a single diamond δ in Σ. Given a triangle t defined by three bounding vertices, its corresponding diamond δ is
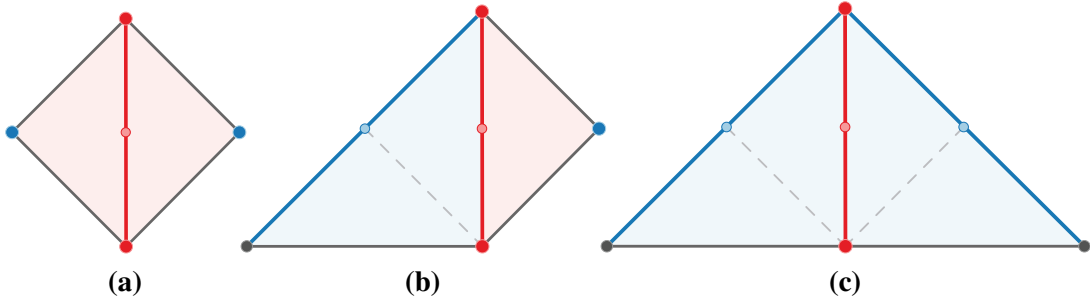
Figure 8.6: The three possible *Edge-Diamond* cases in 2D. (a) Both parents refined. (b) One parent refined. (c) Neither parent refined.

found by checking the midpoints $\mathbf{v}_m$ of its edges such that CONTAINSDIAMOND($\Sigma, \mathbf{v}_m$) is TRUE and also that the third vertex $\mathbf{v}'$ of t is a vertex of this diamond (i.e. it belongs to $R_{\diamond,0}(\delta)$). The complexity of $R_{2,\diamond}$ is constant (with cardinality one).

As an example, consider the three vertices of the red triangle t_{red} in Figure 8.5d. If we test the midpoint of the upper right edge of t_{red}, we find that it is a diamond δ_{green} in Σ (i.e. the one containing the green triangle). However, since the third vertex of t_{red} is not a vertex of δ_{green}, it is not the diamond containing t. In contrast the left edge is a diamond δ_{red} in Σ, whose vertices include the third vertex of t_{red}. so $R_{2,\diamond}(t_{red}) = \{\delta_{red}\}$.

We use a similar approach to extract the *Edge-Diamond* relation $R_{1,\diamond}$ for the diamond(s) incident to a given edge $\mathbf{e}$. Let $\mathbf{v}_e$ be the midpoint of $\mathbf{e}$, and δ_e the diamond whose spine is $\mathbf{e}$. Then $\mathbf{e}$ is either: (a) a spine of a complete diamond δ_e, containing both of its duets (Figure 8.6a), (b) a spine of an incomplete diamond δ_e, containing one of its duets (Figure 8.6b) or (c) not a spine of a diamond in Σ (Figure 8.6c). In the second case, the parent diamond associated with the other duet of δ_e is the second diamond in $R_{1,\diamond}(\mathbf{e})$. In the final case, $R_{1,\diamond}(\mathbf{e})$ consists of both parents of δ_e. The complexity of this operation is constant.

Extraction of the *Vertex-Diamond* relation $R_{0,\diamond}(\mathbf{v})$ for a vertex $\mathbf{v}$ in diamond mesh Σ involves three steps (as illustrated in Figure 8.7 for the center point of the diamond mesh from Figure 8.1).

Step 1: First, we find a single edge $\mathbf{e}$ incident to $\mathbf{v}$, i.e. $\mathbf{e}$ is in the partial relation $R_{0,1}^*(\mathbf{v})$, through a series of edge bisections. Let δ be the diamond whose central vertex is $\mathbf{v}$. Then, based on Property 1, we know that δ is a subdivided diamond, and, thus, an edge was created from $\mathbf{v}$ to the two spine vertices of δ at some previous refinement step. Let us denote these vertices as $\mathbf{v}_{\delta_a}$ and $\mathbf{v}_{\delta_b}$.

As a first approximation to $R_{0,1}^*(\mathbf{v})$, consider the edge $\mathbf{e} := (\mathbf{v}, \mathbf{v}_{\delta_b})$, with midpoint $\mathbf{v}_e$. If $\mathbf{v}_e$ is not a vertex in Σ, then $\mathbf{e}$ is an edge in Σ. Otherwise, $\mathbf{e}$ is not an edge in Σ, so it must have been bisected at some point. Therefore, we can replace the second vertex in $\mathbf{e}$ with $\mathbf{v}_e$ and repeat until we find an edge without a midpoint in Σ (Figure 8.7a).

Step 2: Next, we find the diamonds incident to edge $\mathbf{e}$. This is accomplished via the constant relation $R_{1,\diamond}(\mathbf{e})$ as described above(see Figure 8.7b).

113

| **(a)** Step 1 | **(b)** Step 2 | **(c)** Step 3 |
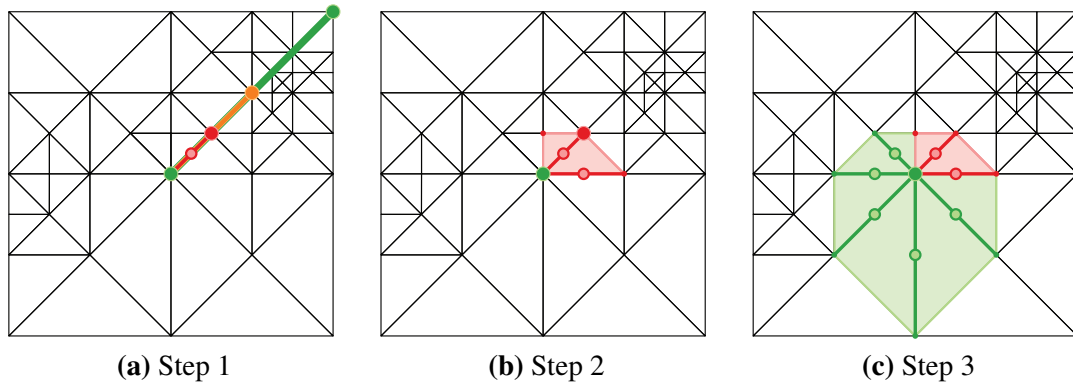
Figure 8.7: The *Vertex-Diamond* relation $R_{0,\diamond}$ for the midpoint $\mathbf{v}$ (green vertex) of a diamond mesh: (a) First an edge (red) in $R_{0,1}^*$ is found. (b) Then, the diamonds (red triangles) in its co-boundary are found using $R_{1,\diamond}$. (c) Finally, its adjacent diamonds (green) are found by iterating on $R_{\diamond,\diamond|\mathbf{v}}$.

Step 3: Finally, we iterate around the *star* of $\mathbf{v}$ by finding all unique diamonds in $R_{\diamond,\diamond|\sigma}(\mathbf{v})$ (see Figure 8.7c).

Together, $R_{0,\diamond}$ is found through a combination of relations $R_{0,1}^*(\sigma)$, $R_{1,\diamond}$ and $R_{\diamond,\diamond|\sigma}$. To analyze its complexity, let the level of $\mathbf{v}$, denoted as $\textsc{Level}(\mathbf{v})$, be the refinement level of diamond δ whose central vertex is $\mathbf{v}$. Then the algorithm for $R_{0,1}^*(\sigma)$ checks at most $N - \textsc{Level}(\mathbf{v})$ vertices, where N is the maximum level of resolution. Since we consider N to be constant for all practical applications, $R_{0,1}^*$ is a constant operation. The second step is constant, since $R_{1,\diamond}$ is a constant relation. Finally, $R_{\diamond,\diamond|\sigma}$ is run once per diamond in $R_{0,\diamond}$, so $R_{0,\diamond}$ is optimally supported.

8.4.4 Deriving the remaining topological relations

By combining the above relations, we can easily define the vertex adjacency and co-boundary relations $R_{0,\{0,1,2\}}(\mathbf{v})$ of a vertex $\mathbf{v}$ in terms of the $R_{0,\diamond}$ relation and the $R_{\diamond,\{0,1,2\}|\mathbf{v}}$ relations. Since all steps are optimally supported, relations $R_{0,\{0,1,2\}}$ are as well.

The only remaining relation is the *Edge-Edge* adjacency relation $R_{1,1}$, which consists of all edges in Σ that are incident to a given edge $\mathbf{e} := (\mathbf{v}_1, \mathbf{v}_2)$ along one of its vertices. This operation can be easily satisfied by merging the results of $R_{0,1}(\mathbf{v}_1)$ and $R_{0,1}(\mathbf{v}_2)$, and is therefore optimal as well.

8.5 Retrieving topological relations on 3D diamond meshes

Many of the techniques developed for 2D diamond meshes in Section 8.4 can be generalized to 3D diamond meshes after restructuring them to handle three-dimensional duets. In this section, we focus on these differences and on the new techniques required for topological navigation on three-dimensional diamond meshes.

Recall from Section 4.5, that 3D duets are defined by two face-adjacent tetrahedra of δ that are generated concurrently during the subdivision of a single parent δ_p of δ. Their common face is defined by the spine of their associated diamond δ, as well as the

114
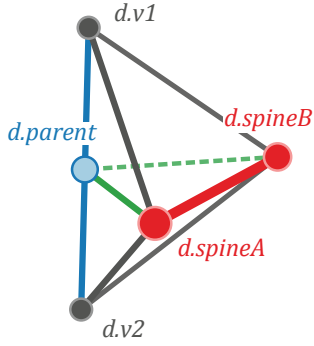
Figure 8.8: Elements of a three-dimensional duet. A 3D duet has five vertices: the two spine vertices *d.spineA* and *d.SpineB* (red), the parent vertex *d.parent* (blue) and the two wing vertices *d.v1* and *d.v2* (gray). It has nine edges: the spine (red), four edges from a spine vertex to a wing vertex (gray), two edges from a spine vertex to the parent vertex (green) and two edges from the parent vertex to the wing vertices (blue). Finally, it has seven triangular faces: the face between the two tetrahedra, four faces incident to the parent vertex and one of the spine vertices and two faces incident to the spine.

central vertex of δ_p (see Figure 8.8). We first observe that neighboring tetrahedra within a diamond have three vertices in common. By connecting the adjacent edges of a diamond's tetrahedra that are not incident to its spine, we obtain a loop, which we refer to as the diamond's *belt*, which provides a linear ordering on the simplices within a diamond (see blue edges in Figure 4.9b).

In particular, when traversing around the belt vertices, every other vertex corresponds to the central vertex of a parent of δ (see Figure 8.2b), and, thus, to a 3D duet. The intermediary vertices, which we refer to as the *wing vertices* of the duets are shared between two adjacent duets within δ.

8.5.1 Boundary relations involving 3D diamonds

As in 2D, the boundary relations are defined in terms of their atomic building blocks, the duets. A duet *d* belongs to a diamond $\delta \in \Sigma$, if its associated parent vertex *d.parent* is in the mesh. Thus, processing a diamond requires 3, 2 or 4 tests on the vertices in Σ for 0-, 1- and 2-diamonds, respectively. One caveat is that, for 2-diamonds on the domain boundary, some of the duets contain only a single tetrahedron, so additional checks on the wing vertex might be necessary.

The *Diamond-Vertex* relation $R_{\diamond,0}(\delta)$ of a diamond δ in Σ always contains the spine vertices of δ. Additionally, for each duet in Σ, the parent vertex and both wing vertices belong to $R_{\diamond,0}$. The result of a $R_{\diamond,0}$ query is a set of vertices of cardinality at most 8, 6 or 10 (depending on the diamond class). Similarly, the *Diamond-Edge* relation $R_{\diamond,1}$ and the *Diamond-Triangle* relation $R_{\diamond,2}$ are satisfied by adding all edges and triangles, respectively, from the duets of δ that are in the mesh, and can be answered in terms of vertex tuples. $R_{\diamond,1}$ has a constant cardinality of at most 19, 13 or 25 elements (i.e the spine plus three edges per tetrahedron in δ), while $R_{\diamond,2}$ has a constant cardinality of at most 18, 12 or 24 elements (three unique faces per tetrahedron in δ).

As in the 2D case, the *Diamond-Tetrahedron* relation $R_{\diamond,3}$ can either be answered as a set of vertex tuples, or as a *simplex bit flag*, where each bit corresponds to a tetrahedron in δ, and the tetrahedra are ordered according to the traversal of the diamond's belt vertices. Since there are at most eight tetrahedra in a diamond, a single byte is sufficient to satisfy this query.

To analyze the constrained diamond boundary relations $R_{\diamond,q|\sigma}(\delta)$ subject to incidence with a *q*-simplex σ in δ's boundary, we need to consider the types of vertices, edges and
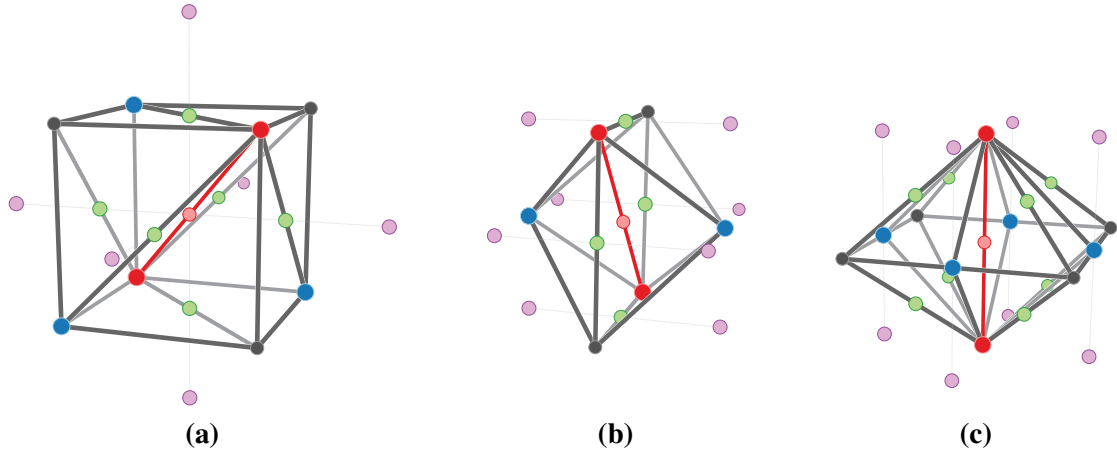
Figure 8.9: Tests to find the diamonds adjacent to a given 3D diamond. (a) 0-diamond (b) 1-diamond (c) 2-diamond. Vertex colors indicate spine (red), parent (blue), child (green) and neighbor (purple).

faces that exist in a duet d of a diamond δ (see Figure 8.8). A duet d has: (a) two spine vertices (*d.spineA* and *d.spineB*); (b) two wing vertices (*d.v1* and *d.v2*); and (c) one parent vertex (*d.parent*). Spine vertices are common to all duets in δ, while wing vertices are common to a pair of adjacent duets in δ. The parent vertex belongs to only one duet in δ.

The four categories of edges within a duet are: (a) the spine edge; (b) the four edges containing a spine vertex and a wing vertex; (c) the two edges containing a spine vertex and the parent vertex; and (d) the two edges containing the parent vertex and a wing vertex. The first category is common to all duets in δ, while the second category is shared between an adjacent pair of duets in δ. The final two categories belong to only a single duet in δ. Note that an edge can never be bounded by both wing vertices of a duet.

Similarly, the three categories of faces in a duet are: (a) one face containing both spine vertices and the parent vertex; (b) four faces containing a spine vertex, a wing vertex and the parent vertex; and (c) two faces containing both spine vertices and a wing vertex. The first and second categories belongs to only one duet in δ, while the third category is shared by two duets in δ.

Thus, if the incident simplex σ includes at least one of the spine vertices, all duets are involved in the query, although, some boundary relations involving its other spine vertex can be safely omitted. Otherwise, the simplex σ can be incident to only one or two duets in δ.

8.5.2 Adjacency relations involving 3D diamonds

As was the case in 2D, the neighbors of a duet in Σ along the spine of δ either belong to δ or to a parent of δ, while those along the exterior faces can belong to a child of δ or to a parent of the child that is at the same refinement depth as δ.

Our *Diamond-Diamond* adjacency query $R_{\diamond,\diamond}$ utilizes the general algorithm provided in Algorithm 8.1. However, we must redefine appropriate *childPt* and *neighborPt* locations depending on the class of the diamond δ (see Figure 8.9).

For 0-diamonds, each duet d has three *childPt* vertices, which are located at the midpoint of the three cube faces incident to *d.parent*. Their locations are:*

$$ChildPointSet = \begin{cases} \frac{1}{2}(d.spineB + d.parent) \\ \frac{1}{2}(d.spineA + d.v1) \\ \frac{1}{2}(d.spineA + d.v2). \end{cases}$$

Each *childPt* has a corresponding

$$neighborPt = \mathbf{v}_c + 2(childPt - \mathbf{v}_c).$$

A duet d of a 1- or 2-diamond, has four *childPt* vertices

$$ChildPointSet = \begin{cases} \frac{1}{2}(d.spineA + d.v1) \\ \frac{1}{2}(d.spineA + d.v2) \\ \frac{1}{2}(d.spineB + d.v1) \\ \frac{1}{2}(d.spineB + d.v2). \end{cases}$$

Each *childPt* of a 1-diamond has a corresponding

$$neighborPt = childPt + (d.parent - \mathbf{v}_c),$$

while, the two *childPt*s of a 2-diamond corresponding to each spine vertex share the same *neighborPt*. Thus, these duets have only two unique *neighborPt*s:

$$neighborPt \in \begin{cases} d.spineA + (d.parent - \mathbf{v}_c) \\ d.spineB + (d.parent - \mathbf{v}_c). \end{cases}$$

The implementation of the constrained diamond-adjacency relations $R_{\diamond,\diamond|\sigma}$ follows a similar analysis as the constrained diamond-boundary relations $R_{\diamond,i|\sigma}$ (see Section 8.5.1).

8.5.3 Co-boundary relations involving 3D diamonds

The *Tetrahedron-Diamond* co-boundary relation $R_{3,\diamond}$ is similar to the 2D case as well. Each tetrahedron $T \in \Sigma$ uniquely maps to a single diamond whose spine coincides with an edge of T. However, in the 3D case, both non-spine vertices of T must belong to the diamond for the correct results.

The analysis of the *Triangle-Diamond* relation $R_{2,\diamond}$ in 3D is similar to that of the *Edge-Diamond* relation in 2D, and its cardinality is either one or two. Each triangle t either contains the spine of its incident diamond(s) or it is incident to two diamonds whose common child's spine belongs to t, thus, we can satisfy the $R_{2,\diamond}$ relation in constant time by checking t's three edges.

---

*This formulation assumes the *d.spineA* is a corner vertex of its associated supercube's domain, while *d.spineB* is at the midpoint of its domain (see Section 6.1.4).

The *Edge-Diamond* relation $R_{1,\diamond}$ in 3D is more complicated than its 2D analogue (but is still constant). Recall from Section 8.2 that edges can exist in a mesh Σ for approximately d subdivision steps. Thus, we satisfy this query in two steps. We first find one diamond in $R_{1,\diamond}^*$ of an edge $\mathbf{e}$, and then iterate on the $R_{\diamond,\diamond|\mathbf{e}}$ to find all remaining diamonds. The second step is implemented in the same way as the 2D *Vertex-Diamond* relation $R_{\diamond,\diamond|\mathbf{v}}$ for a vertex $\mathbf{v}$ (see Section 8.4.3).

Let δ_e be the diamond whose spine is $\mathbf{e}$ (which is not necessarily in Σ). A diamond incident to $\mathbf{e}$ can be found by considering the ancestors of δ_e that are at most four subdivisions higher, and returning the first such ancestor in Σ that contains $\mathbf{e}$. Despite the hierarchical traversal, this step is still constant due to the structure of the hierarchy. In practice, we found that about 66% of the cases require only one or two tests, and approximately 30% require three to six tests. We have not observed any cases where an $R_{1,\diamond}^*$ query required more than seven tests.

The final relation to consider is the *Vertex-Diamond* relation $R_{0,\diamond}$. Although the cardinality of this relation is higher in 3D, the query is identical to that of the 2D case. We first find an edge $\mathbf{e}$ in the partial relation $R_{0,1}^*(\mathbf{v})$. We then find a diamond δ in the partial relation $R_{1,\diamond}^*(\mathbf{e})$ as described above. Finally, we iterate through the $R_{\diamond,\diamond|\mathbf{v}}$ relation until all diamonds in the co-boundary are found. This is accomplished through a graph traversal that marks visited diamonds as we traverse the adjacent diamonds. Since the first two steps are constant, and the final step is optimal in the cardinality of $R_{0,\diamond}(\mathbf{v})$, this operation is optimally supported.

The remaining topological operations are also optimally supported, and can be defined in terms of the co-boundary $R_{p,\diamond}$ relations coupled with the constrained diamond-boundary relations $R_{\diamond,q\,|\,\sigma}$. For example, the *Face-Face* relation $R_{2,2}$ finds all triangles adjacent to a given triangle along its edges. This can be satisfied for a triangle t by first finding a diamond δ_t incident to t, i.e. using a constant query on $R_{2,\diamond}$. Next, we find the three edges in the boundary of t using the optimal $R_{\diamond,1\,|\,t}\,(\delta_t)$ relation. Finally, using δ_t to initialize $R_{1,\diamond}^*$, we find the $R_{1,2}$ relation for each of the three edges by combining $R_{1,\diamond}$ and $R_{\diamond,2}$ and merge the results.

8.6 Results

In this section, we discuss representations for encoding diamond meshes, and compare the storage costs to a simplex-based representation of RSB meshes [LDS01] and to the extended IA data structure [PBCF93] defined for general simplicial complexes over a manifold domain.

As discussed in Section 8.3, the primary requirements for our encoding is that we have a valid diamond mesh defined over a *closed* set of vertices from a regular grid of resolution ($2^N + 1$) along each axis and that we can efficiently detect the presence or absence of vertices and diamonds in the mesh. Furthermore, many applications of these meshes encode scalar values at the vertices of the grid.

Thus, we can store the vertices and diamonds in the mesh using a diamond-based approach or using a supercube-based approach, which we can generate from the active front representation of Section 6.3 by iterating through the *Diamond-Vertex* relationships.

That is, for each diamond δ in the active-front based representation, (a) add the vertices in $R_{\diamond,0}(\delta)$ to *VertexSet* by iterating through its duets; (b) add the central vertex of δ to *DiamondSet*.

Assuming that each coordinate can be encoded using two bytes of memory, the cost of this data structure for a diamond meshes with $|V|$ vertices (each with an associated scalar value) and $|\delta|$ diamonds is: $6 \cdot |V| + 4 \cdot |\delta|$ bytes for a 2D mesh and $8 \cdot |V| + 6 \cdot |\delta|$ bytes for a 3D mesh.

Since we only require support for testing the presence or absence of the encoded grid points bitflag-based supercube representations for the vertices and diamonds can provide efficient support for topological navigation on diamond meshes. In 2D, the cost of representing each supercube in the vertex set is 10 bytes: (a) 4 bytes to encode the spatial coordinates; (b) 2 bytes to encode the 12 bit flags; and (c) 4 bytes to encode the start index of the scalar values in a global vertex array. Similarly, since diamonds do not have any attached values, each diamond supercube would require only 6 bytes: (a) 4 bytes to encode the spatial coordinates; and (b) 2 bytes to encode the 12 bit flags. Additionally, the cost of encoding the scalar values is $2 \cdot |V|$. Thus, in 2D the cost of a supercube-based representation is $2 \cdot |V| + 10 \cdot |S_v| + 6 \cdot |S_\delta|$ bytes.

In 3D, the spatial coordinates require an additional two bytes and the bitflags require and additional five bytes (i.e. seven bytes are required to encode the 56 bits rather than two bytes in 2D), so the cost of a supercube representation is: $2 \cdot |V| + 17 \cdot |S_v| + 13 \cdot |S_\delta|$ bytes.

The simplex-based encoding of [LDS01] utilizes *bintree-based location codes* (see Section 6.3.1) for simplices stored in a forest of binary trees. The vertices can be encoded through a hash table as in the diamond-based approach above, and simplices require six bytes to index their location code. Thus, the difference in storage requirements between this representation and the diamond-based representation is proportional to the average number of triangles or tetrahedra per diamond in the mesh. The cost of this data structure is: $6 \cdot |V| + 6 \cdot |t|$ bytes for a 2D mesh with $|t|$ triangles and $8 \cdot |V| + 6 \cdot |T|$ bytes for a 3D mesh with $|T|$ tetrahedra.

As a final comparison, we consider the storage cost incurred by the *extended Indexed data structure with Adjacency (IA)* data structure in 2D and 3D [PBCF93]. This data structure is among the most compact topological data structures for general manifold simplicial complexes [DH07]. It encodes an array of $|V|$ vertices as well as $|t|$ triangles or $|T|$ tetrahedra, in 2D and 3D, respectively. Each vertex requires 10 bytes: six bytes for the spatial coordinates and four bytes to encode the index of a single tetrahedron in its star. Each triangle requires 24 bytes: twelve bytes to encode the indices of its three vertices and twelve bytes to encode the indices of its adjacent triangles. The cost of the IA in 2D is therefore $10 \cdot |V| + 24 \cdot |T|$ bytes. In 3D, each vertex requires two additional bytes for the extra coordinate, and eight additional bytes per tetrahedron to encode the extra vertex index and adjacent tetrahedron, for a total storage cost of $12 \cdot |V| + 32 \cdot |T|$ bytes.

Table 8.1 summarizes the storage costs for the 2D and 3D representations.

Table 8.2a provides representative experimental results on the number of vertices, tetrahedra, diamonds, in some adaptive 3D diamond meshes extracted from a hierarchy of diamonds at uniform approximation error (column 2) from several volume datasets, as well as the number of supercubes for the vertices and diamonds of the mesh. The average

Table 8.1: Storage costs (in bytes) of 2D data structures (a) and 3D data structures (b) based on the extended Indexed data structure with Adjacencies (IA), as well as the simplex-based, diamond-based and supercube-based representations for conforming RSB meshes consisting of $|V|$ vertices, $|\sigma|$ top simplices and $|\delta|$ diamonds. For the supercube-based representation, $|S_V|$ is the number of supercubes required to encode the vertices (i.e. a partial DMSF) and $|S_\delta|$ is the number of supercubes required to encode the diamonds.

