# Supercubes: A High-Level Primitive for Diamond Hierarchies

Kenneth Weiss, *Student Member, IEEE*, and Leila De Floriani, *Member, IEEE*

**Abstract**— Volumetric datasets are often modeled using a multiresolution approach based on a nested decomposition of the domain into a polyhedral mesh. Nested tetrahedral meshes generated through the longest edge bisection rule are commonly used to decompose regular volumetric datasets since they produce highly adaptive crack-free representations. Efficient representations for such models have been achieved by clustering the set of tetrahedra sharing a common longest edge into a structure called a *diamond*. The alignment and orientation of the longest edge can be used to implicitly determine the geometry of a diamond and its relations to the other diamonds within the hierarchy. We introduce the *supercube* as a high-level primitive within such meshes that encompasses all unique types of diamonds. A supercube is a coherent set of edges corresponding to three consecutive levels of subdivision. Diamonds are uniquely characterized by the longest edge of the tetrahedra forming them and are clustered in supercubes through the association of the longest edge of a diamond with a unique edge in a supercube. Supercubes are thus a compact and highly efficient means of associating information with a subset of the vertices, edges and tetrahedra of the meshes generated through longest edge bisection. We demonstrate the effectiveness of the supercube representation when encoding multiresolution diamond hierarchies built on a subset of the points of a regular grid. We also show how supercubes can be used to efficiently extract meshes from diamond hierarchies and to reduce the storage requirements of such variable-resolution meshes.

**Index Terms**—Longest edge bisection, diamonds, hierarchy of diamonds, multiresolution models, selective refinement.

## 1 Introduction

Discrete volumetric datasets are often modeled as polyhedral meshes where scalar values are associated with the vertices of the mesh. Due to the increasing sizes of volumetric datasets, a multiresolution model is often used, thus creating a hierarchical spatial decomposition on the vertices of the mesh. Decompositions based on nested cubes, known as octrees, are popular for modeling 3D scalar fields as they allow one to focus resources on regions of interest within the dataset, while aggregating the less relevant regions into larger blocks. However, an octree partitioning may introduce cracks into the scalar field representation. Additionally, variable-resolution representations extracted from octrees have limited adaptability in that each cubic block is replaced with eight new cubes. Longest edge bisection (LEB) hierarchies were introduced as a multiresolution modeling scheme over regular grids to increase adaptability while enforcing crack-free, or *conforming*, modifications to the model. However, this adaptability increases the number of modeling primitives at full resolution where each cube is replaced by six tetrahedra. This problem is mitigated by aggregating the set of tetrahedra involved in each modification to the mesh into a modeling primitive called a *diamond*. The regularity of the decomposition and the structure of the mesh enables implicit encodings for both the hierarchical and geometric relationships among the diamonds in the hierarchy.

Here, we propose the *supercube*, a high-level primitive for the edges of an LEB hierarchy. A supercube is a structured set of edges within an LEB hierarchy that captures the intrinsic symmetry of the model. Whereas previous representations contain several congruence classes of tetrahedra or diamonds, each with different geometric alignments, supercubes are all identical (up to scale). Diamonds and supercubes are related by a one-to-one correspondence from the longest edge of a diamond to a supercube edge. Thus, the set of edges in a supercube corresponds to all distinct *types* of diamonds within the hierarchy.

- *K. Weiss is with the Department of Computer Science, University of Maryland, College Park, 4406 A.V. Williams Building, College Park, MD 20742. E-mail: kweiss@cs.umd.edu.*
- *L. De Floriani is with the Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, Via Dodecaneso, 35, 16146 Genova, Italy. E-mail: deflo@disi.unige.it.*

*Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.*
*For information on obtaining reprints of this article, please send email to: tvcg@computer.org .*