(a) Storage costs of 2D data structures

2D Data Structure	Cost of vertices		Cost of cells
Extended IA	$10 \cdot |V|$	+	$24 \cdot |\sigma|$
Simplex-based RSB	$6 \cdot |V|$	+	$6 \cdot |\sigma|$
Diamond-based RSB	$6 \cdot |V|$	+	$4 \cdot |\delta|$
Supercube-based RSB	$2 \cdot |V| + 10 \cdot |S_V|$	+	$6 \cdot |S_\delta|$

(b) Storage costs of 3D data structures

3D Data Structure	Cost of vertices		Cost of cells
Extended IA	$12 \cdot |V|$	+	$32 \cdot |T|$
Simplex-based RSB	$8 \cdot |V|$	+	$6 \cdot |\sigma|$
Diamond-based RSB	$8 \cdot |V|$	+	$6 \cdot |\delta|$
Supercube-based RSB	$2 \cdot |V| + 17 \cdot |S_V|$	+	$13 \cdot |S_\delta|$

ratios of simplices to diamonds ($|\sigma|/|\delta|$) tends to increases with the resolution from around 3 to 4 while the number of diamonds per supercube ($|\delta|/S_\delta$) oscillates between 8 to 10, and the number of tetrahedra per supercube increases from around 22 to 39. The number of vertices per supercube increases as the error decreases, since the vertices of the mesh correspond to the refined ancestors of the diamonds in the mesh. From Table 8.2b, we see that the simplex-based RSB representation requires around 2-3 times as much space as the diamond-based representation, and the diamond-based representation requires around 3.5 times as much space as the supercube representation. Compared to the general IA data structures, the RSB representations are an order of magnitude more compact. The IA requires around ten times the storage space as the diamond-based representation, and 30-45 times as much space as the supercube-based representation. In the lossless approximations (0% error), the IA data structure requires more than 1 GB, while the supercube-based representation requires only 25 MB. Besides the storage space, the adjacency and partial vertex co-boundary relations must also be generated for the IA data structure.

Another interesting property of these meshes is the average cardinality of the vertex star and edge star operations which are among the most useful operations for navigation on these meshes. In fact, the primary objective of the neighbor-finding scheme of [LDS01] relates to finding the star of each bisection edge for efficient conforming updates to a simplex-based RSB mesh. Due to properties of the decomposition scheme, the maximum possible cardinality of $R_{0,3}$ on 3D diamond meshes is 48, while that of $R_{1,3}$ is 8 [WD10d]. As we can see from Table 8.3, the average cardinality of the Vertex-Tetrahedra co-boundary

Table 8.2: (a) Number of vertices ($|V|$), tetrahedra ($|\sigma|$), diamonds ($|\delta|$), and supercubes for vertices ($|S_V|$) and diamonds ($|S_\delta|$) in some adaptive diamond meshes extracted at uniform approximation error from several volumetric datasets. Also listed are the average number of vertices per super-cube in the *VertexSet* ($|V|/|S_V|$), tetrahedra per diamond ($|\sigma|/|\delta|$), diamonds per supercube in the *DiamondSet* ($|\delta|/|S_\delta|$) and tetrahedra per supercube in the *DiamondSet* ($|\sigma|/|S_\delta|$). (b) A comparison of the storage requirements for extended IA, Simplex-based, Diamond-based and Supercube-based representations using the statistics from (a) and the calculations from Table 8.1b. Also listed are some relative storage sizes of the Simplex, Diamond, Supercube and IA data structures. Volumetric datasets courtesy of [Vol].

(a) Statistics for topological data structures in 3D

Dataset	Error	$|V|$	$|S_V|$	$|\sigma|$	$|\delta|$	$|S_\delta|$	$|V|/|S_V|$	$|\sigma|/|\delta|$	$|\delta|/|S_\delta|$	$|\sigma|/|S_\delta|$
	50%	1.6 K	139	8.6 K	3.3 K	397	11.9	2.6	8.3	21.7
	30%	12.5 K	1.03 K	68 K	24.7 K	2.84 K	12.1	2.8	8.7	24.0
	10%	254 K	18.9 K	1.40 M	515 K	49.9 K	13.5	2.7	10.3	28.1
Visible Human Head	5%	620 K	38.4 K	3.40 M	1.22 M	128 K	16.1	2.8	9.6	26.7
($128 \times 256 \times 256$)	2%	1.21 M	53.5 K	6.66 M	2.29 M	267 K	22.7	2.9	8.6	24.9
	1%	1.71 M	69.4 K	9.46 M	3.06 M	350 K	24.6	3.1	8.7	27.0
	0.5%	2.55 M	94.2 K	14.2 M	4.37 M	504 K	27.0	3.1	8.7	28.3
	0%	5.47 M	139 K	31.3 M	7.79 M	963 K	39.4	4.0	8.1	32.5
	50%	19 K	1.54 K	103 K	37.7 K	4.24 K	12.3	2.7	8.9	24.3
	30%	82.4 K	6.72 K	453 K	161 K	17.3 K	12.3	2.8	9.3	26.2
	10%	1.14 M	73.1 K	6.33 M	2.17 M	237 K	15.6	2.9	9.1	26.7
Foot	5%	2.80 M	102 K	15.1 M	4.59 M	625 K	27.5	3.3	7.3	24.2
($256 \times 256 \times 256$)	2%	4.48 M	125 K	25.0 M	6.31 M	810 K	35.8	4.0	7.8	30.9
	1%	5.17 M	138 K	30.0 M	6.79 M	873 K	37.3	4.4	7.8	33.9
	0%	5.90 M	151 K	34.7 M	7.03 M	898 K	39.2	4.9	7.8	38.7

(b) Storage costs for topological data structures in 3D

Dataset	Error	IA	Simplex	Diamond	Supercube	$\frac{\text{Simplex}}{\text{Diamond}}$	$\frac{\text{Diamond}}{\text{Supercube}}$	$\frac{\text{IA}}{\text{Diamond}}$	$\frac{\text{IA}}{\text{Supercube}}$
	50%	0.28 MB	0.06 MB	0.03 MB	0.01 MB	2.0 x	3.0 x	8.9 x	27.2 x
	30%	2.2 MB	0.48 MB	0.24 MB	0.08 MB	2.0 x	3.2 x	9.4 x	29.3 x
	10%	45.7 MB	9.97 MB	4.89 MB	1.41 MB	2.0 x	3.5 x	9.4 x	32.4 x
Visible Human Head	5%	111 MB	24.2 MB	11.7 MB	3.39 MB	2.1 x	3.5 x	9.5 x	32.8 x
($128 \times 256 \times 256$)	2%	217 MB	47.4 MB	22.3 MB	6.50 MB	2.1 x	3.4 x	9.7 x	33.5 x
	1%	308 MB	67.2 MB	30.6 MB	8.73 MB	2.2 x	3.5 x	10.1 x	35.3 x
	0.5%	308 MB	67.2 MB	30.6 MB	12.6 MB	2.3 x	3.5 x	10.4 x	36.7 x
	0%	0.99 GB	221 MB	86.3 MB	24.6 MB	2.6 x	3.5 x	11.8 x	41.3 x
	50%	3.37 MB	0.74 MB	0.36 MB	0.36 MB	2.0 x	3.2 x	9.3 x	29.6 x
	30%	14.8 MB	3.2 MB	1.55 MB	0.48 MB	2.1 x	3.2 x	9.5 x	30.7 x
	10%	206 MB	45.0 MB	21.1 MB	6.31 MB	2.1 x	3.3 x	9.8 x	32.7 x
Foot	5%	494 MB	108 MB	47.7 MB	14.7 MB	2.3 x	3.2 x	10.4 x	33.5 x
($256 \times 256 \times 256$)	2%	815 MB	177 MB	70.3 MB	20.6 MB	2.5 x	3.4 x	11.6 x	39.5 x
	1%	962 MB	209 MB	78.3 MB	22.9 MB	2.7 x	3.4 x	12.3 x	42.0 x
	0%	1.1 GB	244 MB	85.3 MB	24.8 MB	2.9 x	3.4 x	13.2 x	45.4 x

Table 8.3: Number of vertices ($|V|$), tetrahedra ($|\sigma|$) and diamonds ($|\delta|$) and the average cardinality of the *Vertex-Tetrahedra* $R_{0,3}$, *Vertex-Diamond* $R_{0,\diamond}$, *Edge-Tetrahedra* $R_{1,3}$ and *Edge-Diamond* $R_{1,\diamond}$ relations for three-dimensional diamond meshes.

Dataset	resolution	$|V|$	$|\sigma|$	$|\delta|$	$|R_{0,3}|$	$|R_{0,\diamond}|$	$|R_{1,3}|$	$|R_{1,\diamond}|$
	30%	506	2.3 K	970	18.2	10.3	4.66	3.29
Fuel	10%	2.4 K	12.5 K	4.2 K	21.1	10.5	4.96	2.99
	0%	19.9 K	115 K	27.8 K	23.0	9.36	5.09	2.99
	10%	450	2.28 K	926	20.28	11.2	4.9	3.31
Hydrogen	1%	5.92 K	31.4 K	11.2 K	21.2	10.08	4.98	3.31
	0%	545 K	3.06 M	885 K	22.45	10.27	5.06	3.4

relation $R_{0,3}$ is about twice that of the Vertex-Diamond co-boundary relation $R_{0,\diamond}$. While the cardinality Edge-Tetrahedra co-boundary relation $R_{1,3}$ is about 50% greater than that of the Edge-Diamond co-boundary relation $R_{1,\diamond}$. Due to the great deal of adaptivity of these meshes, the numbers most likely reflect the fact that pairs of tetrahedra in the star are grouped to the same diamond via the 3D duet construct.

8.7 Discussion

In this chapter, we introduced optimal algorithms for topological navigation on two- and three-dimensional diamond-based RSB meshes. These algorithms exploit the structure of the mesh to query the topological connectivity of the mesh without requiring explicit generation or storage of any topological relations. Compared to a general adjacency-based data structure, our diamond-based and supercube-based representations for diamond meshes require an order of magnitude less space while still supporting optimal queries on the local topological connectivity of the mesh.

Since the template that we use for extracting the co-boundary relations depends only on the $R_{0,1}^*$ relation, which is dimension independent, and the $R_{1,\diamond}^*$ relation, whose complexity depends only on the mesh dimension, but not the dataset complexity, we anticipate extending these relations to higher dimensional diamond meshes. This can be useful, for example, in analyzing time-dependent volumetric datasets. In this case, we must analyze the properties of the 4D (and higher dimensional) duets, which can be defined by a variable number of top-simplices (see Table 4.5). On the other hand, traversal of a diamond's duets should be even more efficient than a simplex-based approach since the number of top simplices within a diamond is $O(d!)$, while the number of parents (i.e. the number of duets) is $O(d)$.

Chapter 9

Isodiamond hierarchies

Due to the size of isosurfaces or interval volumes extracted from volume data sets, simplified representations for such structures can greatly aid in their analysis and visualization. These simplified representations are usually obtained by applying a local mesh coarsening operator, such as an edge collapse, to the full resolution mesh describing the isosurface or interval volume.

However in scientific and medical applications, details at the highest available resolution are required on demand, and thus, simplified approximations of these datasets are not sufficient. Therefore, it is often desirable to have a multiresolution representation of a *specific* isosurface or interval volume from which simplified adaptive representations can be efficiently extracted on demand.

When a multiresolution isosurface or interval volume is extracted from a multiresolution model of the underlying scalar field, its structure should be coherent with the model of the scalar field. In other words, a natural multiresolution representation for an isosurface or an interval volume is defined by the intersection of the former with the atomic modifications in the multiresolution model of the field. The resulting multiresolution model clearly inherits the dependency relation from the dependency graph of the multiresolution field model.

We consider here the problem of defining multiresolution models of isosurfaces, and interval volumes, when the underlying multiresolution field model is defined by a hierarchy of diamonds. As shown in Chapter 6, the regularity of the vertex distribution of a hierarchy of diamonds enables a very compact encoding which we exploit to produce effective and compact multiresolution models for isosurfaces and interval volumes. Specifically, each local modification to the isosurface or interval volume intersecting a specific diamond δ has a one-to-one correspondence with the modification associated with δ. Since diamonds are defined on a regular grid, they are much simpler to encode than the modifications to the general triangle or tetrahedral mesh representing the isosurface or interval volume.

Note that, although our descriptions and experiments in this chapter focus on the three dimensional case, the isodiamond hierarchy framework can be generalized in a dimension-independent manner.

9.1 Isodiamonds

The basic idea in defining a multiresolution model for an isosurface S, or interval volume I, extracted from a hierarchy of diamonds Δ consists of considering only a subset of the diamonds in Δ and possibly the intersection of S or I with such diamonds. We call the diamonds in this multiresolution model *isodiamonds*, and the intersection of an isodiamond with S or I an isosurface or interval volume *patch*, respectively. Each patch

is triangulated based on the values of the sign field associated with the corresponding diamond in Δ (see Section 2.3.1). We call the ordered set of such sign values the *bit pattern* of the diamond.

Since we are interested in the ability to reconstruct the isosurface S, or interval volume I, at intermediate uniform or variable-resolutions, we need to include in the model all isodiamonds with non-empty patches, that we call *active isodiamonds*. However, since S or I depends on the range of the scalar field rather than its domain, we require a spatial index on the active isodiamonds. The latter is obtained by considering the *relevant isodiamonds*, which are the ancestors of active isodiamonds that have empty patches. An important subset of the relevant isodiamonds are the *creation isodiamonds*, which create a new topological component of S or I upon subdivision. The remaining isodiamonds are *inactive* with respect to S or I.

More formally,

- an *active isodiamond* δ is an isodiamond such that both δ and the associated subdivided isodiamond δ_s contain at least one active tetrahedron; thus, both δ and δ_s are intersected by the isosurface or interval volume (see Figure 9.2b).

- a *creation isodiamond* is an isodiamond δ which does not intersect the isosurface S or the interval volume I, but the tetrahedra of the associated subdivided isodiamond δ_s are all active (see Figure 9.3).

- a *relevant isodiamond* is an isodiamond δ which does not intersect the isosurface S or interval volume I but at least one of its descendants intersects either S or I.

Figure 9.1a illustrates the various isodiamond types on a small isodiamond hierarchy whose underlying hierarchy of diamonds Δ covers a square 2D domain and has the dependency graph depicted in Figure 9.1b. For simplicity, we only show the unsubdivided diamonds. The central vertices of active, relevant, creation and inactive isodiamonds are indicated by green, blue and red and white vertices at their spine centers, respectively. The top row describes the single (unsubdivided) isodiamond corresponding to the root of Δ. Since it intersects the isosurface, it is an active isodiamond (indicated with a green central vertex). The next row describes the four children of the root diamond in Δ. Two of these are active isodiamonds (green), and the other two are relevant isodiamonds (blue). The third row shows the four children of those at level two, of which, two are active isodiamonds, one is a relevant isodiamond and one is a creation isodiamond. As illustrated in the final row, the patches generated by subdividing the creation isodiamond form a new isosurface component. Figure 9.4 illustrates the isodiamond types of the 2D Bonsai tree dataset (with isovalue $\kappa = 58$), by coloring the central vertices as above.

We have developed two multiresolution models for isosurfaces and interval volumes extracted from a diamond hierarchy. The *Relevant Isodiamond (RI)* hierarchy (described in Section 9.3) closely follows the hierarchy of diamonds encoding the underlying field since it represents both the active isodiamonds and the relevant isodiamonds. Thus, a modification in an RI hierarchy corresponds to a diamond δ in the associated diamond hierarchy Δ such that the range of the field values within δ contains the isovalue κ defining the isosurface, or the isorange K defining the interval volume.

(a) Isodiamonds   **(b)** HD dependency graph **(c)** RI dependency graph **(d)** MI dependency graph
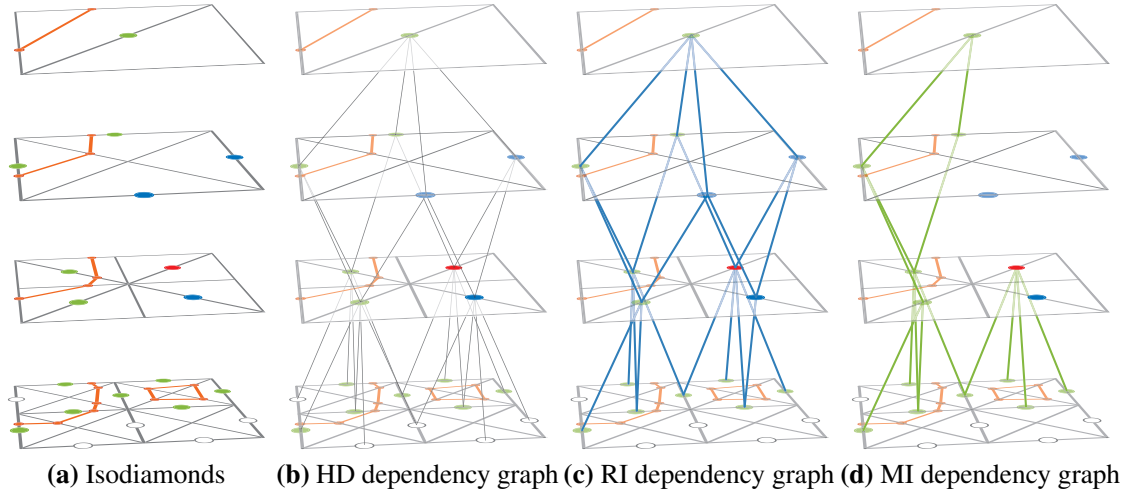
Figure 9.1: Isodiamonds (a) and associated dependency graphs (b-d) for a small 2D isodiamond hierarchy. *Active isodiamonds* (green central vertices) intersect the isosurface (orange lines). *Relevant isodiamonds* (blue central vertices) are empty ancestors of the active isodiamonds. *Creation isodiamonds* (red central vertices) are relevant isodiamonds that create a new topological component after subdivision. All other isodiamonds are inactive (white central vertices). The dependency graph of the MI hierarchy (d) is a subgraph of the RI hierarchy's dependency graph (c), which, in turn, is a subgraph of the HD's dependency graph (b).

In contrast, the *Minimal Isodiamond (MI)* hierarchy (described in Section 9.4) just contains the modifications that intersect the isosurface S or interval volume I. Since the relevant isodiamonds do not intersect S or I, and mainly serve as a spatial access structure for the active and creation isodiamonds, they are not strictly necessary to reconstruct S or I. The key novelty in the MI model is its ability to extract simplified conforming meshes describing the isosurface, or interval volume, without storing the relevant isodiamonds.

9.2 Encoding isodiamonds

Due to the one-to-one correspondence between isodiamonds and diamonds in the hierarchy of diamonds, each patch in the RI or MI hierarchy is encoded (using a marching cells rule)

(a) Diamond modification   **(b)** Isodiamond modification
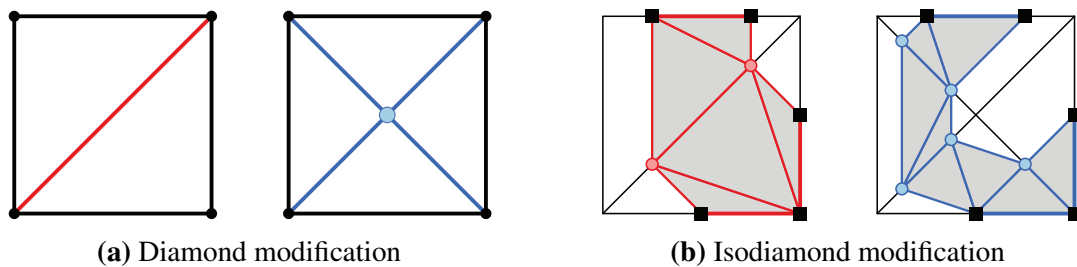
Figure 9.2: Modifications associated with an isodiamonds in an isodiamond hierarchy (b) corresponds to modifications of diamonds in the diamond hierarchy (a).
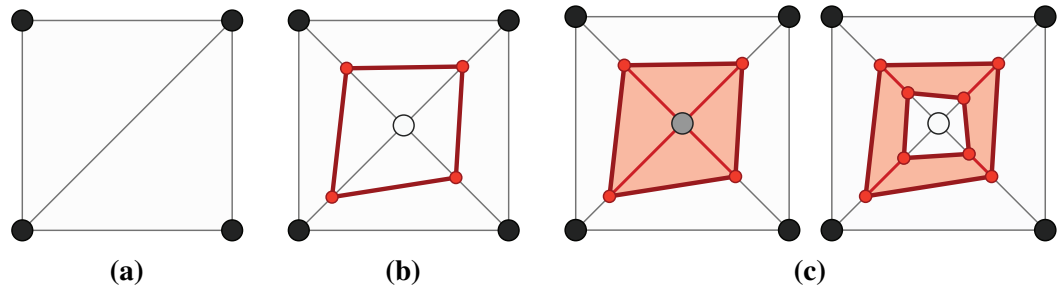
Figure 9.3: All possible creation isodiamond modifications. The unsubdivided isodiamond δ (a) and the subdivided isodiamond δ_s for an isosurface (b) or for an interval volume (c, two cases).
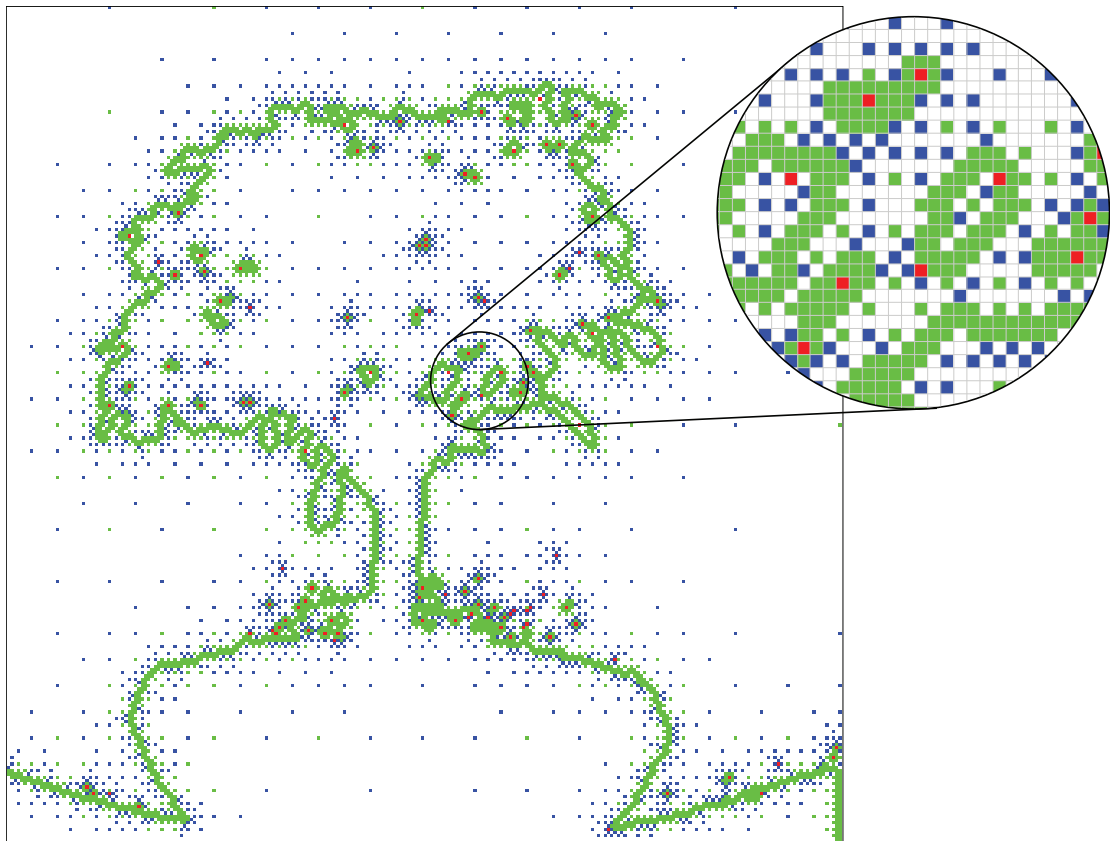
Figure 9.4: Isodiamond types for 2D bonsai tree dataset ($\kappa = 58$). The color of the central vertex of a diamond indicates its corresponding isodiamond type. Of the 257^2 diamonds, there are 5,242 *active* isodiamonds (green), 123 *creation* isodiamonds (red) and 2,877 *relevant* isodiamonds (blue).

through:

- the bit pattern of the corresponding diamond and

- the intersection vertices of the corresponding patch, which we call *isovertices*.

Each patch can then be triangulated via a lookup table on the bit pattern, by using the isovertices.

Since diamond modifications are local and only affect the interior vertices and edges of the diamond, (see Section 4.4.1), we may assume that the bit pattern and isovertices on the boundary of an isodiamond have already been encoded in ancestor isodiamonds. Thus, the patch corresponding to a subdivided isodiamond δ_s can be reconstructed from the patch corresponding to δ using only

- the sign value of the central vertex of δ_s and

- the isovertices for each active subdivision edge of δ_s.

Figure 9.2 illustrates a modification to an interval volume patch (in 2D) corresponding to the subdivision of an active isodiamond δ. Observe that the modification removes the isovertices on the spine of δ (red vertices in Figure 9.2b) and inserts new isovertices along its active subdivision edges (blue vertices in Figure 9.2b). The patch is then retriangulated using the new isovertices. Patch vertices and edges intersecting the diamond's boundary (black squares in in Figure 9.2b) are unaffected by the modification.

Recall that, under linear interpolation, all patch isovertices lie along edges of tetrahedra. Thus, rather than encoding each isovertex using its (x, y, z) position, we use an interpolation coefficient along the unique subdivision edge containing the vertex. For compactness, we quantize each isovertex to a single byte. This is an appropriate compromise between storage space and accuracy, since the lengths of the diamond edges shrink as their level increases.

Based on the bit pattern associated with the isodiamond, a lookup table can be used to determine the number of isovertices introduced by the modification as well as an (ordered) list of indices for the active subdivision edges. To maintain constant-sized modifications, the isovertices are stored in a global *isovertex array* and only the index of the first isovertex is encoded with each modification.

Note that the encoding of an isodiamond does not include the scalar values of the original volume data set, but only the sign field of the vertices of the data set. The efficiency of the resulting data structure derives from exploiting the regular spatial decomposition and the implicit dependency relation induced by the hierarchy of diamonds representation, while only encoding the modifications that relate to the specific isosurface or interval volume.

9.3 Relevant isodiamonds

In this section, we present the Relevant Isodiamond (RI) hierarchy for an isosurface S or an interval volume I. We describe first the data structure encoding it (Section 9.3.2). Next, we review the algorithm for generating an RI hierarchy from a hierarchy of diamonds

(Section 9.3.3). Finally, we discuss how variable-resolution isosurfaces or interval volumes can be extracted from an RI hierarchy through selective refinement (Section 9.3.4).

9.3.1 Definition

A *Relevant Isodiamond (RI)* hierarchy, that we denote as Δ_R, is defined by the subset of modifications of the corresponding hierarchy of diamonds Δ that are *active* or *relevant* with respect to the isosurface S or interval volume I. The dependency relation in Δ_R is defined as the restriction of Δ's dependency relation to the active and relevant isodiamonds. Thus, every relevant and active isodiamond will have in Δ_R the same parents as the corresponding diamond in Δ but some of the children isodiamonds might be missing in Δ_R (as compared to Δ), if they are neither active nor relevant. Figure 9.1c shows the dependency graph of the RI hierarchy Δ_R associated with the multiresolution isosurface representation of Figure 9.1a. The graph describing Δ_R contains a subset of the arcs of Δ's dependency graph (see Figure 9.1b), where the missing arcs have endpoints corresponding to *inactive* isodiamonds (white central vertices).

It can be easily seen that the dependency graph describing Δ_R is a connected subgraph of the one describing Δ and has the same root, which can be an active or a relevant isodiamond. Although the relevant isodiamonds do not intersect S or I, and thus their associated patches are empty, they are used here to guarantee the transitive closure of the dependency relation (as discussed in Section 2.4).

9.3.2 Data structure

The patch of S or I corresponding to the base mesh is encoded through the sign field of the eight corner vertices of the cubic field domain and through interpolation coefficients for all the isovertices along the active edges of the root diamond.

All isovertices introduced during a modification are located along the active subdivision edges of its associated subdivided isodiamond δ_s, and are stored in the isovertex array as interpolation coefficients. For interval volumes, each subdivision edge can be active with respect to the lower surface and/or the upper surface, and thus can generate at most two isovertices. Since isovertices between the two surfaces of I coincide with the vertex of a diamond they are implicitly encoded. Thus, only the isovertices of an interval volume's upper and lower surfaces need explicit encoding. Since the dependency relation in Δ_R is inherited from the dependency relation of Δ, we can compute all parents and children from the coordinates of the central vertex of the isodiamonds.

With each modification $u = (\delta, \delta_s)$ in an RI hierarchy, we encode the following information, thus using 12 bytes per modification:

- central vertex $\mathbf{v}_c$ of the isodiamond δ (encoded using six bytes as three unsigned shorts);

- sign value of $\mathbf{v}_c$ (encoded using one bit for isosurfaces or two bits for interval volumes);

- index of the first isovertex associated with the modification in the isovertex array (encoded as a four-byte unsigned int);

128

- approximation error associated with the isodiamond to accelerate selective refinement queries (quantized to fourteen bits in the range $[0, 1]$).

Assuming that the vertices of the base mesh of the hierarchy of diamonds Δ are located at offset zero in the vertex array, the cost of encoding the base mesh is just that of encoding its sign field. Since the root diamond is a 0-diamond with eight vertices, its sign field requires a single byte for isosurfaces or two bytes for interval volumes. Since each isovertex is encoded as an interpolation coefficient using eight bits, storing the $|\mathbf{v}|$ isovertices requires $|\mathbf{v}|$ bytes. Finally, since the dependency relation is implicitly derived, the cost of the data structure for encoding a three dimensional RI hierarchy is

$$\underbrace{2}_{\text{base mesh}} + \underbrace{12 * (|m_a| + |m_r|)}_{\text{modifications}} + \underbrace{|\mathbf{v}|}_{\text{vertices}} \text{ bytes,}$$

where $|m_a|$ is the number of active isodiamonds and $|m_r|$ is the number of relevant isodiamonds.

9.3.3 Generating an RI hierarchy

Given a hierarchy of diamonds Δ and an isovalue κ for isosurfaces (or an isorange K for interval volumes), an RI hierarchy Δ_R is generated from Δ by performing a top-down traversal of the dependency graph describing Δ. At each node, the algorithm checks whether the min/max field values associated with the corresponding modification contain the isovalue (or the isorange). Modifications that do not contain the isovalue (or isorange) are irrelevant and neither they nor their descendants are stored in the RI hierarchy. All modifications that contain the isovalue (or isorange) will be copied in Δ_R as pointed out before, but only some of them (i.e. the active ones) intersect the isosurface (or interval volume). A closure operation must then be applied to ensure that all relevant ancestors are retained.

For all diamonds δ which intersect the isosurface or interval volume, the field values associated with the vertices of δ are used to compute the interpolation coefficients of each new isovertex along the active subdivision edges. Specifically, for each active subdivision edge $\mathbf{e} := \{\mathbf{v}, \mathbf{v}_c\}$ of δ, where $\mathbf{v}_c$ is the central vertex of δ, the interpolation coefficient γ is computed as $\gamma = (\kappa - F(\mathbf{v}_c))/(F(\mathbf{v}) - F(\mathbf{v}_c))$. It is then quantized to eight bits as the integer value $\lfloor \gamma * 2^8 + 0.5 \rfloor$.

9.3.4 Querying an RI hierarchy

Simplified isosurfaces or interval volumes can be extracted from the relevant isodiamond hierarchy through selective refinement. The selective refinement query is defined through a selection criterion based on the approximation error, which can vary in different parts of the isosurface (or interval volume).

The selective refinement algorithm operates as a top-down traversal of the dependency graph describing the RI hierarchy Δ_R. It is initialized with the modification corresponding to the root of Δ_R and at each step extracts a closed set of modifications defining a cut in the dependency graph of Δ_R, called the *active front*.

Since each modification in an RI hierarchy Δ_R is a modification in the corresponding hierarchy of diamonds Δ there are two meshes associated with a cut C defining an active front. The *current diamond mesh* Σ_δ is formed by tetrahedra of Δ belonging to the base mesh or generated by the modifications in C or in their ancestors. The *current extracted mesh* Σ is formed by triangles of the isosurface (or tetrahedra of the interval volume) intersecting the tetrahedra in the current diamond mesh. The latter contains the triangles (or tetrahedra) intersecting the base mesh or generated by the active or creation modifications in C or in their ancestors. In our implementation, we keep track of the active front and encode Σ's patches by storing the bit patterns and isovertices of the unsubdivided isodiamonds corresponding to diamonds in Σ_δ.

The selective refinement process extracts the current extracted mesh satisfying the specified selection criterion by moving the active front down from the root of the dependency graph. Modifications are performed if an isodiamond does not satisfy the selection criterion, or they are forced in order to satisfy the transitive closure of the dependency relation (thus, they may be applied even to isodiamonds which satisfy the error criterion). In either case, a modification cannot be applied until all of its ancestor modifications have been applied. Since relevant isodiamonds are also stored in an RI hierarchy, all ancestors of a given isodiamond are guaranteed to be available.

All meshes extracted from an RI hierarchy are conforming since the patch within each diamond is conforming, and a careful generation rule for the intersection cases ensures that adjacent patches are conforming (e.g. edges of neighboring patches are aligned). This is not an issue for isosurfaces, but for interval volumes this is guaranteed through the use of a unique lexicographic ordering on the vertices [NS97].

Figure 9.6a illustrates the result of a selective refinement query on the RI hierarchy defined by isovalue $\kappa = 58$ on the 2D Bonsai tree dataset (see Figure 9.4). The *current diamond mesh* (gray, blue and green triangles) covers the entire domain, and the *current extracted mesh* (blue line segments) approximates the isosurface at full resolution.

9.4   Minimal isodiamonds

In this section, we introduce the Minimal Isodiamond (MI) hierarchy for an isosurface S or an interval volume I. A *Minimal Isodiamond (MI)* hierarchy, that we denote as Δ_M, is defined by the subset of modifications of a relevant isodiamond hierarchy Δ_R that increase the number of simplices in the extracted isosurface, or interval volume. Thus, only creation and active isodiamonds are included. The remaining relevant isodiamonds are excluded from Δ_M, and their absence must be accounted for in the model as well as during selective refinement.

9.4.1   Definition

The base mesh of Δ_M is formed by the (unsubdivided) creation isodiamonds as well as the unsubdivided diamond associated with the root of Δ_R, if the latter corresponds to an active isodiamond. The modifications in Δ_M correspond to the subdivision of active and creation isodiamonds. Finally, Δ_M's dependency relation is a subset of Δ_R's dependency relation restricted to the active and creation isodiamonds. Clearly, it can also be viewed

as the restriction of the dependency relation of Δ to the modifications in Δ_M. Thus, the dependency graph of Δ_M is a (possibly disconnected) subgraph of the dependency graph describing Δ_R, whose roots are the creation isodiamonds as well as the root of Δ_R, if it is an active isodiamond. Figure 9.1d shows the dependency graph for the MI hierarchy Δ_M associated with the multiresolution isosurface representation of Figure 9.1a. The single creation isodiamond on the third row is a root of Δ_M's dependency graph. Additionally, since the root of Δ_R is an active isodiamond, it is also a root of Δ_M's dependency graph. Compared to the dependency graph of Δ_R in Figure 9.1c, we observe that all arcs leading from active or creation isodiamonds to active isodiamonds are retained, while those containing a relevant isodiamond or leading to a creation isodiamond are not retained by Δ_M.

We can get a better understanding of the structure of the dependency graph by analyzing the arcs between its modifications. Recall that the *parent-child duets* have a one-to-one correspondence with the arcs of the dependency graph of Δ (see Section 4.4.3). We consider a parent-child duet to be active if at least one of its tetrahedra are active, i.e. intersected by the isosurface or interval volume. We now show that the arcs of the dependency graph of Δ_M correspond to the active parent-child duets of the dependency graph of Δ_R.

Theorem 9.4.1. *Let d_{pc} be an active parent-child duet between a parent diamond δ_p and its child diamond δ_c. Then, δ_p is either an active isodiamond or a creation isodiamond.*

Proof. Since d_{pc} is active, it has at least one vertex $\mathbf{v}$ with a different sign value than some other vertices in d_{pc}. If $\mathbf{v}$ is not the central vertex of δ_p, then through the duet correspondence between vertices of the parent and child diamond, δ_p has two vertices with different bit values and is therefore an active isodiamond. Otherwise, assume that $\mathbf{v}$ is the central vertex of δ_p and that δ_p is not an active isodiamond. Then, since $\mathbf{v}$ has a distinct bit value from the other vertices of δ_p, it is a creation isodiamond. $\square$

Since non-active duets do not carry information related to the isosurface S or interval volume I, the arcs of the dependency graph of Δ_M retained from the ones of Δ_R are exactly those between parents and children corresponding to active parent-child duets. However, recall that the dependency graph is not explicitly represented in the model.

9.4.2 Properties

We report here some properties of creation isodiamonds δ_c and active isodiamonds δ_a in the MI hierarchy and in the corresponding RI hierarchy. These properties are important for extracting conforming representations from an MI hierarchy.

The parents of a creation isodiamond in Δ_R can be either active or relevant isodiamonds, as depicted in Figure 9.5, where the parents of a creation isodiamond (red) are an active isodiamond (green) and a relevant isodiamond (blue), respectively. Since creation isodiamonds are roots of Δ_M, none of these arcs are retained in the dependency graph of Δ_M.

The children of a creation isodiamond in Δ_R are all active isodiamonds. Since the duet d_{ca} between a creation isodiamond δ_c and a child δ_a is active, the corresponding arcs are all retained in the dependency graph of Δ_M.
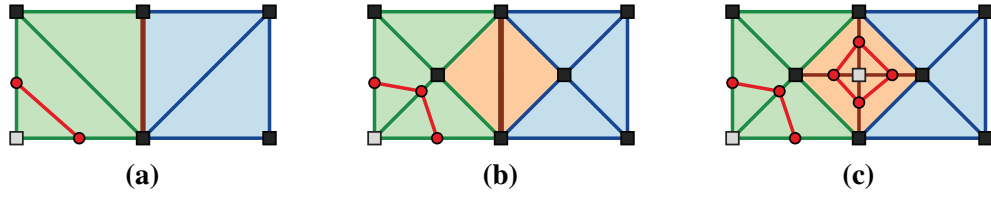
Figure 9.5: (a) A parent of a creation isodiamond in the hierarchy of diamonds (red diamond in (b)) can be either an active isodiamond (green) or a relevant isodiamond (blue). (b) Since a creation isodiamond (red) is a local root in the dependency graph of Δ_M, its ancestors in the hierarchy of diamonds are not retained. (c) All children of the creation isodiamond (red triangles) are active.

An active isodiamond δ_a, by definition, has at least one active duet such that δ_a is the parent (unless δ_a is a leaf in the dependency graph), and at least one active duet such that δ_a is the child (unless δ_a is the root of Δ_R). The parents of an active isodiamond δ_a in Δ_R can be either active, creation or relevant isodiamonds:

- All creation isodiamond parents corresponds to active duets and the corresponding arcs are retained in the dependency graph of Δ_M.

- An active isodiamond parent is retained in the dependency graph of Δ_M only if the corresponding duet is active.

- The relevant isodiamond parents in Δ_R do not correspond to active duets and are not retained in Δ_M.

Similarly, the children of an active isodiamond δ_a in Δ_R can be either active, creation, relevant or inactive isodiamonds. The only arcs retained in the dependency graph of Δ_M are those corresponding to an active child diamond δ and to an active duet between δ_a and δ.

Recall that, by definition, the spine of an active diamond δ_a is part of all duets containing δ_a as child. If the sign values of the spine differ, all duets containing δ_a as child are active, and their corresponding arcs are retained in Δ_M. This corresponds to a diamond whose spine intersects the isosurface or one (or both) of the bounding surfaces of the interval volume. In addition, when representing interval volumes, all duets are active if the signs of the spine vertices are both zero, corresponding to a diamond whose spine is located between the two bounding surfaces. Otherwise, some of the duets could be inactive. Only the arcs between δ_a and a parent δ corresponding to active duets are retained by Δ_M. Table 9.1 summarizes the status of Δ_M's dependency graph arcs based on the isodiamond types of the parent δ_p and the child δ_c of a duet d_{pc}.

9.4.3 Data structure

The MI hierarchy is represented in the same way as the RI hierarchy (see Section 9.3.2), where each modification is encoded using 12 bytes, and each vertex requires a single byte. Consequently, a model with $|m_c|$ creation isodiamonds, $|m_a|$ active isodiamonds and $|\mathbf{v}|$

Table 9.1: Status of arc in dependency graph of Δ_M based on isodiamond types of parent δ_p (rows) and child δ_c (columns) of a parent-child duet d_{pc}. Types $\{R, C, A, I\}$ denote, respectively, Relevant, Creation, Active, Inactive.

$\delta_p \setminus \delta_c$	R	C	A	I
R	x	x	x	x
C	–	–	✓	–
A	x	x	*	x
I	–	–	–	x

✓   Always retained
*   Retained if active duet
x   Never retained
–   Not possible

isovertices requires

$$\underbrace{2}_{\text{base mesh}} + \underbrace{12 * (|m_a| + |m_c|)}_{\text{modifications}} + \underbrace{|\mathbf{v}|}_{\text{vertices}} \text{ bytes.}$$

Since the creation isodiamonds are required to initialize the base mesh, for efficiency, we store the creation isodiamonds separately from the active isodiamonds.

Note that, since active and creation isodiamonds are the only modifications that contain patches of the output isosurface or interval volume, the MI hierarchy represents exactly the same set of isovertices (and thus patches) as the RI hierarchy, despite encoding significantly fewer modifications.

9.4.4 Generating an MI hierarchy

The Minimal Isodiamond hierarchy is generated from a hierarchy of diamonds in a similar manner as the RI hierarchy (as detailed in Section 9.3.3). The only difference is that when adding isodiamonds to the model, we must additionally test for the type of isodiamond. These tests are efficiently carried out by applying bitmasks to the bit pattern of each diamond. If the bit pattern of a diamond δ is not homogeneous, then δ is an active isodiamond. If only the sign of the central vertex differs from those of the other vertices, then δ is a creation isodiamond. Otherwise, the diamond is discarded. In all cases, if the min/max range of a diamond contains the isovalue or isorange, we recursively test all of its descendants.

9.4.5 Querying an MI hierarchy

Uniform– and variable-resolution isosurfaces or interval volumes can be extracted from an MI hierarchy through a similar selective refinement process as the RI hierarchy (as discussed in Section 9.3.4).

There are, however, two key differences due to the fact that the relevant isodiamonds are not encoded and due to the changes to the dependency relation in the MI hierarchy.

The first difference concerns the bit pattern of a diamond. Recall that selective refinement requires a closure operation, that is, all parent modifications of a diamond δ must be applied before δ can be applied. Moreover, our isodiamond encoding scheme (see
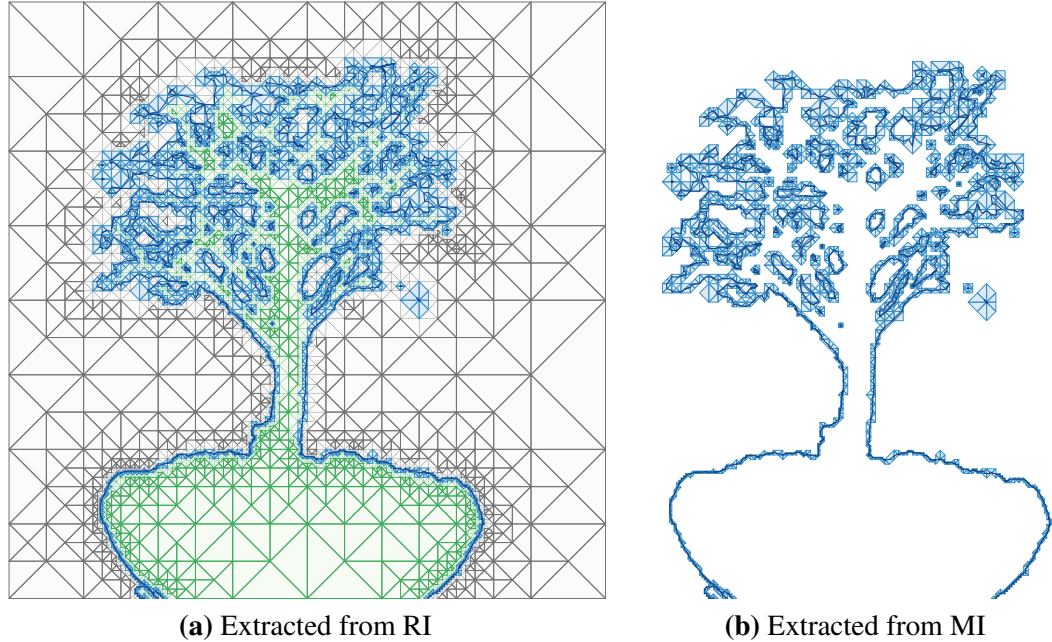
(a) Extracted from RI   **(b)** Extracted from MI

Figure 9.6: Diamond mesh Σ_δ (gray, green and blue triangles) and current mesh Σ (dark blue line segments) of the 2D bonsai tree dataset ($\kappa = 58$) extracted from (a) an RI hierarchy Δ_R and (b) an MI hierarchy Δ_M. Observe that the mesh extracted from Δ_R covers the entire domain while the mesh extracted from Δ_M only covers the isosurface patches.

Section 9.2) assumes that the entire bit pattern of a diamond can be recovered from its ancestors. But, in an MI hierarchy Δ_M, the parents of an active isodiamond δ are a subset of the parents of its corresponding isodiamond in the RI hierarchy Δ_R. As a consequence, the sign values of the vertices from an *RI* parent δ_r, that is not an *MI* parent of δ, will not be available to complete the bit pattern associated with δ. This situation, however, only occurs in those active isodiamonds in which the two spine vertices have the same sign value. Since the duet corresponding to the RI parent δ_r is not active, all of its vertices, including the two spine vertices of δ, must have the same sign value. Thus, the missing vertices can be safely assigned the same sign value as the spine vertices.

The other issue is related to the active front when extracting a mesh from an MI hierarchy Δ_M. As in selective refinement from an RI hierarchy (see Section 9.3.4), we consider an active front C on the dependency graph of Δ_M and two meshes: the *current diamond mesh* Σ_δ, i.e., the tetrahedral mesh associated with front C and extracted from diamonds in Δ, and the *current extracted mesh* Σ, formed by the triangles (or tetrahedra) of the currently extracted isosurface (or interval volume) and defined by the patches intersected by the diamonds in Σ_δ. Since the relevant isodiamonds are not encoded in Δ_M, the domain Ω of the dataset is no longer covered by the tetrahedra in Σ_δ (see Figure 9.6b, where Σ_δ consists of blue triangles and Σ consists of dark blue line segments). Further, since Δ_M is not necessarily described by a connected dependency graph, C does not need to be connected. Consequently, the current diamond mesh can be disconnected and is no longer guaranteed to be conforming (see the interface between the red and purple triangles

134
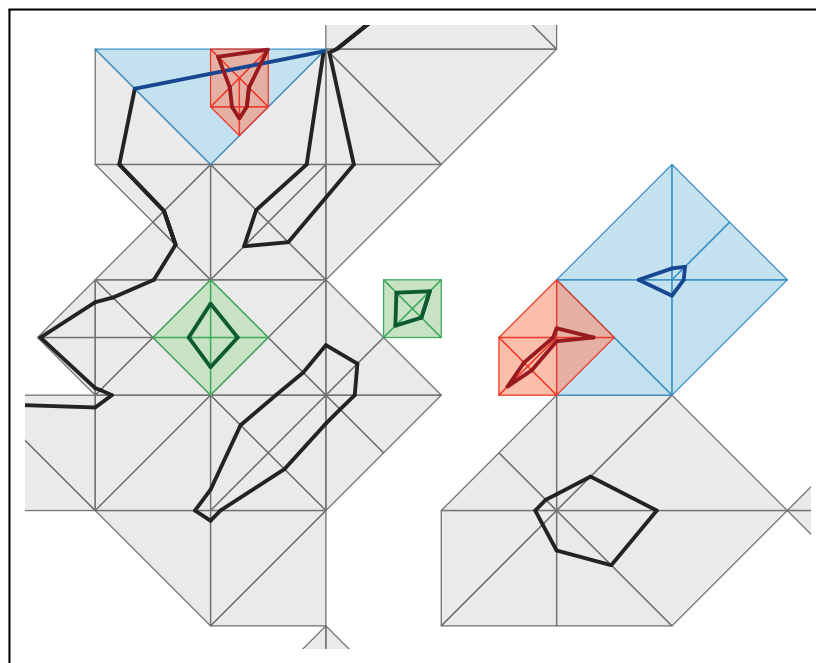
(a) Before closure operator
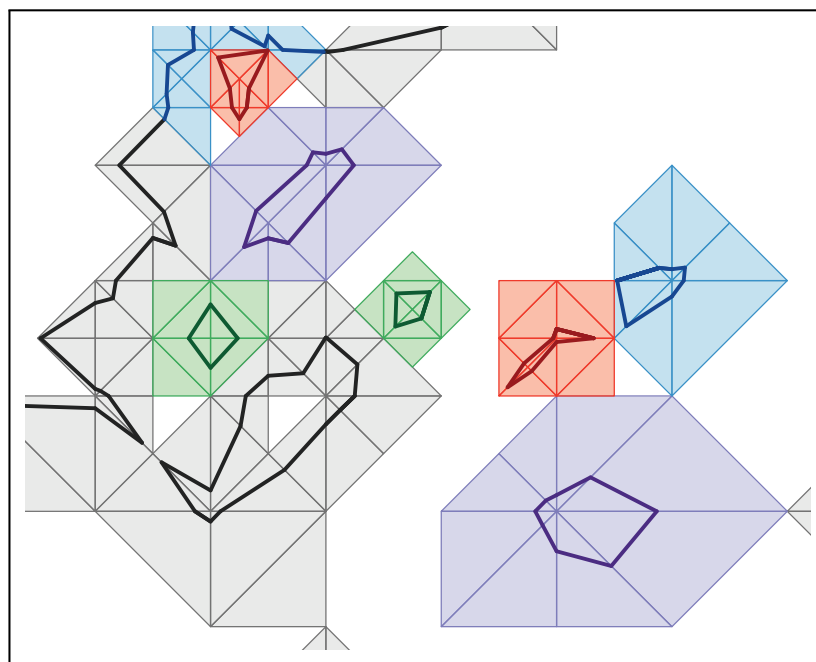
(b) After closure operator

Figure 9.7: Subdivision of a creation isodiamond δ_c can cause the new patches to intersect other patches of the current mesh as in the overlap between red and blue triangles in (a). However, applying the closure operator to all children of δ_c forces the active front to include δ_c (red and green triangles in b). The extracted isosurface in (b) is conforming even though its diamond-based LEB mesh is not (interface between the red and purple triangles in (b)).

in Figure 9.7b).

Nevertheless, since isosurface or interval volume patches are only embedded within active isodiamonds, the current extracted mesh Σ is guaranteed to be conforming as long as

(a) the interiors of active tetrahedra do not overlap and

(b) face-adjacent tetrahedra are conforming along their shared face if that face intersects the isosurface or interval volume.

Property (b) is ensured by the patch triangulation cases for each isodiamond as well as by the closure operation in selective refinement (as long as property (a) is not violated).