Many operations on LEB hierarchies apply to only a sparse subset of the tetrahedra within the hierarchy. For example, in isosurfacing applications, we are only interested in the subset of tetrahedra that intersect the isosurface. Similarly, large regions of a domain are often oversampled due to the use of a regularly sampled representation. However, implementations of these LEB hierarchies have mostly focused on efficient representations for the entire hierarchy, where the tetrahedra are implicitly indexed. We propose the use of supercubes as containers for data associated with coherent subsets of the elements within an LEB hierarchy. Since this clustering incurs overhead related to the storage of explicit spatial indexes, we consider the number of encoded diamonds with respect to an encoding of the full hierarchy, as well as the number of elements associated with each cluster. We demonstrate the effectiveness of supercubes as a multiresolution model for complete, or sparse volume datasets.

The remainder of this paper is organized as follows. We review related work and the longest edge bisection model in Sections 2 and 3, respectively. In Section 4, we introduce the supercube primitive, while, in Section 5, we discuss efficient encodings for supercubes. In Section 6, we discuss an application of supercubes to the encoding of multiresolution models of 3D scalar fields, while in Section 7 we discuss an efficient encoding of the mesh extracted though selective refinement. We present experimental results in Section 8. Finally, we draw some concluding remarks in Section 9.

## 2 Related work

Multiresolution tetrahedral models for 3D scalar fields based on longest edge bisection (LEB) were introduced for domain decomposition in finite element analysis [10, 17, 22], and have been applied in several contexts, including scientific visualization [31, 23, 6, 8, 16, 25], surface reconstruction [18] and volume segmentation [12].

The containment hierarchy among the tetrahedra in an LEB mesh induces a natural tree representation, in which the nodes are tetrahedra and the two children of a tetrahedron $t$ are the tetrahedra generated by bisecting $t$. 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 [31, 7, 6, 13, 8, 16].

LEB meshes can be non-conforming and, thus, can generate isosurfaces with cracks. This problem can be avoided by simultaneously subdividing all tetrahedra sharing that edge. This can be performed through neighbor finding algorithms [10, 17, 13] or through a precomputed saturated error which implicitly forces all longest edge neighbors to split [7, 6].

An alternative approach is to directly encode the cluster of tetrahedra sharing a longest edge [2, 8, 9, 26, 28]. Gregorski et al. [8] denote such a primitive as a *diamond*, and propose an implicit representation for the spatial and hierarchical relationships of a diamond based on its level, orientation and position. In our implicit diamond representation, the level and orientation can also be derived from the position of the diamond. The diamond paradigm has been generalized to higher dimensions in connection with adaptive mesh generation [19]. A decomposition of an arbitrary dimensional diamond into a pair of axis-aligned hypercubes is introduced in [28] to yield an implicit representation for diamond-based LEB hierarchies over regular grids.

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. They are thus only efficient for representing a complete LEB hierarchy of dimensions $(2^N+1)^3$. However, many operations on LEB hierarchies pertain to only a subset of the tetrahedra. To the best of our knowledge, methods to encode an incomplete LEB hierarchy have only been proposed in 2D for terrain modeling [5, 27]. The former representation [5] uses variable-length pointers to encode the number of containment hierarchy nodes that are skipped when a node is not refined. The latter representation [27] uses a similar clustering strategy to the one presented in this paper for encoding an incomplete hierarchy of 2D diamonds at multiple resolutions.

There have been many proposed optimizations to enable interactive visualization of LEB hierarchies including frame-to-frame coherence [3, 8], parallel rendering [7], front-to-back sorting, view-dependent rendering [16], tetrahedral stripping [20], and chunked updates [9, 1]. Since our model is based on the same hierarchies, these optimizations can be easily incorporated into our pipeline.

Octrees are another popular family of spatial data structures used in visualization [29, 4, 21, 11, 24]. The subdivision rule for an octree node $n$ with side length $\ell$ is to replace $n$ by its 8 children, each with side length $\ell/2$. Since LEB hierarchies extract conforming meshes, they are more closely related to restricted octrees [29, 24] where neighboring nodes can differ by at most one level of resolution.