Since (unsubdivided) creation isodiamonds do not contain active tetrahedra, property (a) is not violated if their tetrahedra overlap other tetrahedra in Σ_δ. However, creation isodiamonds require a modified subdivision rule to ensure that the above two properties remain satisfied after their subdivision. Applying the modification associated with a creation isodiamond δ adds a new disconnected node to the active front C. Since δ is a root in the dependency graph of Δ_M, it has no ancestors and thus, applying the closure operator to δ does not connect it to its ancestors in the DAG describing the hierarchy of diamonds Δ. As a consequence, the diamonds in Σ_δ overlapping the domain of δ will not subdivide. If any of these diamonds are active (in violation of property (a)), their patches can overlap (see the red and blue regions of Figure 9.7a).

We solve this by applying the closure operator to each child of δ, i.e. applying all modifications on which the children depend. The closure operator cascades up on the dependency graph until (a) the existing front is reached (b) an arc corresponding to a non-active duet is reached or (c) another creation node (i.e. a local root of the DAG) is reached. This joins the fronts of δ and the diamonds overlapping its domain (see Figure 9.7b), and thus Σ satisfies both the above properties, guaranteeing that the extracted meshes (isosurfaces or interval volumes) are conforming.

9.5   Results

We demonstrate the compactness and efficiency of the two multiresolution isodiamond models introduced here on We have run experiments on a testbed of over 20 medical, scientific and synthetic regular volume data sets (in 3D) of dimensions up to 512^3 and summarize the aggregate information and trends. All experiments were performed on a 2 GHz Pentium Core 2 Duo laptop with 4 GB RAM.

Recall that our isodiamond encoding requires twelve bytes per isodiamond and a single byte for each isovertex. Also, both the RI and the MI hierarchy encode all active isodiamonds, but the RI hierarchy encodes in addition all relevant isodiamonds, while the MI hierarchy stores only those relevant isodiamonds that are creation isodiamonds. The number of creation isodiamonds is typically less than 0.5% of the number of relevant iso-diamonds (and often significantly less, as shown in columns 5 and 6 in Table 9.2). Further, since isovertices are only encoded in creation or in active isodiamonds, and both models encode these diamonds, the MI hierarchy encodes the same number of isovertices as the RI hierarchy. Our experiments on more than 20 datasets indicate that, when encoding

136

Table 9.2: Comparison between the size and number of elements in the isodiamond hierarchies (isovertices v_i; relevant m_r, creation m_c and active m_a isodiamonds) and those of the mesh at full resolution (vertices v and triangles or tetrahedra t). Data structures sizes are listed in megabytes (1 MB = 1024^2 B).

	Dataset	Isodiamond Hierarchies				Indexed		Data Structure Sizes (MB)			Comparison
		$|v_i|$	$|m_r|$	$|m_c|$	$|m_a|$	$|v|$	$|t|$	MI	RI	Indexed	RI / MI
Isosurface	Bucky.32$_{\{128\}}$	20.2 k	4.74 k	84	7.96 k	16.4 k	32.6 k	0.11	0.16	0.56	1.48 x
	Fuel.64$_{\{7.2\}}$	23.5 k	5.72 k	5	9.93 k	18.2 k	36.3 k	0.14	0.20	0.62	1.48 x
	Neghip.64$_{\{59.1\}}$	56.3 k	14.9 k	17	24.5 k	43.1 k	86.1 k	0.33	0.51	1.48	1.51 x
	Armadillo.12$_{\{0\}}$	62.3 k	19.4 k	5	27.5 k	47.0 k	93.9 k	0.37	0.60	1.61	1.59 x
	Hydrogen.128$_{\{24\}}$	75.8 k	29.5 k	1	33.6 k	56.9 k	114 k	0.46	0.79	1.95	1.74 x
	Tooth.256$_{\{650\}}$	11.5 k	73.7 k	21	113 k	195 k	390 k	1.31	2.15	6.70	1.65 x
	Armadillo.256$_{\{0\}}$	267 k	80.5 k	7	118 k	201 k	401 k	1.60	2.52	6.89	1.57 x
	Aneurism.256$_{\{128\}}$	271 k	139 k	1.25 k	116 k	216 k	430 k	1.60	3.17	7.39	1.99 x
	Bucky128$_{\{128\}}$	368 k	97.8 k	97	161 k	279 k	600 k	2.20	3.32	10.1	1.51 x
	Bunny 201$_{\{0\}}$	671 k	190 k	10	296 k	505 k	1.01 m	4.03	6.21	17.3	1.54 x
	Heptoroid.256$_{\{0\}}$	844 k	194 k	215	364 k	642 k	1.29 m	4.97	7.20	22.1	1.45 x
	Head.256$_{\{58\}}$	1.17 m	258 k	623	490 k	884 k	1.77 m	6.73	9.68	30.4	1.44 x
	Engine.256$_{\{100\}}$	1.39 m	336 k	219	601 k	1.06 m	2.11 m	8.21	12.1	36.2	1.47 x
	Bonsai.256$_{\{35\}}$	2.06 m	491 k	1.89 k	860 k	1.60 m	3.19 m	11.8	17.4	54.9	1.47 x
	Aneurism.512$_{\{650\}}$	2.26 m	758 k	3.75 k	971 k	1.74 m	3.48 m	13.3	21.9	59.7	1.65 x
	Armadillo.512$_{\{0\}}$	1.16 m	343 k	8	513 k	870 k	1.74 m	6.97	10.9	29.9	1.56 x
	XMasTree.512$_{\{868\}}$	2.55 m	760 k	15.6 k	984 k	2.09 m	4.17 m	13.9	22.4	71.7	1.61 x
Interval Volume	Bucky32$_{\{(118,138)\}}$	40.5 k	4.55 k	65	8.75 k	32.8 k	99.0 k	0.14	0.19	1.88	1.37 x
	Fuel.64$_{\{(7.2,11.4)\}}$	45.8 k	5.45 k	2	10.8 k	35.3 k	109 k	0.17	0.23	2.07	1.37 x
	Neghip.64$_{\{(59.1,124.1)\}}$	90.0 k	12.9 k	14	31.7 k	69.0 k	262 k	0.45	0.60	4.79	1.33 x
	Armadillo.128$_{\{(-10,10)\}}$	179 k	32.4 k	58	101 k	135 k	705 k	1.32	1.69	12.3	1.28 x
	Hydrogen.128$_{\{(24,48)\}}$	101 k	18.3 k	1	56.9 k	75.9 k	400 k	0.75	0.96	6.97	1.28 x
	Tooth.256$_{\{(440,1290)\}}$	350 k	68.2 k	1.27 k	182 k	277 k	1.32 m	4.41	5.41	23.3	1.31 x
	Armadillo.256$_{\{(-10,10)\}}$	585 k	95.0 k	39	335 k	439 k	2.35 m	2.43	3.20	41.0	1.25 x
	Aneurism.256$_{\{(118,128)\}}$	557 k	144 k	1.34 k	123 k	444 k	1.33 m	4.39	5.48	25.3	1.84 x
	Bucky128$_{\{(128,188)\}}$	595 k	70.2 k	75	246 k	456 k	2.01 m	1.95	3.58	35.8	1.24 x
	Bunny 201$_{\{(0,30)\}}$	850 k	127 k	10	488 k	639 k	3.45 m	3.38	4.19	59.9	1.23 x
	Heptoroid.256$_{\{(-10,10)\}}$	1.40 m	209 k	1.06 k	788 k	1.06 m	5.59 m	6.40	7.85	97.5	1.23 x
	Head.256$_{\{(42,72)\}}$	2.48 m	199 k	429	1.20 m	1.89 m	9.28 m	10.4	12.7	163	1.14 x
	Engine.256$_{\{(55,175)\}}$	1.70 m	196 k	130	809 k	1.29 m	6.17 m	16.1	18.3	109	1.21 x
	Bonsai.256$_{\{(45.5,86.5)\}}$	2.58 m	438 k	3.52 k	864 k	2.03 m	7.08 m	10.9	13.1	131	1.40 x

isosurfaces, the MI hierarchy requires about 65% of the space of a corresponding RI hierarchy (with a standard deviation of less than 5%). For interval volumes, an MI hierarchy requires approximately 78% the space of the corresponding RI hierarchy (with a standard deviation of 8%).

Although the principle goal of multiresolution hierarchies is not compression, we demonstrate the compactness of the two isodiamond hierarchies by comparing them to a simple *indexed* representation of the mesh describing the isosurface S or the interval volume I at full resolution (see Table 9.2).* This representation encodes vertices as three floating-point coordinates, and triangles, or tetrahedra, through the indices of their three, or four vertices, respectively. Thus, encoding an isosurface S with $|v|$ vertices and $|t|$ triangles requires $(12\,|v| + 12\,|t|)$ bytes, and encoding an interval volume I with $|v|$ vertices and $|T|$ tetrahedra requires $(12\,|v| + 16\,|T|)$ bytes.

---

*This is a simplified version of the extended IA data structure from Section 8.6 that only supports the boundary operation for top simplices (triangles and tetrahedra, in 2D and 3D, respectively) and is efficient for rendering applications.

Compared to an indexed representation of the isosurface at full resolution, the RI hierarchy is approximately three times more compact and the MI hierarchy is about five times more compact. For interval volumes, the RI hierarchy is approximately eight times more compact (as reported in [WD08b]), while the MI hierarchy is more than ten times more compact than the mesh at full resolution. In addition to the space savings, a mesh extracted from both isodiamond hierarchies also implicitly encodes the adjacencies between its triangles, or tetrahedra. This is typically required in downstream mesh processing applications and would require an extra 12 bytes per triangle, or 16 bytes per tetrahedron, since we would need to use for the output mesh an indexed representation enhanced with adjacency information among triangles or tetrahedra.

9.5.1 Front-size and extraction times

(a) Front Diamonds - Isosurface

(b) Extraction Times - Isosurface

(c) Front Diamonds - Interval Volume

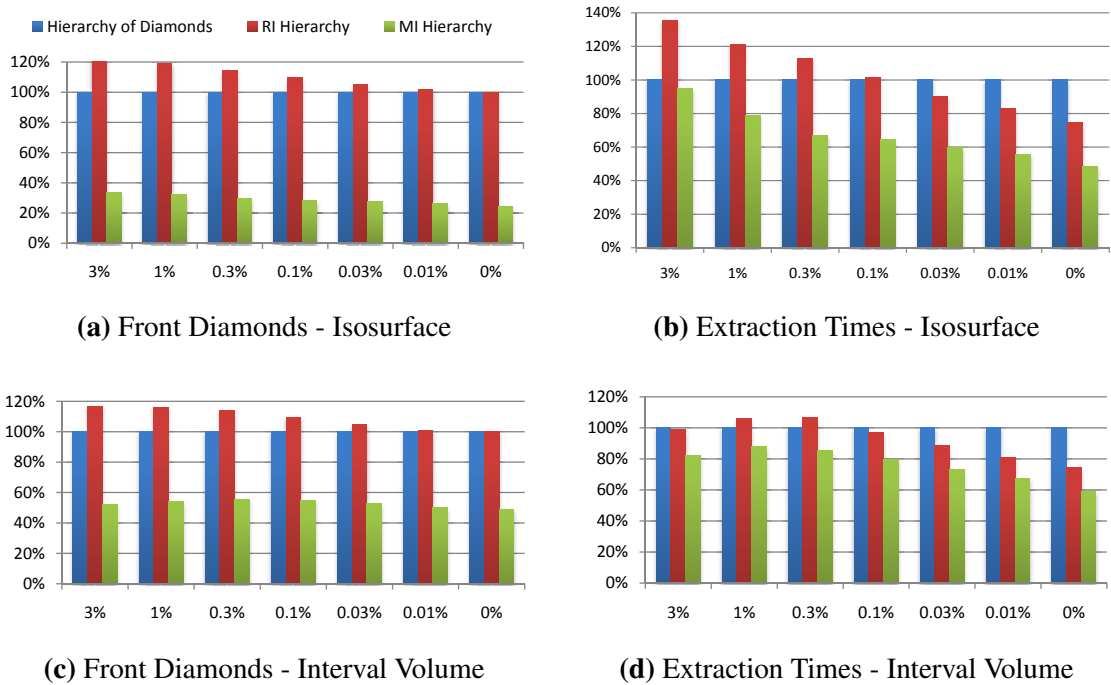**(d)** Extraction Times - Interval Volume

Figure 9.8: Comparison of relative number of diamonds in the active front (a,c) and extraction times (b,d) for isosurfaces and interval volumes extracted from the hierarchy of diamonds (blue), RI hierarchy (red) and MI hierarchy (green). The extraction has been performed at uniform resolution with a maximum error of $\epsilon \in \{3\%, 1\%, .3\%, .1\%, .03\%, .01\%, 0\%\}$ (horizontal axis). All values are relative to the hierarchy of diamonds (vertical axis). Values are averages across over 20 datasets of varying sizes and complexity.

One of the key motivations for the isodiamond hierarchies is performing selective refinement queries efficiently on a multiresolution representation of a specific predetermined isosurface or interval volume. These models enable a quicker extraction of uniform or variable-resolution output meshes with respect to extracting them from a hierarchy of diamonds. Moreover, the active front resulting from the MI hierarchy (i.e. the current

diamond mesh Σ_δ) requires only a fraction of the diamonds to reconstruct an equivalent output mesh as either the original hierarchy of diamonds or the corresponding RI hierarchy. Thus, queries on an MI hierarchy require less memory at runtime and the resultant output meshes can be post-processed more efficiently.

Isosurfaces extracted from an MI hierarchy have associated active fronts containing approximately 25% of the diamonds as in the active fronts extracted from a hierarchy of diamonds or from an RI hierarchy (see Figure 9.8a). Similarly, interval volumes extracted from an MI hierarchy have associated active fronts containing approximately 50% of the diamonds as the other two models (see Figure 9.8c). Note that in these queries, the selection criterion for the original hierarchy of diamonds Δ depends on the range of field values as well as the approximation error, while the selection criterion for the RI hierarchy Δ_R and the MI hierarchy Δ_M only depends on the approximation error. Since many isodiamonds near the root of the dependency graph of Δ_R are relevant, but their corresponding diamonds in Δ might not intersect the isovalue (or isorange), their modifications will be applied in Δ_R but not in Δ. Thus, when allowing a higher error tolerance, Δ_R can have a time and space overhead of around 10-20% compared to Δ. However, this effect is reduced as the error tolerance decreases, and the active front descends to lower levels of the dependency graph (e.g. compare the relative values of Δ_R and Δ on the horizontal axes of Figure 9.8 as the error tolerance decreases).

Both isodiamond hierarchies achieve significant reduction in extraction times during selective refinement, compared to the diamond hierarchy, as the maximum allowed error decreases (see Figure 9.8 (b) and (d)). Specifically, as ϵ approaches zero, the RI hierarchy can extract an equivalent isosurface or interval volume in about 3/4 the time with respect to the hierarchy of diamonds, while the MI hierarchy can extract an equivalent mesh in about half the time with respect to the hierarchy of diamonds. Note that, when the error is high (e.g. the values toward the left side of each graph in Figure 9.8), the extracted meshes and times are relatively small, and, thus, relative differences are less significant, while for lower errors (e.g. toward the right side of each graph), the times and active front sizes increase. This suggests that the hierarchy of diamonds is ideally suited for extracting low resolution isosurfaces and interval volumes from the model during the exploratory phases of the analysis of a dataset. Isodiamond hierarchies are better suited for in-depth analysis once the desired isosurface or interval volume has been determined and higher resolution approximations are required for inspection and processing.

The left side of Figure 9.9a depicts a zero-error isosurface ($\kappa = 868$) containing approximately two million vertices and four million triangles extracted from the 512^3 Christmas Tree dataset. On the right half, the blue points depict the central vertices of active isodiamonds, and the purple squares coincide with the central vertices of creation isodiamonds. Figure 9.9c depicts an interval volume ($K = [42, 72]$) containing 879 K vertices and 3.3 million tetrahedra extracted from the 128×256^2 Visible Male Head dataset with uniform error less than 1%. The mesh has been clipped along the median plane to illustrate the sizes of tetrahedra.

The adaptability of the isodiamond models enables location dependent queries in addition to those defined by approximation errors. Figure 9.9b illustrates a variable-resolution isosurface ($\kappa = 0$) extracted from the Armadillo dataset and focuses the resolution in a region of interest around the head. The error of any triangle within the box is zero,

and the error outside the box can be arbitrarily large. Colors indicate the resolution of the diamonds framing the triangles. The blue points coincide with the central vertices of active isodiamonds. Figure 9.9d illustrates a variable-resolution interval volume extracted from the 201^3 Bunny data set ($K = [0, 20]$), again with a region of interest around the head. The tetrahedra are shrunken slightly and the model is clipped along a plane in order to illustrate the cells of the interval volume.

9.6 Discussion

We have developed two efficient mesh-based multiresolution models for individual iso-surfaces and interval volumes which are extracted from a volume data set described as a hierarchy of diamonds. Both models exploit the one-to-one correspondence between diamonds in the hierarchical representation of the field and modifications in the multiresolution representation of the extracted isosurface or interval volume.

The Relevant Isodiamond (RI) hierarchy encodes the set of active isodiamonds as well as their relevant ancestors. Extracted isosurfaces and interval volumes are guaranteed to be conforming due to the one-to-one correspondence between isodiamonds in the RI hierarchy and the diamonds of the diamond hierarchy. As demonstrated in Figure 9.8, current diamond meshes extracted from the RI hierarchy are generated by approximately the same number of diamonds as those extracted from the diamond hierarchy, in less time. The faster extraction times are likely due to the pre-calculation of interpolation coefficients for the patch isovertices during the generation of the RI hierarchy. Since the scalar values of each diamond's vertices do not need to be retrieved, the modification corresponding to an isodiamond can be applied via a single lookup into the isovertex array.

The Minimal Isodiamond (MI) hierarchy encodes all active isodiamonds and only the subset of relevant isodiamonds that are creation isodiamonds. The dependency relation R of its modifications forms a forest of DAGs. An MI hierarchy is described by a dependency graph which is a subgraph of the dependency describing Δ. Since inactive isodiamonds are not created during mesh extraction, current diamond meshes that result from selective refinement applied to MI hierarchies are significantly smaller than those arising from a selective refinement applied to the RI hierarchy.

As indicated in Table 9.2, both models can be encoded quite compactly compared to an indexed representation of the extracted isosurface or interval volume at full resolution. However, the primary benefit of these models is that they efficiently support selective refinement queries, and thus enable the extraction of meshes satisfying an application-defined error criterion.

Since isodiamond hierarchies only encode the sign field and vertex interpolation coefficients, and not the higher dimensional simplices, this relative advantage increases significantly with the dimension of the encoded mesh. Thus, as long as an appropriate set of lookup tables is defined (as in [BWC04]), the isodiamond representation should lead to significant savings for higher dimensional simplicial meshes. Furthermore, since an interval volume encodes the subvolume enclosed between two isosurfaces, and the storage requirements are related to the number of isovertices on its lower and upper surfaces, an interval volume with only one boundary surface can be used as an efficient multiresolution
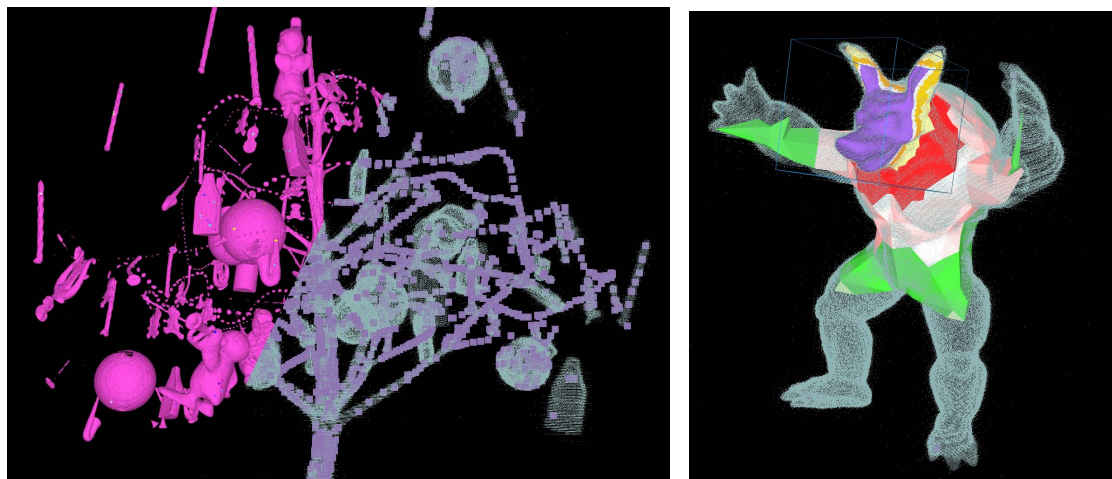
(a) Xmas Tree Isosurface

(b) Armadillo Isosurface
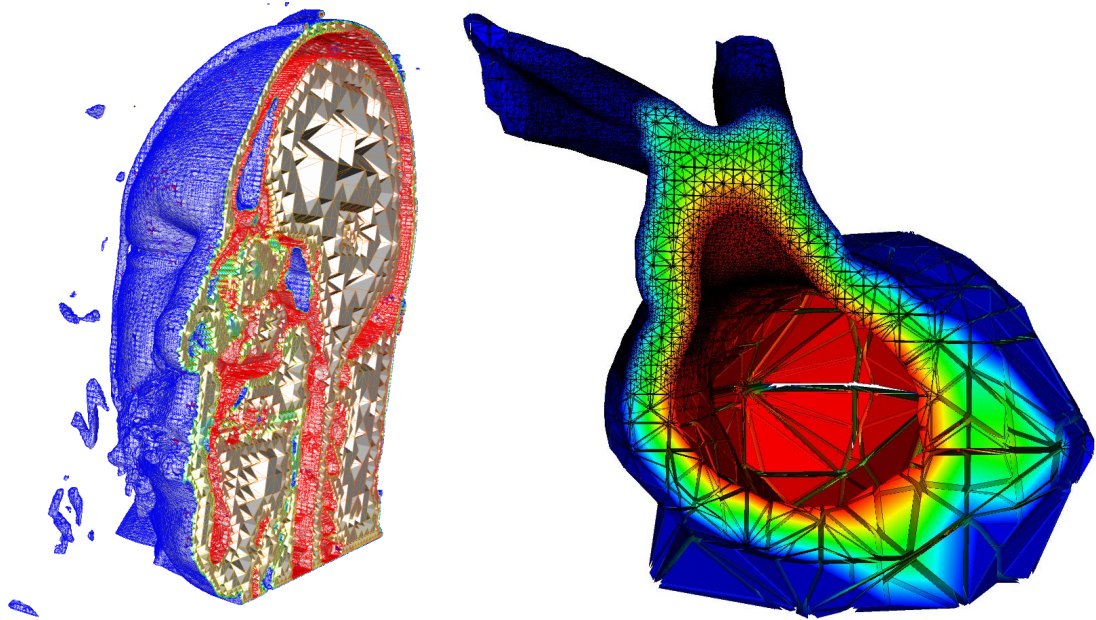
(c) Head Interval Volume

(d) Bunny Interval Volume

Figure 9.9: (a) Full resolution isosurface extracted from 512^3 Christmas tree ($\kappa = 868$). Purple squares on right half indicate creation isodiamonds while blue points indicate active isodiamonds. (b) Isosurface extracted from 512^3 Armadillo ($\kappa = 0$) using a cubic ROI. Triangle colors indicate the DAG level of the isodiamonds and the blue points indicate active isodiamonds in the model. (c) Interval volume ($K = [42, 72]$) extracted from the Head dataset. The mesh is clipped along the median plane to show the internal tetrahedra. (d) Region-based LOD on the bunny model (around the head) with isorange $K = [0, 20]$.

volumetric representation of an object.

Here, we have applied isosurface and interval volume cases to the tetrahedra within each extracted diamond. Given a consistent set of lookup tables defined on the vertex categories, our framework should work equally well for multiresolution non-manifold meshes [HSSZ97], multi-material interface reconstruction [JLSW02, BDS+03] or dual contouring on the tetrahedra or duets within each diamond [Nie08]. We intend to explore these possibilities in our future work.

The two concepts of supercubes and isodiamonds provide benefits that are mutually orthogonal to each other. Isodiamonds reduce the cost of irregular updates to a multiresolution isosurface or interval volume defined on a DMSF. However, the set of active and relevant modifications (isodiamonds) are defined on a sparse, coherent subset of the nodes of a full DMSF. Thus, supercubes seem ideal to reduce the geometric overhead of such a representation. Additionally, since subdivision edges of a diamond can be uniquely identified with the diamond whose subdivision introduces them, their associated isovertices can be clustered together to locally. Rather than having a pointer into a global isovertex array, each supercube can contain a pointer into a global isovertex array, and each isodiamond can point into a local isovertex array using a smaller offset, thereby significantly reducing the cost of isodiamond modifications as well.

Chapter 10

Hierarchies of balanced hypercubes

Although the focus of this thesis has been on nested simplicial decompositions, there are many applications of nested hypercubic grids including those based on quadtrees, octrees and their higher dimensional extensions [Sam06]. Downstream applications typically 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 [Bey00]. A common *adaptivity* constraint is to ensure that neighboring elements differ in resolution by at most one refinement level, i.e. the ratio of edge lengths between *neighboring* elements can be at most 2:1. This constraint has been considered in many different application domains, including computational geometry [BEG94, BG06], scientific visualization [EKT01, WKE99] and computer graphics [VHB87] under various terms such as *restricted* [VHB87, SS92, RMSS01], *smooth* [Moo92] and *balanced* [Moo95, SSB08]. However, different definitions of the term neighbor lead to different balancing restrictions e.g. two hypercubes can be adjacent along a common vertex, edge, . . . , facet.

In this chapter, we introduce *hierarchies of balanced hypercubes* defining families of nested hypercubic meshes with balancing restrictions. Although they do not generate conforming meshes (as discussed in Section 2.1.1), these hierarchies satisfy the *MultiTessellation (MT)* model [DPM97, DM02] as described in Section 2.4. Specifically, the modifications are defined by the regular refinement of a hypercube into 2^d hypercubes, and we define the *direct dependency relation* among such modifications to yield balanced hypercubic meshes.

We provide a formal treatment of the dependency relation among hypercubes in a nested hypercubic mesh that is necessary to generate balanced hypercubic meshes. Whereas previous attempts have placed an upper bound on the number of such dependencies [Wei81, Moo95], we identify the exact number and location of all such dependencies. This framework is general enough to encompass traditional quadtrees and octrees, which we refer to as *unbalanced*.

Our analysis stems from a novel reinterpretation of nested hypercubic meshes through the lens of diamond hierarchies, whereby hypercubes are seen as a special class of diamonds. In particular, we observe that 0-diamonds have a hypercubic domain, and thus hypercubic meshes can be seen as non-conforming diamond meshes composed entirely of complete 0-diamonds. Thus, we propose a diamond-based approach to modeling, encoding and processing balanced hypercubic meshes.

This yields a compact pointerless encoding for balanced hierarchies of hypercubes, which provides random access to the hierarchical ancestors of each hypercube, and for their extracted hypercubic meshes. The connection to diamonds also suggests a supercube-
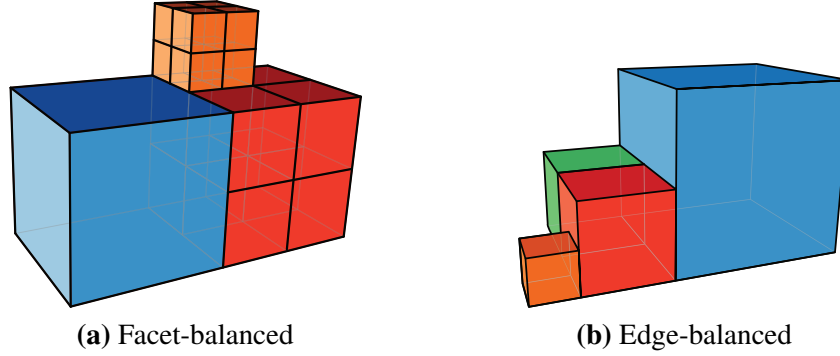
(a) Facet-balanced   **(b)** Edge-balanced

Figure 10.1: In a balanced hypercubic mesh, the neighbors of a node at refinement level ℓ (red) can be at level $\ell - 1$ (blue), ℓ (red or green) or $\ell + 1$ (orange). A facet-balanced mesh (a) is not necessarily edge-balanced, but an edge-balanced mesh (b) is always facet-balanced.

based representation for encoding the vertices and cells of a balanced 2^d-tree mesh. The coherence among neighboring cubes is exploited through the use of an implicit clustering scheme based on supercubes (Chapter 5).

Although the cells of a 2^d-tree are all cubic, the vertices of a nested hypercubic mesh cover the entire domain, and thus, the supercube-based DMSF model of Chapter 7 is applicable when associating information with the vertices of a 2^d-tree.

Finally, we introduce a dimension–independent algorithm for triangulating nested hypercubic meshes. Our algorithm is based on a local diamond-based triangulation of each hypercube (see Section 10.3). As this algorithm is based on regular simplex bisection, the triangulation has guaranteed geometric and adaptivity constraints. Compatibility between adjacent hypercube triangulations is implicitly enforced and the complexity of the mesh is bounded by a multiplicative constant (assuming a fixed dimension).

10.1 Hypercube hierarchies

Recall from Section 3.1.2 that applying regular refinement to a hypercubic domain generates *nested hypercubic meshes*. The containment relation of hypercubes can be modeled as a 2^d-tree, in which the 2^d *sibling* hypercubes generated by refining a *parent* node are the *children* of that node. The *leaves* of this tree (the hypercubes without children) define the cells of the hypercubic mesh.

10.1.1 Balanced refinement

As was the case for simplex hierarchies, the unrestricted refinement of a hypercube can lead to meshes with undesirable qualities. A common compromise is to restrict the degree of refinement in such a way that adjacent hypercubes are at most one refinement apart. The characteristics of such meshes depend on the type of adjacency relationships that we wish to restrict.

Recall from Section 2.1 that two d-cubes h_1 and h_2 in a nested hypercubic mesh are *k-adjacent*, $0 \le k < d$, if their intersection ($h_1 \cap h_2$) defines a k-cube on the boundary of h_1 or h_2. They are *k-neighbors* if they are k-adjacent, but not $(k + 1)$-adjacent. A

nested hypercubic mesh C is said to be *j-balanced* if all k-neighbors, $j \leq k \leq d$ differ in refinement level by at most one [Moo95]. Observe that for $0 \leq j < k \leq d$, any *j*-balanced hypercubic mesh is also k-balanced [Moo95] (see Figure 10.1). Since the only d-neighbor of a d-cube is itself, all nested hypercubic meshes are d-balanced. Given a k-balanced hypercubic mesh, its *j*-balanced counterpart, $j \leq k$, is uniquely defined, and can be generated by using a simple greedy algorithm [Moo95].

For simplicity, we will refer to 0-balanced as *vertex-balanced*; to 1-balanced as *edge-balanced*; to $(d-1)$-balanced as *facet-balanced*; and, finally, to d-balanced as *unbalanced*.

10.1.2   Balanced hypercube hierarchies

Consider the collection $\boxminus$ of all hypercubes at level $\ell \leq N$, generated by applying regular refinement to a hypercubic domain Ω. The k-balancing restriction induces a *direct dependency relation* among the cells of $\boxminus$, where the refinement of a d-cube h_c at level ℓ *depends* on that of a d-cube h_p at level $(\ell - 1)$ if h_c and h_p are k-neighbors in some hypercubic mesh. Recall that for 2^d-trees, the hypercube h_p whose refinement generates a hypercube h_c is referred to as its *parent*, which we denote as *Parent*(h_c), while the 2^d cubes generated during the refinement of h_p are referred to as its *children*, which we denote as *Children*(h_p) and refer to as *siblings* in the 2^d-tree. The direct dependency relation in a hierarchy of k-balanced hypercubes, for a node h at level ℓ is defined by a set of *immediate predecessors* at level $\ell - 1$ in $\boxminus$, which include *Parent*(h), on which h depends for k-balanced refinement, as well as a set of *immediate successors* at level $\ell + 1$ in $\boxminus$, which include *Children*(h), whose k-balanced refinement depend on h. The collection $\boxminus$ of hypercubes, along with the dependency relation induced by the k-balancing restriction defines a multiresolution model that we refer to as a *hierarchy of k-balanced hypercubes*.

We now use properties of regular hypercubic refinement to analyze the various types of neighbor-balancing dependencies in a hierarchy of balanced hypercubes. For example, since all 2^d *sibling* cubes are generated at the same time, neighbors along these faces can never be at a lower resolution (see Figures 2.4).

Let us consider *Neighbors*(h_p), the 3^d neighbors of a cube h_p at the same level of resolution as well as *Children*(h_p), the 2^d cubes generated by the refinement of h_p (see Figure 10.2, where h_p is the red square, *Neighbors*(h_p) contains h_p and its adjacent blue and green squares, and *Children*(h_p) are the four squares below h_p at the next level of resolution). Observe that each cube in *Neighbors*(h_p) is offset from h_p along one of the axis-aligned directions in $\mathbb{R}^d$. That is, if $\vec{f}$ is the offset from the midpoint of h_p to the midpoint of a cube in *Neighbors*(h_p), then $\vec{f} = s_p \cdot \vec{g}$, where $g_i \in \{-1, 0, 1\}$, and s_p is the side length of h_p. Of the 3^d elements in *Neighbors*(h_p), $\binom{d}{k} \cdot 2^{d-k}$ are k-neighbors of h_p. Furthermore, the midpoints of the edges connecting each neighbor to h_p coincide with the vertices of *Children*(h_p) (see Figure 10.2). Thus, there is a one-to-one correspondence between cubes in *Neighbors*(h_p) and vertices in *Children*(h_p).

Although the cubes in *Neighbors*(h_p) define the set of all possible balancing neighbors of a cube $h \in$ *Children*(h_p), it is not necessary to refine all of these cubes to maintain a k-balanced hypercubic mesh upon the refinement of h. In fact, since h only has 2^d vertices, and each element of *Neighbors*(h_p) is associated with a single vertex of *Children*(h_p), the

(a) Unbalanced dependencies   **(b)** Edge-balanced dependencies

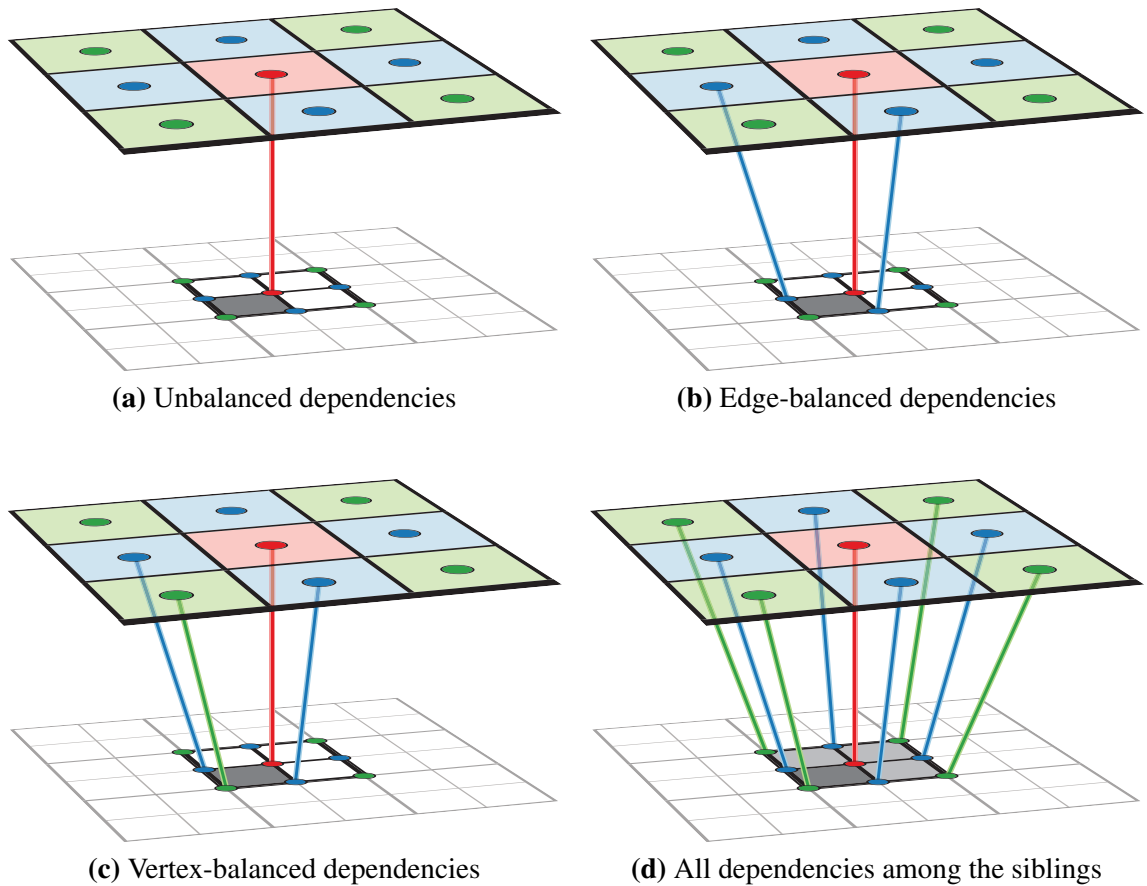**(c)** Vertex-balanced dependencies   **(d)** All dependencies among the siblings

Figure 10.2: A hypercube (gray) depends on its *immediate predecessors*, a subset of the same-sized neighbors of its parent (red), for k-balanced refinement. A hypercube in an unbalanced hierarchy depends only on its parent (a), while those in a j-balanced hierarchy also depend on the same-sized k-neighbors of its parent, where $j \leq k \leq d$. 1-balanced dependencies include the edge-neighbors (blue, in (b)), while 0-balanced dependencies include the vertex neighbors as well (green in (c)). The set of dependencies for all sibling hypercubes (gray) is a subset of the 3^d same-sized neighbors (d) surrounding their common parent hypercube (red). Note the one-to-one correspondence between vertices of the sibling nodes and neighbors of the parent.

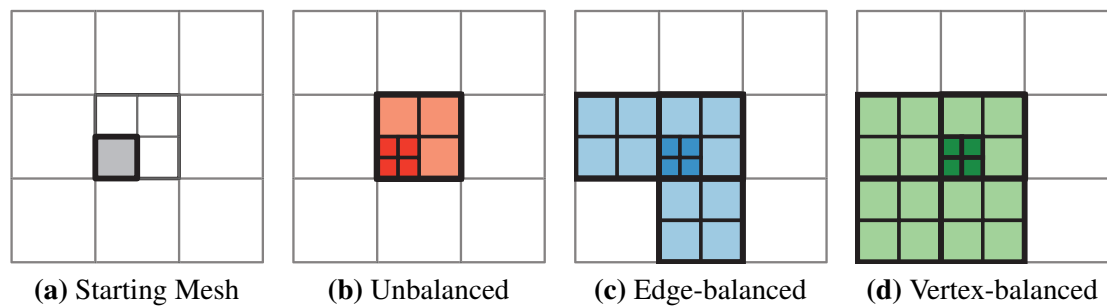**(a)** Starting Mesh   **(b)** Unbalanced   **(c)** Edge-balanced   **(d)** Vertex-balanced

Figure 10.3: Balanced refinement of a hypercube (gray square in (a)) following the k-balanced dependency relation (see Figure 10.2). (a) Original mesh. (b) Unbalanced refinement. (c) Edge-balanced refinement. (d) Vertex-balanced refinement in 2D.

number of *immediate predecessors* of h is at most 2^d. Due to the regular refinement, only the $\binom{d}{k}$ elements of *Neighbors*(h_p) which share a vertex with h are possible k-neighbors of h. Thus, each cube $h \in \square$ has $\sum_{i=k}^{d} \binom{d}{i}$ immediate predecessors in a k-balanced hypercube hierarchy. In particular, the number of immediate predecessors is

- 2^d for vertex-balanced hierarchies;

- $(2^d - 1)$ for edge-balanced hierarchies;

- $(d + 1)$ for facet-balanced hierarchies; and

- 1 for unbalanced hierarchies.

Note that only the unbalanced hierarchy defines a containment relation.

Figure 10.2 illustrates the dependency relations of a hypercube in unbalanced (Figure 10.2a), edge-balanced (Figure 10.2b), and vertex-balanced (Figure 10.2c) hypercubic hierarchies (in 2D). Figure 10.3 illustrates the portion of a nested hypercubic mesh (i.e. quadtree) obtained by refining a square (gray square in Figure 10.3a) according to unbalanced (Figure 10.3b), edge-balanced (Figure 10.3c) and vertex-balanced (Figure 10.3d) refinement.

10.2   Encoding hypercube hierarchies and their extracted meshes

In this section, we propose a compact pointerless encoding for hypercube hierarchies and for their extracted nested hypercubic meshes that is inspired from the diamond hierarchy encoding of Chapter 6. We consider a regularly sampled d-dimensional hypercubic domain Ω covering an integer grid of $(2^N + 1)^d$ samples, where N is the maximum level of resolution, and the samples have integer coordinates in the range $[0, 2^N]$.

Our encoding for k-balanced hypercubic hierarchies (which can also be d-balanced and thus unbalanced) and their extracted meshes is based on the observation that the hypercubes in a hypercubic hierarchy are in one-to-one correspondence with the 0-diamonds in a hierarchy of diamonds (see Section 4.4). This is also true for nested hypercubic meshes, where the correspondence is with the 0-diamonds in a diamond mesh. Due to the coherence among cells of the hypercubic mesh, we utilize supercube clustering to reduce the storage requirements. We first briefly review the encoding from Chapter 6 and highlight differences for hypercube hierarchies and their extracted meshes.

10.2.1   Encoding hypercubes

Our encoding for hypercubes depends on three quantities: the *scale* γ, the *type* τ and the supercube origin *s*. These can be efficiently extracted from the binary representation of the unique midpoint of a hypercube through bit shifting operations.
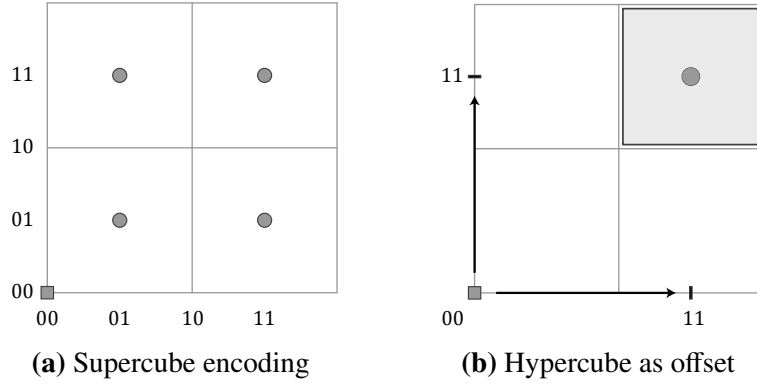
(a) Supercube encoding   **(b)** Hypercube as offset

Figure 10.4: A supercube **s** indexes 2^d *sibling* hypercubes that are generated concurrently (circles in (a)). Each hypercube type τ can be interpreted as an offset from the origin (square in lower left corner) of a supercube (b).

Let

$$
\mathbf{v}_c =
\begin{bmatrix}
\mathbf{x}_1 &=& x_1^1\, x_1^2 \ldots x_1^{m-1}\, x_1^m & \tau_1^1\, 1 & 00\ldots0 \\
\mathbf{x}_2 &=& x_2^1\, x_2^2 \ldots x_2^{m-1}\, x_2^m & \tau_2^1\, 1 & 00\ldots0 \\
& & \vdots & & \\
\mathbf{x}_d &=& \underbrace{x_d^1\, x_d^2 \ldots x_d^{m-1}\, x_d^m}_{\mathbf{s}} & \underbrace{\tau_d^1\, 1}_{\tau} & \underbrace{00\ldots0}_{\gamma}
\end{bmatrix}^T
\tag{10.1}
$$

be the binary representation of the midpoint $\mathbf{v}_c$ of a hypercube $h \in \square$.

The scale γ of h is encoded by the number of trailing zeros among the coordinates $\mathbf{x}_i$ of $\mathbf{v}_c$ (see γ in Equation 10.1). In contrast with the level ℓ of h, which encodes the length of a path from the root hypercube of $\square$ to h, the scale γ of h encodes the length of a path from h to the leaf nodes of $\square$. The level and scale are therefore related through the predetermined maximum resolution N as $\ell = N - \gamma$.

The two bits at position $\gamma + 1$ and $\gamma + 2$ of each coordinate $\mathbf{x}_i$ of $\mathbf{v}_c$ uniquely encode the *type* τ of h. Since h corresponds to a 0-diamond in a hierarchy of diamonds, the rightmost bits of τ (i.e. the bits in position $\gamma + 1$ of each coordinate) will all be 1.

Finally, the remaining m bits in each coordinate correspond to the origin (i.e. the lower left corner) of the supercube indexing h as well as its siblings (see Figure 10.4). Observe that the supercubes at each scale γ define a regular grid of resolution $2^{\gamma+2}$. If we consider the coordinates of the midpoints of all 2^d siblings of h, the only difference will be in the leftmost bits of τ (i.e. the bits in position $\gamma + 2$ of each coordinate). Thus, the d bits τ_i^1 provide an implicit index on the 2^d siblings within a supercube.

Offsets to the midpoints of a cube's faces can be determined as *scaled offsets* from its midpoint. Let h_i be an i-face of a cube h. Then, the offset $\vec{g} = 2^\gamma \cdot \vec{f}$ from the midpoint $\mathbf{v}_c$ of h to the midpoint $\mathbf{v}_i$ of h_i will contain $(d - i)$ non-zero components. I.e.

$$
\mathbf{v}_i = \mathbf{v}_c + 2^\gamma \cdot \vec{f},
\tag{10.2}
$$

where the components $f_j \in \{-1, 0, 1\}$, and $\sum |f_j| = (d - i)$. In particular, the offsets to the 0-faces (vertices) of a cube can be obtained by adding the 2^d scaled vectors whose components f_j are all nonzero, i.e. $\vec{f}_{vertex} = (\pm 1, \pm 1, \ldots, \pm 1)$. These offsets can be precomputed and stored in a lookup table, or can be generated on demand at runtime.

10.2.2 Encoding dependency relations

The dependency relation of a hypercube h in a hierarchy of hypercubes can also be implicitly encoded in terms of scaled offsets from the midpoint of h. Since all nodes in ⊡ are hypercubes, their midpoints will have the same form as Equation 6.1.

Parent. Since the parent h_p of a hypercube h is at one level of resolution lower in the hierarchy, its scale is $\gamma + 1$. Thus, the rightmost bit of the supercube origin (i.e., position x^m in Equation 6.1) of h becomes the leftmost bit of the type τ of h_p. The coordinates of the midpoint of h_p can be obtained from those of h by clearing the rightmost bits of τ in $\mathbf{v}_c$ (i.e. those at position $\gamma + 1$ from the right), and setting the leftmost bits of τ in $\mathbf{v}_c$ to 1.

Children. In contrast, the midpoints of the 2^d children of h share the leftmost $m + 1$ bits as well as the final $\gamma - 1$ bits of each coordinate of h. The bits corresponding to the type τ of a child h_c of h will be at positions γ and $\gamma + 1$ (from the right). The 2^d children are distinguished by the bits at position $\gamma + 1$ of each coordinate, which take on all 2^d possibilities, while the bits at position γ are all set to 1.

Immediate predecessors. Since the immediate predecessors of h are all neighbors of its parent h_p, their scale is $\gamma + 1$. They can differ from h_p in their supercube origin and in the leftmost bit of τ.

Recall that the midpoint of each such k-neighbor of h_p is located at an offset $\vec{f}$ from the midpoint $\mathbf{v}_p$ of h_p, and this offset is twice the distance from $\mathbf{v}_p$ to a vertex of its child h. Thus, the immediate predecessors of h for k-balancing can be determined from the k-neighbors of its parent h_p. For a k-balanced dependency relation, all such offsets to the midpoint $\mathbf{v}'$ of an immediate predecessor h' have the form

$$v' = v_p + 2^{\gamma+1} \vec{f}$$

where $\vec{f}$ encodes the difference in offsets from v_p to a vertex of h, and $\sum |f_j| \leq k$.

Table 10.1 summarizes the number of bits in each component of the midpoints of a hypercubes parents, children and immediate predecessors. In all cases, the type τ is defined by two bits, of which, the right bit is set to 1. Note that the coordinates of the midpoint of the parent and the immediate predecessors can differ in supercube origin ($\mathbf{s}$) as well as in (the left bits of) the type (τ).

10.2.3 Encoding k-balanced hypercubic meshes

We propose a supercube-based encoding for the leaf nodes of a nested hypercubic mesh extracted from a hierarchy of k-balanced hypercubes. Our encoding will only consider

Table 10.1: Number and position of bits in the midpoint coordinates of a cube's hierarchical relationships. Compare to Equation 10.1.

Relationship	Supercube (left)	Type (middle)	Scale (right)
Hypercube	m	2	γ
Children	$m + 1$	2	γ-1
Parent	$m - 1$	2	γ+1
Predecessor	$m - 1$	2	γ+1

the presence or absence of a hypercube within a mesh (as in Chapter 8), but associating information with each node is straightforward.

Let B_c be the number of bytes required to encode each coordinate. Using the encoding from Section 10.2.1, each d-dimensional hypercube can be identified entirely from the coordinates of its midpoint using $(d \cdot B_c)$ bytes, so a hypercubic mesh C containing $|h|$ hypercubes requires $|h| \cdot (d \cdot B_c)$ bytes to encode.

Since all 2^d children of a hypercube are generated concurrently during its refinement, a supercube-based representation for nested hypercubic meshes is able to exploit the spatial and hierarchical coherence of the regular refinement operation. Furthermore, since k-neighbors in a j-balanced hypercubic mesh, $(0 \leq j \leq k < d)$ are required to be within one level of refinement, we expect more coherence among the nodes for lower values of j. A supercube-based representation requires $d \cdot B_c$ bytes to encode the origin coordinates of all supercubes with at least one hypercube in the mesh C, as well as some bookkeeping to encode which of its 2^d children are actually present in the mesh. For simplicity in encoding and updating, we assume that we require one bit for each of the sibling hypercubes, for a total of 2^d bits (i.e. 2^{d-3} bytes) of bookkeeping per supercube. Thus, the cost for encoding the cells of a supercube-based nested hypercubic mesh defined by $|s|$ supercubes is: $|s| * (d \cdot B_c + 2^{d-3})$ bytes. In practice, we store the set of supercubes at each level of resolution as a separate hash table, and there is some storage overhead related to the load factor of the hash table (as discussed in Section 7.6).

To evaluate the cost of this representation, we consider three categories of nodes in a 2^d-tree.

I. Leaf nodes of the tree. Each leaf corresponds to a hypercube in the mesh.

II. Internal nodes of the tree with at least one child that is a leaf node.

III. Internal nodes of the tree whose children are all internal nodes.

Given a *complete* 2^d-tree of maximum level or resolution N, the number of leaf nodes (i.e. type I) is $(2^d)^N$, and the number of internal nodes (i.e. types II and III) is $\frac{(2^d)^N - 1}{2^d - 1}$ [CLRS01]. The number of internal nodes with leaf children (i.e. type II) is therefore $(2^d)^{N-1}$, while the number of internal nodes without children (type III) is $\frac{(2^d)^{N-1} - 1}{2^d - 1}$.

In the classic pointer-based 2^d-trees encoding all nodes are represented [Sam06]. Linear quadtrees [Gar82a] do not encode the internal nodes, and thus only encode nodes of type I, corresponding to hypercubes in the mesh. Our cube-based encoding above is

similar to this approach. In contrast, autumnal quadtrees [FM86] encode nodes of type II and III, that is, each node is encoded in terms of its parent.* The proposed supercube-based representation requires storage for nodes of type II, each of which corresponds to a supercube in the mesh.

Table 10.2 lists the number of hypercubes $|h|$ (nodes of type I) and supercubes $|s|$ (nodes of type II) as well as the concentration of the supercube clustering ($\mathbf{C} = |h|/|s|$) in k-balanced cubic meshes, $0 \leq k \leq 3$ extracted from volumetric scalar fields using a range-based query based on a given isovalue as well as a uniform error criterion of 0. Interestingly, even unbalanced meshes ($k = 3$) have a very high concentration averaging more than 7 out of the possible 8 cubes per supercube. As expected, the concentration increases as the degree of balancing increases (i.e. as k decreases).

We observed (not listed in Table 10.2) that nodes of type III (internal nodes without children in the mesh) comprise only 2-4% of the internal nodes, and less than 0.5% of the total nodes. Thus, they do not add a significant overhead to the representation. However, since we only require the presence or absence of a hypercube for our application, the pointer-based autumnal representation would incur an overhead of 2^d pointers per supercube instead of 2^d bits in our application.

Since our hypercube-based data structure requires 6 bytes per hypercube, and our supercube-based data structure requires 7 bytes per supercube, and the concentration of the supercube-based representation is between 7 and 7.5, our proposed supercube-based pointerless representation requires about 1/6 the storage space as the hypercube-based pointerless representation for these datasets.

In applications where we need to encode information about the vertices of the mesh. Such information could be limited to the presence of a vertex, or we could be encoding scalar values or other information. In general, an 2^d-tree mesh will correspond to a coherent subset of the vertices of a full DMSF. Thus, the partial DMSF model of Section 7.3 will be a useful representation for our vertices. As was the case with diamond meshes, for top-down traversals of the hierarchy, vertices are only inserted into the mesh and are never removed.

10.3 Triangulating nested hypercubic meshes

In this section, we introduce a simple dimension–independent algorithm to triangulate a nested hypercubic mesh 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 C. This converts C into a conforming *RSB* mesh Σ covering the domain of C, i.e. Σ is a simplicial complex defined by a collection of *Regular Simplex Bisection (RSB)* simplices. The properties of RSB simplices [Mau95] 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 $2^d d! = (2d)!!$.

---

*The name is due to the fact that the leaf nodes are 'dropped' in this representation [Wil88].

Table 10.2: Comparison between nested cubic meshes extracted from k-balanced hypercube hierarchies ($0 \leq k \leq 3$) over volumetric scalar fields, where the selection criterion is based on a uniform approximation error of 0% error as well as an isovalue κ. For each mesh, we list the number of hypercubes $|h|$ and supercubes $|s|$ as well as the concentration **C**.

Dataset	Unbalanced			Facet-balanced			Edge-balanced			Vertex-balanced		
	$|h|$	$|s|$	**C**	$|h|$	$|s|$	**C**	$|h|$	$|s|$	**C**	$|h|$	$|s|$	**C**
Bunny$_0$	381 K	54.3 K	7.01	439 K	61.5 K	7.14	467 K	64.9 K	7.19	477 K	66.2 K	7.21
Fuel$_{7.2}$	12.8 K	1.78 K	7.17	14.7 K	2.01 K	7.30	15.3 K	2.08 K	7.34	15.5 K	2.10 K	7.35
Engine$_{100}$	726 K	103 K	7.07	827 K	115 K	7.21	850 K	117 K	7.24	857 K	118 K	7.25
Buckyball$_{128}$	215 K	30.3 K	7.08	237 K	33.0 K	7.19	249 K	34.5 K	7.22	252 K	34.9 K	7.23
Armadillo$_0$	155 K	22.1 K	7.01	182 K	25.5 K	7.13	195 K	27.2	7.20	200 K	27.7 K	7.22
Plasma$_{1.2}$	77.4 K	10.9 K	7.13	82.6 K	11.5 K	7.22	84.4 K	11.7 K	7.24	85.0 K	11.7 K	7.26
Aneurysm$_{128}$	186 K	26.0 K	7.14	239 K	33.2 K	7.22	268 K	36.8 K	7.28	275 K	37.7 K	7.31
Tooth$_{650}$	148 K	21.0 K	7.04	170 K	23.8 K	7.16	181 K	25.1 K	7.21	184 K	25.4 K	7.23
Head$_{58}$	648 K	91.7 K	7.06	748 K	104 K	7.19	784 K	108 K	7.23	791	109 K	7.24
Hydrogen$_{24}$	50.0 K	7.04 K	7.10	58.5 K	8.14 K	7.19	62.4 K	8.60 K	7.26	63.9 K	8.77 K	7.28
Foot$_{23.5}$	3.35 M	446 K	7.50	3.47 M	461 K	7.53	3.51 M	466 K	7.54	3.52 M	467 K	7.55
Bonsai$_{35}$	1.13 M	156 K	7.23	1.26 M	172 K	7.31	1.31M	179 K	7.34	1.32 M	180 K	7.35
Neghip$_{59}$	32.2 K	4.5 K	7.11	36.4 K	5.05 K	7.21	38.0 K	5.23 K	7.25	38.3 K	5.28 K	7.25
Heptoroid$_0$	475 K	66.6 K	7.14	529 K	73.4 K	7.20	558 K	77.0 K	7.24	567 K	78.2 K	7.25
CT Head$_{650}$	589 K	82.1 K	7.17	664 K	91.4 K	7.27	691 K	94.6 K	7.30	699 K	95.7 K	7.31

Our algorithm consists of three stages. First, we *edge-balance* the mesh (Section 10.3.1). This ensures that each face of a hypercube within C is refined by at most one internal vertex (see Figure 10.1). Next, we iterate through the vertices of the edge-balanced mesh and cache them (Section 10.3.2). 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 10.3.3). 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 C. Our algorithm is summarized in Figure 10.5(a)-(c).
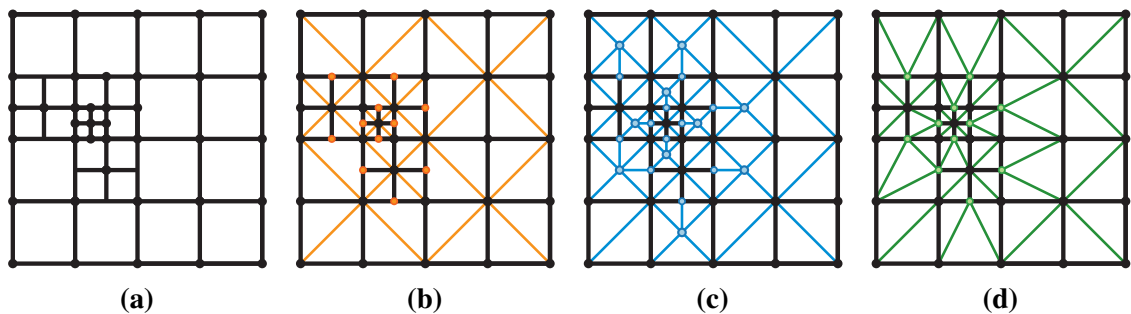

| (a) | (b) | (c) | (d) |

Figure 10.5: Given an edge-balanced hypercubic mesh (a) we first generate complete 0-diamonds for each hypercube (b), and then apply our local bisection-based algorithm to each hypercube (c). Observe that hypercubes that have neighbors at a deeper level of resolution (orange vertices in (b)) need to refine. Compare to a Delaunay-based triangulation (d).
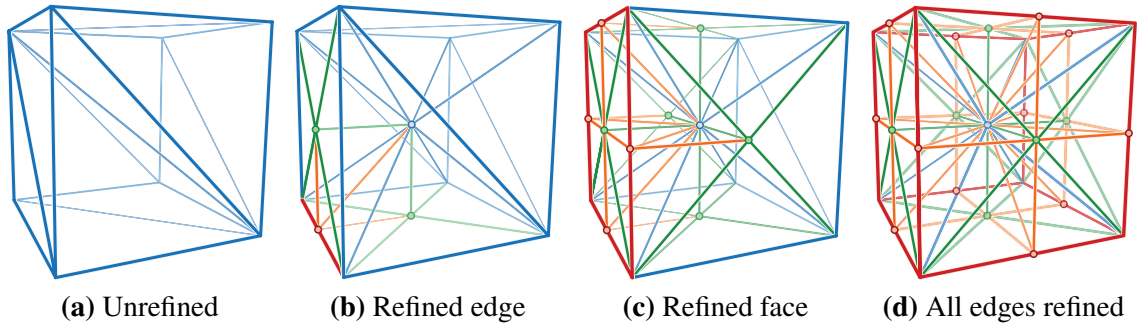
| **(a)** Unrefined | **(b)** Refined edge | **(c)** Refined face | **(d)** All edges refined |

Figure 10.6: The bisection-based triangulation of a hypercube (shown in 3D) depends entirely on the refined edges of neighboring hypercubes. The triangulation ranges from a that of a Kuhn-subdivided cube (a) to that of a fully-subdivided cube (d).

10.3.1 Mesh balancing

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

For our triangulation algorithm, we require 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 10.1a, 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 C_{in} is not edge-balanced, we can convert it to an edge-balanced mesh C in a bottom-up manner by following the direct dependency relation of the edge-balanced hypercube hierarchy (as introduced in Section 10.1.2). Note that this increases the number of hypercubes in the mesh by at most a constant factor (assuming a fixed dimension d) [Wei81, Moo95], and is uniquely defined [Moo95].

10.3.2 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 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 10.1b and 10.5a). Once we determine if v_m exists in C, we no longer require these neighbor-finding operations.

We therefore cache the vertices of C in a hash-table, (see Algorithm 10.1). 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 $O(1)$.

10.3.3 Hypercube triangulation

Our triangulation algorithm (see Algorithm 10.2) generates a globally conforming RSB mesh Σ through a local triangulation Σ_h of each hypercube $h \in C$. This triangulation is

---

Algorithm 10.1 CacheVertices(C)

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

Algorithm 10.2 TriangulateHypercubicMesh(C)

Require: C is an *edge-balanced* nested hypercubic mesh.
Require: *Vertices*(C) contains all vertices of hypercubes in C.
Require: Σ_h is an RSB mesh covering hypercube $h \in C$.
Ensure: $\Sigma = \bigcup_{h \in C} \{\Sigma_h\}$ is a conforming RSB mesh covering C.
 1: **for all** hypercubes $h \in C$ **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*(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).

---

entirely determined from a hypercube's refined edges.

We first convert each hypercube $h \in C$ to an RSB mesh Σ_h defined by the 0-diamond associated with h (lines 2–4 of Algorithm 10.2). Since this is a Kuhn-subdivision of h (see Section 4.4), it contributes $d!$ simplices to Σ_h. See Figure 10.6a 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 10.3.2, we can determine the refined edges of a hypercube by checking if the midpoint of each edge is a vertex in C (lines 6–7).

The function LocalRefineDiamond(δ_e, h) (Algorithm 10.3) 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 [WD09a] for more details).

Figure 10.6 shows some possible triangulations Σ_h of a 3-cube h. In Figure 10.6a, 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 C are at the same level of refinement or one level higher in the hierarchy.

In Figure 10.6b 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

---

Algorithm 10.3 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 $\delta_p \in \text{Parents}(\delta)$ **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).

---

edge-neighbor of h that is not a facet-neighbor, is refined.

Figure 10.6c 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 10.6d illustrates the triangulation Σ_h of h when all edge-neighbors of h are refined. Σ_h is a *fully-subdivided hypercube*, 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 10.2 always produces a simplicial complex. Furthermore, the complexity of the generated mesh Σ with respect to the input hypercubic mesh C is bounded by a constant that depends only on the dimension d of the domain.

Theorem 10.3.1. *Given an edge-balanced hypercubic mesh C defined by $|h|$ hypercubes, Algorithm 10.2 generates a conforming RSB mesh $\Sigma = \bigcup_{h \in C} \{\Sigma_h\}$ 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 C, as well as the properties of Kuhn-subdivided and fully-subdivided hypercubes (see Chapter 4). 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 10.2 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 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 10.2, 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 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 C is a uniform resolution hypercubic mesh.

Similarly, since each edge (i.e. the $(d-1)$-faces) of a hypercube in 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. $\square$

10.3.4 Results

As a proof of concept, we demonstrate the bisection-based algorithm of Section 10.3 in an adaptive 3D isosurfacing application. We compare triangulations extracted from edge-balanced cubic meshes using bisection-based and Delaunay-based triangulations as well as triangulations extracted from a corresponding hierarchy of diamonds (see Table 10.3). In each case, C is a nested hypercubic mesh (in 3D), Σ_h is its associated bisection-based triangulation (extracted using Algorithm 10.2), Σ_p is its associated Delaunay-based triangulation (using the Algorithm of Plantinga and Vegter [PV07]) 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 [PV07], 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 10.3, 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 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. 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 cubic faces of a 4-cube.

From Table 10.3, we see that nested cubic meshes C require approximately 66%

Table 10.3: 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, C is an edge-balanced nested hypercubic mesh, Σ_h is a conforming diamond mesh generated from C using Algorithm 10.2, Σ_p is the tetrahedral mesh extracted from C using the Delaunay-based triangulation algorithm of Plantinga and Vegter [PV07] and Σ_d is a diamond mesh extracted from the corresponding *hierarchy of diamonds*.

| Dataset	N	Mesh type	Vertices $|v|$	Primitives $|h|$ or $|\delta|$	Tetrahedra $|\sigma|$
Fuel	6	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	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	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	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	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	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	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	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

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 10.7, which illustrates a cubic mesh extracted from the 201^3 Bunny dataset (Figure 10.7a) as well as its bisection-based triangulation (Figure 10.7b), and the diamond mesh extracted from a corresponding hierarchy of diamonds (Figure 10.7c), for the isosurface depicted in Figure 10.7d.
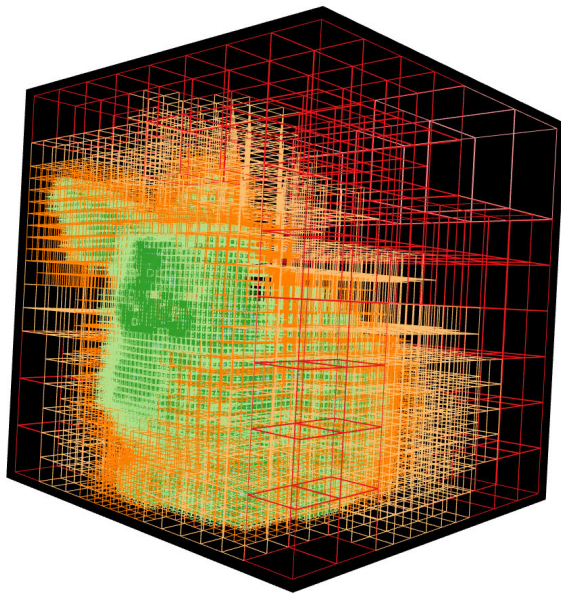
(a) Nested cubical mesh, C
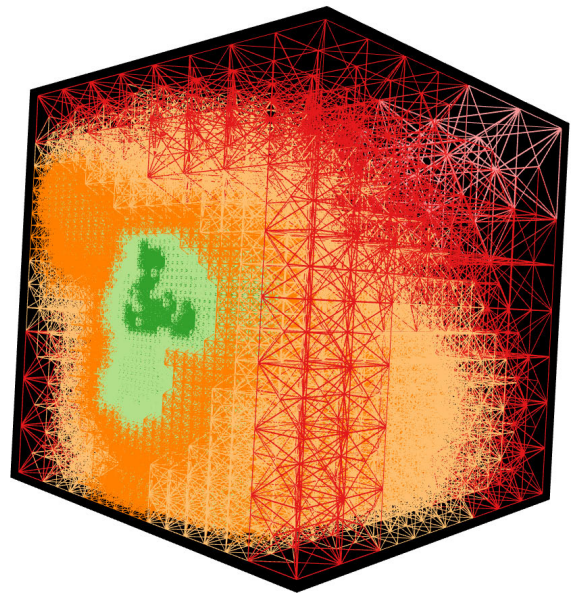
(b) Bisection-based triangulation, Σ_h
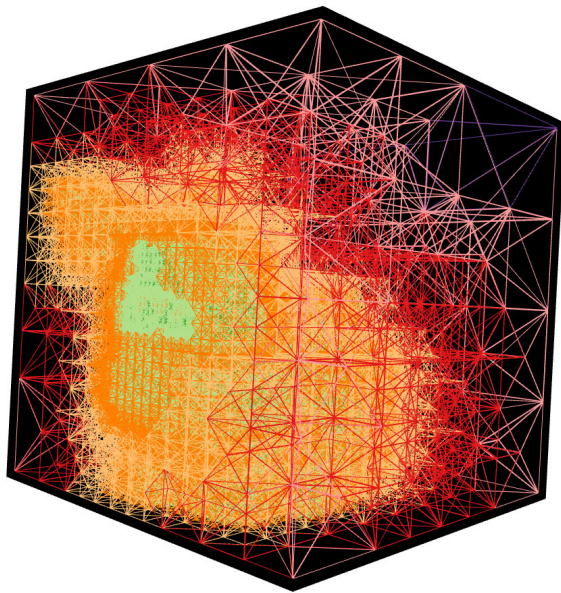
(c) Diamond mesh, Σ_d

(d) Extracted isosurface

Figure 10.7: Decompositions of the 201^3 bunny dataset (a-c) associated with isovalue $\kappa = 0$ (d), colored by level of resolution. An edge-balanced cubic 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.

Chapter 11

Conclusions

In this thesis, we have introduced several diamond-based approaches for scientific visualization.

Diamond hierarchies of arbitrary dimension. We have formalized the notion of *diamonds* of arbitrary dimension [WD09a] via a constructive combinatorial decomposition into a Kuhn-subdivided hypercube and the boundary of a fully subdivided hypercube. Previously, diamonds were explicitly defined in two [DWS+97] and three dimensions [GP00, GDL+02], leading to efficient data structures. While the definition of diamonds in terms of the simplices sharing a common bisection edge, as well as the diamond decomposition paradigm have been previously considered in arbitrary dimension [Pas02], neither a data structure nor an analysis of the combinatorial complexity of these constructs have been previously proposed. Our definition of diamonds has enabled us to develop the first closed-form equations for the number of vertices, simplices, parents and children of a diamond, which we use to define a compact pointerless encoding for hierarchies of diamonds and for their extracted simplicial complexes.

We have also argued that diamonds are the natural representation for extracting, representing and processing conforming RSB meshes. Conforming updates to a simplex-based representations have storage and time complexity that is factorial in the dimension of the underlying domain, while those to a diamond-based representation have complexity that is linear in the dimension.

Our combinatorial decomposition also highlights *parent-child duets*, which are in one-to-one correspondence with the arcs of the diamond dependency graph, as the fundamental building block within conforming RSB meshes. We have demonstrated their utility in defining a compact encoding for adaptive diamond meshes (Chapter 6), in the development of efficient navigation queries on diamond meshes (Chapter 8), and in passing information, such as isovertices, down the hierarchy during refinement (Chapter 9).

Supercubes: A high-level primitive for RSB hierarchies. We have introduced the *supercube* as a high-level primitive for RSB hierarchies [WD08c, WD09b, WD11, YWD11]. In particular, we have enumerated the number of edges, diamonds and vertices in a d-dimensional supercube as $4^d - 2^d$. As discussed in Section 5.3, supercubes reinforce our observations about the factorial nature of simplex-based conforming refinements to RSB meshes and the linear nature of diamond-based conforming refinements. In particular, the average number of d-simplices per diamond in a d-dimensional supercube is $d!$, while the average number of duets per diamond in a d-dimensional supercube is d.

We have demonstrated the effectiveness of supercube-based representations in exploiting the spatial and hierarchical coherence associated with sparse closed subsets of a

hierarchy of diamonds. This has enabled us to define compact efficient data structures for representing a partial diamond-based multiresolution scalar field (DMSF) and for encoding the RSB mesh associated with an active front of a selective refinement query. This solves a long-standing problem in the GIS community for efficiently representing sparse, regularly sampled terrain datasets. We have also demonstrated that many common volumetric datasets are oversampled by a factor of three or more. Thus, the partial DMSF model can significantly reduce the resources required for processing and analyzing today's increasingly large scientific datasets.

Isodiamond hierarchies. We have introduced the *Isodiamond Hierarchy* [WD08b,WD10b] framework as a multiresolution model for an isosurface or interval volume embedded within a DMSF. Isodiamond hierarchies can significantly reduce the storage requirements and processing times for analyzing and visualizing the features of a multiresolution scalar field once significant isovalues within the dataset have been discovered. Furthermore, the minimal isodiamond model achieves a significant reduction in storage space and extraction times by compensating for the relevant, but empty, ancestors of the intersected diamonds.

Hierarchies of balanced hypercubes. We have also analyzed the hierarchical dependency relation required for extracting k-balanced hypercubic meshes from nested hypercubic meshes generated through regular refinement. An examination of the dependency relation through the lens of diamond hierarchies was invaluable in our definition of a multiresolution model for k-balanced hypercubic meshes.

Furthermore, we derived a pointerless encoding for nested hypercubic meshes from our encoding of diamond hierarchies (Chapter 6), leading to a novel representation for 2^d-trees that incorporates the benefits of pointerless representation (i.e. *linear quadtrees* [Gar82a, Gar82b]), with those of leafless representations (i.e. *Autumnal quadtrees* [FM86]). Our results on volumetric datasets indicate that even unbalanced octrees have very high *concentration* with respect to the supercube clustering (more than 7 out of a possible 8 cubes were retained on average), which increases as the degree of balancing within the hypercubic mesh increases.

Finally, we introduced a diamond-based triangulation for nested hypercubic meshes [WD10a] and proved bounds on the sizes of these triangulations. Specifically, the RSB triangulation of each hypercube lies somewhere between that of a Kuhn-subdivided hypercube and a fully-subdivided hypercube. This leads to an overall inflation factor of between $d!$ and $(2d)!!$ simplices in the corresponding RSB mesh for each d-cube in the original mesh. Since the triangulation is a conforming RSB mesh, a diamond-based representation of this mesh is significantly more compact than a simplex-based representation.

11.1 Three families of nested RSB meshes

Supercubes highlight the distinction between the three families of RSB meshes discussed in this thesis.

The *simplex-based representation* is defined over the containment hierarchy induced by RSB operations. It encodes all possible nested RSB meshes achievable by recursively

bisecting a simplex in the original Kuhn-subdivision of a hypercubic domain Ω. Let us call the family of meshes generated by simplex bisection $\mathcal{S}$. Then, every mesh $\Sigma_\sigma \in \mathcal{S}$ can be generated by applying an adaptive refinement algorithm to $\mathcal{K}(\Omega)$ (Algorithm 4.1).

The *diamond-based representation* for RSB meshes is defined over the dependency relation induced by conforming RSB operations. It encodes all possible conforming RSB meshes achievable by subdividing diamonds generated from the 0-diamond defined by the original Kuhn-subdivision of a hypercubic domain. Let us call the family of meshes generated by diamond refinement $\mathcal{D}$. Then, every mesh $\Sigma_\delta \in \mathcal{D}$ can be generated by applying a selective refinement algorithm applied to $\mathcal{K}(\Omega)$ (Algorithm 4.2).

Finally, the *hypercube-based representation* is defined over the dependency relation induced by edge-balanced refinement of hypercubes. It encodes the conforming RSB meshes obtained by triangulating all edge-balanced hypercubic meshes obtained through regular refinement of a hypercubic domain Ω. Let us call this family of meshes $\mathcal{H}$. Then, every mesh $\Sigma_h \in \mathcal{H}$ can be generated through a triangulation of an edge-balanced hypercubic mesh generated from Ω (for example, using Algorithm 10.2).

It has previously been observed in the 2D case that $\mathcal{H} \subset \mathcal{D}$ [DM02].

Since each diamond subdivision is defined in terms of a set of simplex bisections, and since the hypercube triangulation scheme is defined in terms of diamond refinements, the following relationship holds in arbitrary dimension:

$$\mathcal{H} \subset \mathcal{D} \subset \mathcal{S}. \tag{11.1}$$

Note that each relationship defines a proper subset. That is, 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 that are not extractable from a triangulated edge-balanced hypercubic mesh. (i.e. $\Sigma' \in \mathcal{D}$, but $\Sigma' \notin \mathcal{H}$).

The relative complexities of these three families of RSB meshes is highlighted through the supercube primitive. Recall that each supercube uniquely indexes:

- $2^d \cdot (2^d - 1) \cdot d!$ distinct RSB simplices of order d;

- $2^d \cdot (2^d - 1)$ distinct diamond types; and

- 2^d distinct hypercube types.

The simplex-based representation is the most powerful, as $\mathcal{S}$ contains meshes that are not conforming in addition to those that are conforming. It is also the most verbose, by a factor of $d!$, with respect to the class of conforming RSB meshes. However, a simplex-based representation can be an ideal representation in applications, such as point location, that do not require conforming meshes. Alternatively, a lazy refinement evaluation can be applied to local regions in a non-conforming simplex-based representation when crack-free representations are required [AM04].

The diamond-based representation is the basis for all conforming RSB meshes. As all meshes extracted by a selective refinement query on a hierarchy of diamonds, or on a hierarchy of RSB simplices belong to $\mathcal{D}$, the diamond-based representation is the most efficient representation for conforming RSB meshes.

(a) $\Sigma_h \in \mathcal{H}$ **(b)** $\Sigma_d \in \mathcal{D}$ **(c)** $\Sigma_s \in \mathcal{S}$
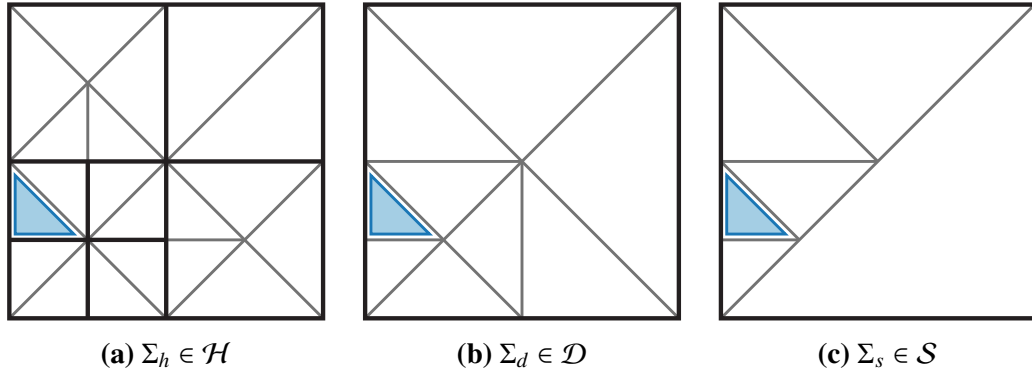
Figure 11.1: Minimal RSB triangulations required to generate a given RSB simplex (blue triangle) for an edge-balanced hypercube hierarchy (a), a hierarchy of diamonds (b) and a hierarchy of simplices (c). Note that $\Sigma_d \notin \mathcal{H}$ since it does not correspond to an edge-balanced hypercubic mesh, and $\Sigma_s \notin \mathcal{D}$ since it is not conforming.

Finally, the hypercube-based representation is the most compact of the three, but is the least powerful representation. In each level of resolution, a supercube can only contain 2^d elements. Due to the increased granularity, RSB meshes in $\mathcal{H}$ are significantly larger than those in $\mathcal{D}$ for the same selection criterion. On the other hand, since there are many existing implementations of quadtrees, octrees and 2^d-trees, the approach outlined in Section 10.3 can be an easy way of generating high-quality simplicial complexes from an existing scientific visualization system based on hypercubes. Figure 11.1 illustrates the smallest member of each family of RSB meshes (in 2D) that contains a given simplex (blue triangle).

11.2 Future work

Recently, there has been an increasing trend in scientific visualization toward the generation and interaction with very large time-varying volume data sets. Such data sets are used as basic tools for analyzing the dynamics and the evolution of phenomena in a variety of application domains, including medicine, meteorology, astrophysics and engineering.

Time-varying volume data sets are sets of points in the 3D Euclidean space describing one or more scalar quantity (e.g. pressure, temperature, strength of an electric or magnetic field) at different instances of time. Approaches to dealing with time-varying data differ in their treatment of the temporal dimension [WD08a]. Values in local regions tend to change slowly over short intervals of time. This temporal coherence can be exploited in interactive applications by efficiently encoding these small changes, thus minimizing storage and retrieval costs. A common metaphor for this approach is that of a video, where users can gain insight into the dataset by 'playing' the discrete snapshots of the volume over time. Decoupling the spatial and temporal components can be advantageous when the spatial and temporal resolutions differ greatly or where there is a higher degree of spatial coherence in a local region than there is temporal coherence, or vice versa. An alternative approach is to treat the temporal and spatial dimensions in a homo-

geneous way. Although this approach can be more complex to conceptualize, it offers several advantages. Since time is assumed to be continuous, smoother animations can be performed by interpolating the field values. Additionally, the correspondence between time steps can be exploited to track features over time.

The huge size and complexity of available time-varying data sets poses interesting challenges for inspecting, analyzing and visualizing such data, as the underlying domain is typically at a much higher resolution than one which could be interactively processed or meaningfully analyzed. This naturally leads to the investigation of hierarchical methods to control and adjust the level of detail of a given data set using a multiresolution model. This enables reducing the geometry in less 'interesting' areas and allows users to focus on a region of interest, thus achieving lower storage costs and better performance.

We believe that a four-dimensional diamond hierarchy would significantly aid in the analysis and visualization of time-varying volumetric data. This would be a natural extension of our work. Supercubes could be used to process the hierarchy, to define the geometric and hierarchical relationships among the elements of the model and to encode its extracted meshes. Due to the large size of such datasets, an isodiamond hierarchy could be used to analyze its isosurfaces and interval volumes offline, or on less powerful workstation, once the interesting cases have been identified. Finally, a duet-based algorithm could be defined to efficiently navigate the connectivity of its extracted meshes.

Appendix A

Double factorial

The *factorial* of a positive number $n \in \mathbb{N}$, denoted $n!$, is recursively defined as:

$$n! = \begin{cases} n \cdot (n-1)! & n > 1 \\ 1 & n = 1. \end{cases}$$

It is the product of all natural numbers from 1 to n.

$$n! = \prod_{i=1}^{n} i.$$

The *double factorial* function [Mes48], denoted $n!!$, is recursively defined as:

$$n!! = \begin{cases} n \cdot (n-2)!! & n > 2 \\ 1 & n \in \{0, 1\}. \end{cases}$$

For even values, $k = 2n, n \in \mathbb{N}$, the double factorial is the product of all even numbers up to n, i.e.,

$$k!! = (2n)!! = \prod_{i=1}^{n} 2i \tag{A.1}$$

and for odd values, $k' = 2n - 1, n \in \mathbb{N}$, the double factorial is the product of all odd numbers up to n, i.e,

$$k'!! = (2n-1)!! = \prod_{i=1}^{n} (2i-1) \tag{A.2}$$

From this, we see that

$$n! = n!!(n-1)!!$$

so the factorial function is the product of successive entries in the double factorial function.

Another relationship between factorials and double factorials can be obtained by separating the terms on the right hand side of Equation A.1

$$(2n)!! = \prod_{i=1}^{n} 2 \cdot \prod_{i=1}^{n} i = 2^n \cdot n! \tag{A.3}$$

Thus, the complexity of the double factorial function is $\Omega(n!)$ for even numbers $k = 2n$.

The first few values of $(2n)!!$, for $n = 0, 1, 2, \ldots$, are thus:

$$1, 2, 8, 48, 384, 3840, \ldots$$

Appendix B

Common terms involving binomials, exponents and factorials

Table B.1: Common terms involving exponents, factorials and binomials.

	n	0	1	2	3	4	5	6
Exponential	2^n	1	2	4	8	16	32	64
	3^n	1	3	9	27	81	243	729
	4^n	1	4	16	64	256	1,024	4,096
	5^n	1	5	25	125	625	3,125	15,625
	$3^n - 1$	0	2	8	26	80	242	728
	$4^n - 2^n$	0	2	12	56	240	992	4,032
	$5^n - 3^n$	0	2	16	98	544	2,882	14,896
Factorial	$n!$	1	1	2	6	24	120	720
	$n!!$	1	1	2	3	8	15	48
	$(2n)!! = 2^n n!$	1	2	8	48	384	3,840	46,080
Binomial $\binom{d}{n} = \frac{d!}{n!(d-n)!}$	$\binom{0}{n}$	1						
	$\binom{1}{n}$	1	1					
	$\binom{2}{n}$	1	2	1				
	$\binom{3}{n}$	1	3	3	1			
	$\binom{4}{n}$	1	4	6	4	1		
	$\binom{5}{n}$	1	5	10	10	5	1	
	$\binom{6}{n}$	1	6	15	20	15	6	1
Supercubes	2^n		2	4	8	16	32	64
	$2^n \cdot (2^n - 1)$		2	12	56	240	992	4,032
	$2^n \cdot (2^n - 1) \cdot n!$		2	24	336	5,760	119,040	2,903,040
	$2^n \cdot (2^n - 1) \cdot n$		2	24	168	960	4,960	24,192

Appendix C

Binomial theorem

C.1 Simplified binomial theorem

In this section, we prove a special case of the binomial theorem,

$$\sum_{k=0}^{n} \binom{n}{k} \cdot x^{n-k} \cdot y^{k} = (x+y)^{n} \tag{C.1}$$

for the case where $x \in \mathbb{N}$ and $y = 1$.

Theorem C.1.1. *Let $\alpha, d, i \in \mathbb{N}$, then*

$$\sum_{i=0}^{d} \binom{d}{i} \cdot \alpha^{d-i} = (\alpha+1)^{d}$$

Proof. We prove this using induction on d. The base case is satisfied by recalling that

$$\binom{0}{0} = \binom{d}{0} = \binom{d}{d} = 1 \tag{C.2}$$

thus, for $d = 0$, $\sum_{i=0}^{0} = \binom{0}{0} \cdot \alpha^{0} = 1 = (\alpha+1)^{0}$.

For the induction case, we need to show that

$$\sum_{i=0}^{d} \binom{d}{i} \cdot \alpha^{d-i} = (\alpha+1)^{d}$$

implies that

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot \alpha^{d+1-i} = (\alpha+1)^{d+1}.$$

We can separate the first and last terms of the summand to obtain:

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot \alpha^{d+1-i} = \binom{d+1}{0} \cdot \alpha^{d+1} + \sum_{i=1}^{d} \binom{d+1}{i} \cdot \alpha^{d+1-i} + \binom{d+1}{d+1} \cdot \alpha^{0}$$

Next, we utilize Pascal's identity

$$\binom{d+1}{i} = \binom{d}{i} + \binom{d}{i-1} \tag{C.3}$$

169

on the middle term, and the identity from Equation C.2 on the outer terms to obtain:

$$\sum_{i=0}^{d+1}\binom{d+1}{i}\cdot\alpha^{d+i-i} = \alpha^{d+1} + \sum_{i=1}^{d}\left(\left[\binom{d}{i}+\binom{d}{i-1}\right]\cdot\alpha^{d+1-i}\right) + 1$$

We can then regroup terms as

$$\sum_{i=0}^{d+1}\binom{d+1}{i}\cdot\alpha^{d+i-i} = \left(\alpha^{d+1} + \sum_{i=1}^{d}\binom{d}{i}\cdot\alpha^{d+1-i}\right) + \left(\sum_{i=1}^{d}\binom{d}{i-1}\cdot\alpha^{d+1-i} + 1\right)$$

Finally, by reversing Equation C.2, pulling an α out from the left term, and converting the index i to $j = i - 1$ on the right term, we obtain:

$$\sum_{i=0}^{d+1}\binom{d+1}{i}\cdot\alpha^{d+i-i} = \alpha * \left(\binom{d}{0}\cdot\alpha^{d} + \sum_{i=1}^{d}\binom{d}{i}\cdot\alpha^{d-i}\right) + \left(\sum_{j=0}^{d-1}\binom{d}{j}\cdot\alpha^{d-j} + \binom{d}{d}\right)$$

Which, after applying the induction hypothesis, gives us the desired result:

$$\sum_{i=0}^{d+1}\binom{d+1}{i}\cdot\alpha^{d+i-i} = \alpha * \sum_{i=0}^{d}\binom{d}{i}\cdot\alpha^{d-i} + \sum_{j=0}^{d}\binom{d}{j}\cdot\alpha^{d-j}$$

$$= \alpha * (\alpha + 1)^{d} + (\alpha + 1)^{d}$$

$$= (\alpha + 1) * (\alpha + 1)^{d}$$

$$= (\alpha + 1)^{d+1}$$

$\square$

As a special case, when $\alpha = 1$, we get the familiar

$$\sum_{i=0}^{d}\binom{d}{i} = 2^{d}$$

i.e. that the binomial coefficients of row d of Pascal's triangle sum to 2^{d}.

Similarly, when $\alpha = 2$, we obtain the result

$$\sum_{i=0}^{d}\binom{d}{i}\cdot 2^{d-i} = 3^{d}.$$

Since a d-cube has 2^{d-i} faces of dimension $d - i$ and there are $\binom{d}{i}$ distinct combinations, we see that a d-cube has a total of 3^{d} faces.

C.2 Related proof

Here we prove that $\sum\left(\binom{d}{i}\cdot 2i\right) = 2^{d}\cdot d$, which we use in Section 5.2.

Theorem C.2.1. *Let $d, i \in \mathbb{N}$, then*

$$\sum_{i=0}^{d} \binom{d}{i} \cdot 2i = 2^d \cdot d.$$

Proof. This proof is very similar in structure of Proof C.1.1, and uses induction on d.

The base case ($d = 0$) is satisfied since $2 \cdot 0 = 2^0 \cdot 0 = 0$.

For the inductive case, we assume

$$\sum_{i=0}^{d} \binom{d}{i} \cdot 2i = 2^d \cdot d \tag{C.4}$$

to prove

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot 2i = 2^{d+1} \cdot (d+1). \tag{C.5}$$

Starting with the left side of the above equation, we first separate the first and last terms from the summand to obtain

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot 2i = 0 + \sum_{i=1}^{d} \binom{d+1}{i} \cdot 2i + 2(d+1).$$

Next, we apply Pascal's identity (Equation C.3) to separate the binomial coefficient as

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot 2i = \sum_{i=1}^{d} \left(\binom{d}{i} + \binom{d}{i-1} \right) \cdot 2i + 2(d+1).$$

We then separate the two summands involving binomials, change the index in the right term to $j = i - 1$, add the initial term ($i = 0$) of 0 to the first sum and incorporate the final term ($j = d$) to the second sum using the $2(d+1)$ summand to obtain:

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot 2i = \sum_{i=0}^{d} \binom{d}{i} \cdot 2i + \sum_{j=0}^{d} \binom{d}{j} \cdot 2(j+1).$$

Finally, distributing $2(j+1)$ to $2j + 2$ and separating the sums in the second term, we obtain

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot 2i = \sum_{i=0}^{d} \binom{d}{i} \cdot 2i + \sum_{j=0}^{d} \binom{d}{j} \cdot 2j + \sum_{j=0}^{d} \binom{d}{j} \cdot 2.$$

After applying the induction hypothesis to the first two terms and the binomial theorem

(Theorem C.1.1) to the third term, we obtain our desired result

$$\sum_{i=0}^{d+1} \binom{d+1}{i} \cdot 2i = 2^d \cdot d + 2^d \cdot d + 2^{d+1}$$
$$= 2^{d+1} \cdot d + 2^{d+1}$$
$$= 2^{d+1} \cdot (d+1).$$

(C.6)

$\square$

Bibliography

[AG79]   E. Allgower and K. Georg. Generation of triangulations by reflection. *Utilitas Mathematica*, 16:123–129, 1979. 42

[Ago05]   M. Agoston. *Computer Graphics and Geometric Modeling*. Springer, 2005. 9

[Ale30]   J.W. Alexander. The combinatorial theory of complexes. *The Annals of Mathematics*, 31(2):292–320, 1930. 50

[AM04]   F.B. Atalay and D.M. Mount. Pointerless implementation of hierarchical simplicial meshes and efficient neighbor finding in arbitrary dimensions. In *Proc. 13th International Meshing Roundtable*, pages 15–26, 2004. 24, 163

[AM07]   F.B. Atalay and D.M. Mount. 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, 24, 35, 48, 50, 71, 84, 106

[AMP00]   D.N. Arnold, A. Mukherjee, and L. Pouly. Locally adapted tetrahedral meshes using bisection. *SIAM Journal on Scientific Computing*, 22(2):431–448, 2000. 23

[Ban70]   T.F. Banchoff. Critical points and curvature for embedded polyhedral surfaces. *American Mathematical Monthly*, 77(5):475–485, 1970. 105

[Bän91]   E. Bänsch. Local mesh refinement in 2 and 3 dimensions. *IMPACT of Computing in Science and Engineering*, 3(3):181–191, 1991. 22, 23

[BDS+03]   K. Bonnell, M. Duchaineau, D. Schikore, B. Hamann, and K. Joy. Material interface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):500–511, 2003. 142

[BEG94]   M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384 – 409, 1994. 6, 19, 143

[Ben75]   J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 21

[Bey95]   J. Bey. Tetrahedral mesh refinement. *Computing*, 55:355–378, 1995. 22, 23

[Bey00]   J. Bey. Simplicial grid refinement: On Freudenthal's algorithm and the optimal number of congruence classes. *Numerische Mathematik*, 85(1):1– 29, 2000. 6, 11, 17, 22, 66, 143

[BG06]   H. Brönnimann and M. Glisse. Octrees with near optimal cost for ray-shooting. *Computational Geometry*, 34(3):182 – 194, 2006. 6, 143

[BGP09]   J. Bösch, P. Goswami, and R. Pajarola. Raster: Simple and efficient terrain rendering on the GPU. In D. Ebert and J. Krueger, editors, *EG 2009 - Areas Papers*, pages 35–42. Eurographics Association, 2009. 29

[BKS97]   P. Bunyk, A. Kaufman, and C.T. Silva. Simple, fast, and robust ray casting of irregular grids. In *Proceedings Scientific Visualization*, pages 30–36, 1997. 105

[Blo00]   J. Blow. Terrain rendering at high levels of detail. In *Proceedings of the Game Developers Conference*, 2000. 30

[BLS04]   D.C. Banks, S.A. Linton, and P.K. Stockmeyer. Counting cases in substitope algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):371–384, 2004. 26

[BLV03]   L. Balmelli, T. Liebling, and M. Vetterli. Computational analysis of mesh simplification using global error. *Computational Geometry Theory and Applications*, 25(3):171–196, 2003. 31, 39, 63

[BPSC04]   R. Borgo, V. Pascucci, R. Scopigno, and P. Cignoni. A Progressive Subdivision Paradigm (PSP). *Proceedings of SPIE*, 5295:223, 2004. 37, 39, 74

[BSW83]   R.E. Bank, A. H. Sherman, and A. Weiser. Refinement algorithms and data structures for regular local mesh refinement. In R. Stepleman, M. Carver, R. Peskin, W. F. Ames, and R. Vichnevetsky, editors, *Scientific Computing, IMACS*, volume 1, pages 3–17. North-Holland, Amsterdam, 1983. 12, 22

[BWC00]   P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in higher dimensions. In *Proceedings IEEE Visualization*, pages 267–273. IEEE Computer Society, October 2000. 26

[BWC04]   P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurface construction in any dimension using convex hulls. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):130–141, 2004. 26, 140

[BZX+04]   P. Bhaniramka, C. Zhang, D. Xue, R. Crawfis, and R. Wenger. Volume interval segmentation and rendering. In *Proceedings Volume Visualization Symposium*, 2004. 26

[CDM+04]   P. Cignoni, L. De Floriani, P. Magillo, E. Puppo, and R. Scopigno. Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):29–45, January-February 2004. 16, 105

[CGG+03a] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. BDAM – Batched Dynamic Adaptive Meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, 2003. 29, 30, 31, 39

[CGG+03b] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Planet-sized Batched Dynamic Adaptive Meshes (P-BDAM). In *Proceedings IEEE Visualization*, pages 147–154. IEEE Computer Society Washington, DC, USA, 2003. 39

[CGG+04]   P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics*, 23(3):796–803, 2004. 39

[CLRS01]   T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT press, 2001. 84, 150

[CMM+97]   P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997. 27, 28

[CNS+06]   A. Castelo, L.G. Nonato, M.F. Siqueira, R. Minghim, and G. Tavares. The J_1^a triangulation: An adaptive triangulation in any dimension. *Computers & Graphics*, 30(5):737–753, 2006. 19

[CQ06]   Y.S. Chang and H. Qin. A unified subdivision approach for multidimensional non-manifold modeling. *Computer Aided Design*, 38(7):770–785, 2006. 22

[CSA03]   H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry Theory and Applcations*, 24(2):75–94, 2003. 25, 26

[dBSvKO97] M. de Berg, O. Schwarzkopf, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 1997. 19

[DDM+05]   E. Danovaro, L. De Floriani, P. Magillo, E. Puppo, D. Sobrero, and N. Sokolovsky. The half-edge tree: A compact data structure for level-of-detail tetrahedral meshes. In *Proceeding of the International Conference on Shape Modeling*, June, 15-17 2005. 62

[DFMP00] L. De Floriani, P. Magillo, and E. Puppo. VARIANT: A system for terrain modeling at variable resolution. *Geoinformatica*, 4(3):287–315, 2000. 2

[DH07] L. De Floriani and A. Hui. Shape representations based on simplicial and cell complexes. In D. Schmalstieg and J. Bittner, editors, *Eurographics 2007 - State of the Art Reports*, pages 63–87, Prague, 2007. 105, 119

[DIM⁺11] L. De Floriani, F. Iuricich, P. Magillo, M.M. Mesmoudi, and K. Weiss. Discrete distortion for 3D data analysis. In L. Linsen, H. Hagen, and B. Hamann, editors, *Visualization in Medicine and Life Sciences (VMLS)*, Mathematics and Visualization. Springer Berlin Heidelberg, 2011. 104

[DKT05] M. Desbrun, E. Kanso, and Y. Tong. Discrete differential forms for computational modeling. In *ACM SIGGRAPH 2005 Courses*. ACM, 2005. 105

[DL04] L. De Floriani and M. Lee. Selective refinement on nested tetrahedral meshes. In G. Brunett, B. Hamann, and H. Mueller, editors, *Geometric Modeling for Scientific Visualization*. Springer Verlag, 2004. 60

[DM02] L. De Floriani and P. Magillo. Multiresolution mesh representation: Models and data structures. In M. Floater, A. Iske, and E. Quak, editors, *Principles of Multi-resolution Geometric Modeling*, Lecture Notes in Mathematics, pages 364–418, Berlin, 2002. Springer Verlag. 15, 16, 23, 143, 163

[DM03] L. De Floriani and P. Magillo. Algorithms for visibility computation on terrains: A survey. *Environment and Planning B - Planning and Design*, 30(5):709–728, 2003. 105

[DP95] L. De Floriani and E. Puppo. Hierarchical triangulation for multi-resolution surface description. *ACM Transactions on Graphics*, 14(4):363–411, October 1995. 11

[DPM97] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multi-resolution modeling. In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 302–323. Springer-Verlag, 1997. 2, 15, 30, 62, 143

[Dur88] M.J. Durst. Letters: Additional reference to marching cubes. *Computer Graphics*, 22(2):72–73, 1988. 25

[DWS⁺97] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: Real-time Optimally Adapting Meshes. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization*, pages 81–88, Phoenix, AZ, October 1997. IEEE Computer Society. 2, 23, 29, 33, 34, 39, 48, 50, 161

[Ede80] H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Technical report, Institut für Informationsverarbeitung, Tech. Univ. Graz, 1980. 27

[EHK+06]   K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-time volume graphics*. AK Peters Ltd, 2006. 105

[EHNP03]   H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. In *Proceedings 19th ACM Symposium on Computational Geometry*, pages 361–370, 2003. 105

[EHZ03]   H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete and Computational Geometry*, 30(1):87–107, 2003. 105

[EKT97]   W. Evans, D. Kirkpatrick, and G. Townsend. Right triangular irregular networks. Technical Report TR97-09, University of Arizona, Tucson, AZ, USA, 1997. 28

[EKT01]   W. Evans, D. Kirkpatrick, and G. Townsend. Right-triangulated irregular networks. *Algorithmica*, 30(2):264–286, 2001. 2, 6, 23, 28, 39, 48, 50, 84, 106, 143

[EM94]   H. Edelsbrunner and E.P. Mucke. Three-dimensional alpha shapes. *Transactions on Graphics*, 13:43–72, 1994. 26

[ESV96]   F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Proceedings IEEE Visualization*, pages 319–326, 1996. 105

[FM86]   F. Fabbrini and C. Montani. Autumnal quadtrees. *The Computer Journal*, 29(5):472–474, 1986. 151, 162

[FMS95]   I. Fujishiro, Y. Maeda, and H. Sato. Interval volume: A solid fitting technique for volumetric data display and analysis. In *Proceedings IEEE Visualization*, pages 151–158, Los Alamitos, CA, USA, 1995. IEEE Computer Society. 26

[FMST96]   I. Fujishiro, Y. Maeda, H. Sato, and Y. Takeshima. Volumetric data exploration using interval volume. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):144–155, 1996. 26

[Fre42]   H. Freudenthal. Simplizialzerlegungen von beschrankter flachheit. *Annals of Mathematics*, 43(3):580–582, 1942. 22, 42, 66

[Gar82a]   I. Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, 25(12):905–910, December 1982. 84, 150, 162

[Gar82b]   I. Gargantini. Linear octtrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, 20(4):365–374, 1982. 84, 162

[Gar09]   R. Garimella. Conformal refinement of unstructured quadrilateral meshes. In *Proceedings of the 18th International Meshing Roundtable*, pages 31–44. Springer, 2009. 20

[GB99]   D. M. Greaves and A. G. L. Borthwick. Hierarchical tree-based finite element mesh generation. *Int'l Journal for Numerical Methods in Engineering*, 45(4):447–471, 1999. 19

[GCBB01]   M. Gavriliu, J. Carranza, D. Breen, and A. Barr. Fast extraction of adaptive multiresolution meshes with guaranteed properties from volumetric data. In *Proceedings of IEEE Visualization*, pages 295–303, Washington, DC, USA, 2001. IEEE Computer Society. 35

[GDL⁺02]   B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings IEEE Visualization*, pages 475–484. IEEE Computer Society Washington, DC, USA, October 2002. 2, 23, 24, 33, 35, 39, 50, 79, 88, 91, 104, 106, 161

[Ger02]   T. Gerstner. Multiresolution extraction and rendering of transparent isosurfaces. *Computers & Graphics*, 26(2):219–228, 2002. 24, 33, 39

[Ger03a]   T. Gerstner. Multi-resolution visualization and compression of global topographic data. *GeoInformatica*, 7(1):7–32, 2003. 32, 39, 48, 93, 102

[Ger03b]   T. Gerstner. Top-down view-dependent terrain triangulation using the octagon metric. Technical report, Institut für Angewandte Mathematik, University of Bonn, 2003. 31, 39, 63

[GG98]   R. Grosso and G. Greiner. Hierarchical meshes for volume data. In *Proceedings Computer Graphics International*, pages 761–769, 1998. 22

[GG00]   G. Greiner and R. Grosso. Hierarchical tetrahedral-octahedral subdivision for volume visualization. *The Visual Computer*, 16(6):357–369, 2000. 17, 22

[GG06]   T.D. Gatzke and C.M. Grimm. Estimating curvature on triangular meshes. *International Journal on shape Modeling*, 12:1–29, 2006. 105

[GH97]   M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings SIGGRAPH*, pages 209–216, 1997. 35, 37

[Gib98]   S.F.F. Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 1496 of *Lecture Notes in Computer Science*, pages 888–898. Springer, 1998. 26

[GK03]   A. Greß and R. Klein. Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. *Graphical Models*, 66(6):370–397, 2003. 36

[GLE97]   R. Gross, C. Luerig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization*, pages 387–394, Phoenix, AZ, October 1997. IEEE Computer Society. 17

[GMBP10]   P. Goswami, M. Makhinya, J. Bösch, and R. Pajarola. Scalable parallel
out-of-core terrain rendering. In *Proceedings Eurographics Symposium on
Parallel Graphics and Visualization*, pages 63–71, 2010. 31

[GMBW96]   D.M. Greaves, Q.W. Ma, A.G.L. Borthwick, and G.X. Wu. Octree-based
finite element analysis for three-dimensional steep waves. In *Proceedings
International Workshop on Water Waves and Floating Bodies*, 1996. 19

[GMC+06]   E. Gobbetti, F. Marton, P. Cignoni, M. Di Benedetto, and F. Ganovelli.
C-BDAM - Compressed Batched Dynamic Adaptive Meshes for terrain
rendering. *Computer Graphics Forum*, 25(3):333–342, 2006. 31

[GP00]   T. Gerstner and R. Pajarola. Topology-preserving and controlled topology
simplifying multi-resolution isosurface extraction. In *Proceedings IEEE
Visualization*, pages 259–266, 2000. 23, 24, 33, 39, 50, 63, 161

[GR99]   T. Gerstner and M. Rumpf. Multiresolutional parallel isosurface extrac-
tion based on tetrahedral bisection. In *Proceedings Symposium on Volume
Visualization*, pages 267–278. ACM Press, 1999. 2, 23, 24, 33, 39, 49

[GRW00]   T. Gerstner, M. Rumpf, and U. Weikard. Error indicators for multilevel visu-
alization and computing on nested grids. *Computers & Graphics*, 24(3):363–
373, 2000. 23

[GSDJ04]   B. Gregorski, J. Senecal, M.A. Duchaineau, and K.I. Joy. Adaptive extrac-
tion of time-varying isosurfaces. *IEEE Transactions on Visualization and
Computer Graphics*, 10(6):683–694, 2004. 34, 35, 39

[GSDJ09]   B. Gregorski, J. Senecal, M. Duchaineau, and K. I. Joy. Compression and
occlusion culling for fast isosurface extraction from massive datasets. In
*Mathematical Foundations of Scientific Visualization, Computer Graphics,
and Massive Data Exploration*, Mathematics and Visualization, pages 303–
323. Springer, 2009. 34

[Guo95]   B. Guo. Interval set: A volume rendering technique generalizing isosurface
extraction. In *Proceedings IEEE Visualization*, pages 3–10. IEEE Computer
Society Washington, DC, USA, 1995. 26

[HAF+96]   W.L. Hibbard, J. Anderson, I. Foster, B.E. Paul, R. Jacob, C. Schafer, and
M.K. Tyree. Exploring coupled atmosphere-ocean models using Vis5D.
International Journal of High Performance Computing Applications, 10(2-
3):211–222, 1996. 2

[HDJ04]   L.M. Hwa, M.A. Duchaineau, and K.I. Joy. Adaptive 4-8 texture hierar-
chies. In *Proceedings IEEE Visualization*, pages 219–226. IEEE Computer
Society Washington, DC, USA, 2004. 31

[HDJ05]   L.M. Hwa, M.A. Duchaineau, and K.I. Joy. Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):355–368, 2005. 24, 29, 31, 39

[Heb94]   D.J. Hebert. Symbolic local refinement of tetrahedral grids. *Journal of Symbolic Computation*, 17(5):457–472, May 1994. 23, 24, 49, 50, 71, 84

[Heb98]   D.J. Hebert. Cyclic interlaced quadtree algorithms for quincunx multiresolution. *Journal of Algorithms*, 27(1):97–128, 1998. 30, 71, 84

[HK95]   D. J. Hebert and H. Kim. Image encoding with triangulation wavelets. In *SPIE Conference Series*, volume 2569, pages 381–392, 1995. 104

[HS79]   GM Hunter and K. Stieglitz. Operations on images using quadtree. *IEEE Transactions on Patern Analysis and Machine Intelligence*, 1(2):145–153, April 1979. 20

[HSSZ97]   H.C. Hege, M. Seeba, D. Stalling, and M. Zckler. A generalized marching cubes algorithm. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1997. 26, 142

[ISS09]   Y. Ito, A.M. Shih, and B.K. Soni. Efficient hexahedral mesh generation for complex geometries using an improved set of refinement templates. In *Proceedings of the 18th International Meshing Roundtable*, pages 103–115, 2009. 20

[JLSW02]   T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, 2002. 26, 27, 36, 142

[JSW03]   G. Ji, H. W. Shen, and R. Wenger. Volume tracking using higher dimensional isosurfacing. In G. Turk, J. van Wijk, and R. Moorhead, editors, *Proceedings IEEE Visualization*, pages 209–216. IEEE Computer Society, October 2003. 26

[JU06]   T. Ju and T. Udeshi. Intersection-free contouring on an octree grid. In *Proceedings Pacific Graphics*, 2006. 36

[KKDH07]   M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *Proceedings Eurographics Symposium on Geometry Processing*, pages 125–133. Eurographics Association Aire-la-Ville, Switzerland, Switzerland, 2007. 36

[Kno06]   A. Knoll. A short survey of octree volume rendering techniques. *GI Lecture Notes in Informatics*, June 2006. 17

[Kos94]   I. Kossaczký. A recursive approach to local mesh refinement in two and three dimensions. *Journal of Computational and Applied Mathematics*, 55(3):275–288, 1994. 23

[KTY+04]   A. Kimura, Y. Takama, Y. Yamazoe, S. Tanaka, and H Tanaka. Parallel volume segmentation with tetrahedral adaptive grid. *International Conference on Pattern Recognition*, 2:281–286, 2004. 2

[Kuh60]   H.W. Kuhn. Some combinatorial lemmas in topology. *IBM J. Res. Develop*, 4:518–524, 1960. 42, 65

[LC87]   W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings SIGGRAPH*, pages 163–169. ACM Press New York, NY, USA, 1987. 25, 26, 27, 35, 36

[LC10]   P. Lindstrom and J. D. Cohen. On-the-fly decompression and rendering of multiresolution terrain. In *Proceedings of ACM Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 65–73, New York, NY, USA, 2010. ACM. 31

[LDS01]   M. Lee, L. De Floriani, and H. Samet. Constant-time neighbor finding in hierarchical tetrahedral meshes. In *Proceedings International Conference on Shape Modeling*, pages 286–295, Genova, Italy, May 2001. IEEE Computer Society. 23, 24, 39, 49, 50, 84, 106, 118, 119, 120

[LDS04]   M. Lee, L. De Floriani, and H. Samet. Constant-time navigation in four-dimensional nested simplicial meshes. In *Proceedings Shape Modeling International 2004*, pages 221–230. IEEE Computer Society, June 2004. 2, 24, 34, 39, 49, 50, 84

[Lee06]   M. Lee. *Spatial Modeling using Triangular, Tetrahedral and Pentatopic Decompositions*. PhD thesis, The University of Maryland, College Park, 2006. 34

[Lev02]   J. Levenberg. Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings IEEE Visualization*, pages 259–266, Washington, DC, USA, 2002. IEEE Computer Society. 29, 39

[LGP+04]   L. Linsen, J. Gray, V. Pascucci, M. A. Duchaineau, B. Hamann, and K.I. Joy. Hierarchical large-scale volume representation with $\sqrt[3]{2}$ subdivision and trivariate b-spline wavelets. In G. Brunnett, B. Hamann, H. Mueller, and L. Linsen, editors, *Geometric Modeling for Scientific Visualization*, Mathematics + Visualization, pages 359–378. Springer Verlag, Heidelberg, Germany, 2004. 34, 35, 39

[LH98]   Y. Livnat and C. Hansen. View dependent isosurface extraction. In *Proceedings IEEE Visualization*, pages 175–180, 1998. 34

[LH04]   F. Losasso and H. Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. In *Proceedings ACM SIGGRAPH*, pages 769–776. ACM New York, NY, USA, 2004. 31

[LH06]   S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics*, 25(3):579–588, 2006. 104

[Lic99]   W.B.R. Lickorish. Simplicial moves on complexes and manifolds. *Geometry and Topology Monographs*, 2(299-320):314, 1999. 41, 50

[LJ95]   A. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on bisection. *SIAM Journal on Scientific Computing*, 16(6):1269–1291, 1995. 23

[LKR⁺96]   P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time continuous level of detail rendering of height fields. In *Proceedings ACM SIGGRAPH*, pages 109–118, August 1996. 28, 30, 32, 33, 39, 108

[LLVM06]   T. Lewiner, H. Lopes, L. Velho, and V. Mello. Extraction and compression of hierarchical isocontours from image data. *Computerized Medical Imaging and Graphics*, 30(4):231–242, 2006. Medical Imaging and Graphics in SIBGRAPI/SIACG. 37

[LP02]   P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, 2002. 2, 24, 30, 32, 34, 39

[LPD⁺04]   L. Linsen, V. Pascucci, MA Duchaineau, B. Hamann, and KI Joy. Wavelet-based multiresolution with $\sqrt[n]{2}$ subdivision. *Journal on Computing, Special Edition: Dagstuhl Seminar on Geometric Modelling*, 72:129–142, 2004. 2, 34, 35, 39, 104

[LRC⁺02]   D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Computer Graphics and Geometric Modeling. Morgan-Kaufmann, San Francisco, 2002. 16, 23, 38

[LS00]   M. Lee and H. Samet. Navigating through triangle meshes implemented as linear quadtrees. *ACM Transactions on Graphics*, 19(2):79–121, April 2000. 66

[LSJ96]   Y. Livnat, H.W. Shen, and C.R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996. 27

[LVLM04a]   T. Lewiner, L. Velho, H. Lopes, and V. Mello. Hierarchical isocontours extraction and compression. In *17th Brazilian Symposium on Computer Graphics and Image Processing*, pages 234–241, Curitiba, PA, October 2004. 37

[LVLM04b] T. Lewiner, L. Velho, H. Lopes, and V. Mello. Simplicial isosurface compression. In *Vision, Modeling, and Visualization*, pages 299–306, Stanford, CA, November 2004. 37, 39

[Mau95] J. M. Maubach. Local bisection refinement for *n*-simplicial grids generated by reflection. *SIAM Journal on Scientific Computing*, 16(1):210–227, January 1995. 1, 2, 22, 23, 44, 45, 50, 75, 151

[Mau96] J. M. Maubach. The efficient location of neighbors for locally refined n-simplicial grids. In *5th Int. Meshing Roundable*, 1996. 24, 50, 66

[MDM04] S. Marchesin, J.M. Dischler, and C. Mongenet. 3D ROAM for scalable volume visualization. In *IEEE Symposium on Volume Visualization and Graphics*, pages 79–86, 2004. 23, 34, 39, 87

[MDP08] M.M. Mesmoudi, L. De Floriani, and U. Port. Discrete distortion in triangulated 3-manifolds. *Computer Graphics Forum*, 27(5):1333–1340, 2008. 104, 105

[Mes48] B.E. Meserve. Double factorials. *The American Mathematical Monthly*, 55(7):425–426, 1948. 46, 167

[Mit91] W.F. Mitchell. Adaptive refinement for arbitrary finite-element spaces with hierarchical bases. *Journal of computational and applied mathematics*, 36(1):65–78, 1991. 22, 23

[Moo92] D.M. Moore. *Simplicial mesh generation with applications*. PhD thesis, Cornell University, Ithaca, NY, USA, 1992. 6, 46, 143

[Moo95] D. Moore. The cost of balancing generalized quadtrees. In *Proc. ACM Solid Modeling*, pages 305–312. ACM, 1995. 6, 19, 143, 145, 153

[MS93] H. Müller and M. Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9(4):182–199, 1993. 36

[MVT03] V. Mello, L. Velho, and G. Taubin. Estimating the in/out function of a surface represented by points. In *Symposium on Solid Modeling and Applications*, pages 108–114, 2003. 2, 36

[MW95] D. Moore and J. Warren. Adaptive simplicial mesh quadtrees. *Houston J. Math*, 21(3):525–540, 1995. 22, 66

[NB93] P. Ning and J. Bloomenthal. An evaluation of implicit surface tilers. *Computer Graphics and Applications, IEEE*, 13(6):33–41, 1993. 25

[New31] M.H.A. Newman. A theorem in combinatorial topology. *J. London Math. Soc*, s1–6(3):186–192, 1931. 50

[NH91]   G.M. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In *Proceedings IEEE Visualization*, pages 83–91, 1991. 25, 26

[Nie03]   G. M. Nielson. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):283–297, 2003. 25

[Nie08]   G. M. Nielson. Dual marching tetrahedra: Contouring in the tetrahedronal environment. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, F. Porikli, J. Peters, J. Klosowski, L. Arns, Y. Chun, T. Rhyne, and L. Monroe, editors, *Advances in Visual Computing*, pages 183–194. Springer, 2008. 26, 142

[NS97]   G. M. Nielson and J. Sung. Interval volume tetrahedralization. In *Proceedings IEEE Visualization*, pages 221–228, 1997. 26, 130

[NSV09]   R.H. Nochetto, K.G. Siebert, and A. Veeser. Theory of adaptive finite element methods: An introduction. In *Multiscale, Nonlinear and Adaptive Approximation*, pages 409 –542. Springer, 2009. Dedicated to Wolfgang Dahmen on the Occasion of his 60th Birthday. 23

[NY06]   T.S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879, October 2006. 25

[OR97]   M. Ohlberger and M. Rumpf. Hierarchical and adaptive visualization on nested grids. *Computing*, 56(4):365–385, 1997. 23, 36

[Paj98]   R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *Proceedings IEEE Visualization*, pages 19–26, Research Triangle Park, NC, October 1998. IEEE Computer Society. 23, 29, 39, 63, 108

[Pas00]   V. Pascucci. *Multi-dimensional and multi-resolution geometric data-structures for scientific visualization*. PhD thesis, Purdue University, West Lafayette, IN, USA, 2000. Major Professor-Bajaj, Chandrajit L. 71, 84

[Pas02]   V. Pascucci. Slow Growing Subdivision (SGS) in any dimension: Towards removing the curse of dimensionality. *Computer Graphics Forum*, 21(3):451–460, September 2002. 2, 23, 24, 34, 50, 161

[Pas04]   V. Pascucci. Isosurface computation made simple: Hardware acceleration, adaptive refinement and tetrahedral stripping. In *Eurographics/IEEE TVCG Symposium on Visualization (VisSym)*, pages 293–300, 2004. 25, 39

[PB00]   V. Pascucci and C. L. Bajaj. Time-critical isosurface refinement and smoothing. In *Proceedings IEEE Symposium on Volume Visualization*, pages 33–42, Salt Lake City, UT, October 2000. IEEE Computer Society. 37

[PBCF93] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(1):56–102, January 1993. 5, 118, 119

[PC00]   A. Plaza and GF Carey. Local refinement of simplicial grids based on the skeleton. *Applied Numerical Mathematics*, 32(2):195–218, 2000. 22

[PF01]   V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Proceedings ACM/IEEE Supercomputing*, pages 45–45, 2001. 88

[PG07]   R. Pajarola and E. Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605, 2007. 28

[Pom00]   A.A. Pomeranz. ROAM using surface triangle clusters (RUSTiC). Master's thesis, U.C. Davis, 2000. 29, 30, 35, 39

[PP09]   E. Puppo and D. Panozzo. RGB subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):295–310, 2009. 22

[PT90]   B.A. Payne and A.W. Toga. Surface mapping brain function on 3D models. *Computer Graphics and Applications, IEEE*, 10(5):33–41, Sept. 1990. 25

[Pup98]   E. Puppo. Variable resolution triangulations. *Computational Geometry Theory and Applications*, 11(3-4):219–238, 1998. 30, 108

[PV07]   S. Plantinga and G. Vegter. Isotopic meshing of implicit surfaces. *The Visual Computer*, 23(1):45–58, 2007. 19, 156, 157

[RH99]   J. C. Roberts and S. Hill. Piecewise-linear hypersurfaces using the marching cube algorithm. In R. Erbacher and A. Pang, editors, *Visual Data Exploration and Analysis VI, Proceedings of SPIE Visualization 2000*, pages 170–181. SPIE, 1999. 26

[RHSS98] S. Roettger, W. Heidrich, P. Slusallek, and H.P. Seidel. Real-time generation of continuous levels of detail for height fields. In *Proceedings Central Europe Winter School of Computer Graphics (WSCG)*, pages 315–322, 1998. 19

[Riv84]   M.C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20(4):745–756, 1984. 22, 45

[Riv91]   M.C. Rivara. Local modification of meshes for adaptive and/or multigrid finite-element methods. *Journal of Computational and Applied Mathematics*, 36(1):79–89, 1991. 22, 45

[RL92]   M.C. Rivara and C. Levin. A 3D refinement algorithm suitable for adaptive and multigrid techniques. *Communications in Applied Numerical Methods*, 8(5):281–290, 1992. 2

[RMSS01]   G. V. S. Reddy, H. J. Montas, A. Shirmohammadi, and H. Samet. Quadtree-based triangular mesh generation for finite element analysis of heterogeneous spatial data. In *Proceedings of the International ASAE Annual Meeting*, Sacramento, CA, 2001. 19, 143

[Ros99]   J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999. 105

[RS72]   C.P. Rourke and B.J. Sanderson. *Introduction to piecewise-linear topology*, volume 69 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer, 1972. 41

[Sam90]   H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990. 24

[Sam06]   H. Samet. *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 2006. 6, 17, 18, 20, 21, 24, 30, 32, 67, 84, 143, 150

[Sch92]   G. Schrack. Finding neighbors of equal size in linear quadtrees and octrees in constant time. *CVGIP: Image Understanding*, 55(3):221–230, May 1992. 84

[Sch96]   R. Schneiders. Refining quadrilateral and hexahedral element meshes. In *5th International Conference on Grid Generation in Computational Field Simulations*, pages 679–688, Mississippi State University, 1996. 20

[Sew72]   E.G. Sewell. *Automatic generation of triangulations for piecewise polynomial approximation*. PhD thesis, Purdue University, 1972. 23

[Sew79]   E.G. Sewell. A finite element program with automatic user-controlled mesh grading. In R. Vichnevetsky and R.S. Stepleman, editors, *Advances in Computer Methods for Partial Differential Equations III*, pages 8–10. IMACS, 1979. 23

[SFYC96]   R. Shekhar, E. Fayyad, R. Yagel, and J.F. Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings IEEE Visualization*, pages 335–342, Los Alamitos, CA, USA, 1996. IEEE Computer Society. 36

[SHLJ96]   H.W. Shen, C.D. Hansen, Y. Livnat, and C.R. Johnson. Isosurfacing in Span Space with Utmost Efficiency (ISSUE). In *Proceedings IEEE Visualization*. IEEE Computer Society Press Los Alamitos, CA, USA, 1996. 27

[Siv96]   R. Sivan. *Surface modeling using quadtrees*. PhD thesis, University of Maryland, College Park, 1996. 19, 36

[SJW07]   S. Schaefer, T. Ju, and J. Warren. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):610–619, 2007. 26, 36

[SS92]   R. Sivan and H. Samet. Algorithms for constructing quadtree surface maps. In *Proc. 5th Int. Symposium on Spatial Data Handling*, pages 361–370, 1992. 6, 19, 143

[SSB08]   H. Sundar, R.S. Sampath, and G. Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal of Scientific Computing*, 30(5):2675–2708, 2008. 6, 19, 143

[Ste08]   R. Stevenson. The completion of locally refined simplicial partitions created by bisection. *Mathematics of Computation*, 77(261):227–242, 2008. 23

[Sur]   U.S. Geological Survey. Global 30 arc second elevation data. http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html. 89

[SZK95]   R. Shu, C. Zhou, and M.S. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11(4):202–217, 1995. 36

[SZL92]   W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *In Proceedings ACM SIGGRAPH*, 26(2):65–70, July 1992. 35

[Tan95]   H.T. Tanaka. Accuracy-based sampling and reconstruction with adaptive meshes for parallel hierarchical triangulation. *Computer Vision and Image Understanding*, 61(3):335 – 350, 1995. 31, 63, 75

[TO04]   T. Tu and D.R. OHallaron. Balanced refinement of massive linear octrees. Technical Report CMU-CS-04-129, Carnegie Mellon School of Computer Science, April 2004. 19

[Tod76]   M.J. Todd. *The computation of fixed points and applications*. Number 124 in Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 1976. 65

[TOL04]   T. Tu, D. R. OHallaron, and J. C. Lòpez. Etree: A database-oriented method for generating large octree meshes. *Engineering with Computers*, 20:117–128, 2004. 19

[TPG99]   G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: Improved iso-surface extraction. *Computers and Graphics*, 23(4):583–598, 1999. 25

[Tra97]   C. T. Traxler. An algorithm for adaptive mesh refinement in *n* dimensions. *Computing*, 59(2):115–137, 1997. 11, 22, 23

[TTW03]   H. Tanaka, Y. Takama, and H. Wakabayashi. Accuracy-based sampling and reconstruction with adaptive grid for parallel hierarchical tetrahedrization. In *Proceedings Volume Graphics*, pages 79–86. ACM Press, 2003. 75

[Tuc45]   A.W. Tucker. Some topological properties of disk and sphere. In *Proceedings First Canadian Math. Congress, Montreal*, volume 285–309, 1945. 65

[VHB87]   B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Proceedings ACM SIGGRAPH*, pages 103–110, New York, NY, USA, 1987. ACM. 6, 19, 36, 143

[VK96]   M.A. Van Kreveld. Efficient methods for isoline extraction from a TIN. *Geographical Information Systems*, 10(5):523–540, 1996. 27

[Vol]   Volvis library. http://www.volvis.org/. 4, 94, 121

[WB96]   C. Weigle and D. Banks. Complex-valued contour meshing. In *Proceedings IEEE Visualization*, pages 173–180. IEEE Computer Society, October 1996. 2, 26

[WB98]   C. Weigle and D. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *Proceedings IEEE Visualization*, pages 103–110. IEEE Computer Society, October 1998. 26

[WD08a]   K. Weiss and L. De Floriani. Modeling and visualization approaches for time-varying volumetric data. In *Advances in Visual Computing*, volume 5359 of *Lecture Notes in Computer Science*, pages 1000–1010. Springer, 2008. 17, 164

[WD08b]   K. Weiss and L. De Floriani. Multiresolution interval volume meshes. In *IEEE/ EG Symposium on Volume and Point-Based Graphics*, pages 65–72. Eurographics Association, 2008. 34, 37, 39, 138, 162

[WD08c]   K. Weiss and L. De Floriani. Sparse terrain pyramids. In *Proceedings ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 115–124, New York, NY, USA, 2008. ACM. 25, 32, 39, 161

[WD09a]   K. Weiss and L. De Floriani. Diamond hierarchies of arbitrary dimension. *Computer Graphics Forum (Proceedings SGP 2009)*, 28(5):1289–1300, 2009. 25, 34, 154, 161

[WD09b]   K. Weiss and L. De Floriani. Supercubes: A high-level primitive for diamond hierarchies. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Visualization 2009)*, 15(6):1603–1610, November-December 2009. 25, 32, 39, 161

[WD10a]   K. Weiss and L. De Floriani. Bisection-based triangulations of nested hypercubic meshes. In S. Shontz, editor, *Proceedings 19th International Meshing Roundtable*, pages 315–333, Chattanooga, Tennessee, October 3–6 2010. 162

[WD10b]   K. Weiss and L. De Floriani. Isodiamond hierarchies: An efficient multiresolution representation for isosurfaces and interval volumes. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):583 – 598, July-Aug. 2010. 34, 37, 39, 162

[WD10c]   K. Weiss and L. De Floriani. Nested refinement domains for tetrahedral and diamond hierarchies. In *IEEE Visualization 2010 Poster Compendium*, 2010. 63

[WD10d]   K. Weiss and L. De Floriani. Simplex and diamond hierarchies: Models and applications. In H. Hauser and E. Reinhard, editors, *EG 2010 - State of the Art Reports*, pages 113–136, Norrköping, Sweden, 2010. Eurographics Association. 17, 120

[WD11]   K. Weiss and L. De Floriani. Simplex and diamond hierarchies: Models and applications. *Computer Graphics Forum*, 30:(To appear), 2011. 17, 25, 161

[WDSB00]   Z.J. Wood, M. Desbrun, P. Schroder, and D. Breen. Semi-regular mesh extraction from volumes. In *Proceedings IEEE Visualization*, pages 275–282. IEEE Computer Society Press Los Alamitos, CA, USA, 2000. 35

[Wei81]   A. Weiser. *Local-mesh, local-order, adaptive finite element methods with a posteriori error estimators for elliptic partial differential equations*. PhD thesis, Yale University, 1981. 143, 153

[Whi57]   H. Whitney. *Geometric integration theory*. Princeton University Press, 1957. 65

[Wil88]   R. Williams. The goblin quadtree. *The Computer Journal*, 31(4):358–363, 1988. 151

[WKE99]   R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999. 6, 36, 143

[WMD10]   K. Weiss, M.M. Mesmoudi, and L. De Floriani. Multiresolution analysis of 3D images based on discrete distortion. In *International Conference on Pattern Recognition (ICPR)*, pages 4093–4096, Istanbul, Turkey, August 2010. IEEE Computer Society. 104

[WVG90]   J. Wilhelms and A. Van Gelder. Topological considerations in isosurface generation extended abstract. In *Proceedings Workshop on Volume Visualization*, pages 79–86. ACM Press New York, NY, USA, 1990. 25

[WVG92]   J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992. 27, 36, 63, 87

[YWD11]   M.A. Yalçın, K. Weiss, and L. De Floriani. GPU algorithms for diamond-based multiresolution terrain processing. In *Eurographics Symposium on Parallel Graphics and Visualization*, Bangor, Wales, April 10–11 2011. 31, 39, 75, 161

[ZBS03]   Y. Zhang, C. Bajaj, and B.S. Sohn. Adaptive and quality 3d meshing from imaging data. In *Proceedings ACM Symposium on Solid Modeling and Applications*, pages 286–291. ACM Press New York, NY, USA, 2003. 27, 36

[ZCK97]   Y. Zhou, B. Chen, and A. Kaufman. Multiresolution tetrahedral framework for visualizing regular volume data. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization*, pages 135–142. IEEE Computer Society, October 1997. 2, 23, 32, 33, 34, 39, 49, 74

[Zha95]   S. Zhang. Successive subdivision of tetrahedra and multigrid methods on tetrahedral meshes. *Houston Journal of Mathematics*, 21:541–556, 1995. 22

[ZS01]   D. Zorin and P. Schröder. A unified framework for primal/dual quadrilateral subdivision schemes. *Computer Aided Geometric Design*, 18(5):429–454, 2001. 19