## 3 Longest Edge Bisection and Diamond Hierarchies

In this Section, we review hierarchical spatial decompositions of the domain of a 3D scalar field based on the longest edge bisection operation applied to a tetrahedral mesh partitioning the domain.

A mesh in which all cells are defined by the uniform subdivision of a cell into scaled copies is called a *nested mesh*. A special class of nested meshes are those generated by bisecting tetrahedra along their longest edge, which we denote as *Longest Edge Bisection (LEB) meshes*. The *bisection rule* for a tetrahedron $\mathbf{t}$ in such a mesh consists of replacing $\mathbf{t}$ with the two tetrahedra obtained by splitting $\mathbf{t}$ along the plane defined by the middle point of its longest edge $\mathbf{e}$ and the two vertices of $\mathbf{t}$ not adjacent to $\mathbf{e}$. When this rule is applied recursively to an initial decomposition of a cubic domain $\Omega$ into six tetrahedra sharing a diagonal of $\Omega$, it generates three congruent *classes* of tetrahedra, each with a single longest edge. We denote the tetrahedra congruent to those sharing a diagonal of the base cube as 0-*tetrahedra*. Tetrahedra congruent to those obtained by splitting a 0-tetrahedron are denoted as 1-*tetrahedra* and have a longest edge aligned with a face diagonal of the base cube. Finally, 2-*tetrahedra* have a longest edge aligned with an edge of the base cube and are congruent to those obtained by splitting a 1-tetrahedron. Tetrahedra obtained by splitting a 2-tetrahedron are congruent to 0-tetrahedra.

Applying the longest edge bisection rule to tetrahedra does not generate conforming meshes. Thus, we consider a clustering of the tetrahedra that need to be subdivided concurrently. Given an LEB mesh $\Sigma$, the set of tetrahedra sharing a common longest edge forms a *diamond*, which we denote as $\delta$. The longest edge of $\delta$ is called its *spine*. Since all tetrahedra within a diamond share their spine, they all belong to the same class of tetrahedra. As a consequence, there are three congruence *classes* of diamonds: those with spines aligned with diagonals of an axis-aligned cube (*0-diamonds*), with face diagonals of such a cube (*1-diamonds*) or with edges of such a cube (*2-diamonds*). An $i$-diamond, for $i \in \{0,1,2\}$ is formed by $\{6,4,8\}$ tetrahedra, and contains $\{8,6,10\}$ vertices, respectively. Figure 1 illustrates the three diamond classes and highlights their spines (black edges), parents (hollow gray vertices) and children (red vertices).

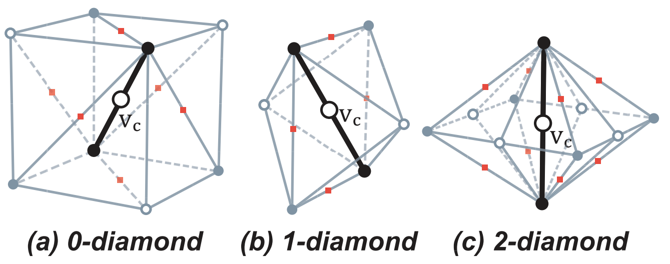


Fig. 1. The three *classes* of diamonds, distinguished by the alignment of their spines (black edges). Central vertices of children coincide with the midpoint of a subset of the diamond's edges (red vertices), while those of parents coincide with a subset of its vertices (hollow gray vertices).

A diamond $\delta$ is subdivided by bisecting all of its tetrahedra according to the longest edge bisection rule, thus doubling the number of tetrahedra within $\delta$. We note that all changes to the LEB mesh $\Sigma$ due to the subdivision of a diamond $\delta$ occur within the interior of the domain of $\delta$ and, thus, the vertices, edges and faces on the boundary of $\delta$ are unaffected by its subdivision. The local effect of such a subdivision is to remove its spine, to add a vertex at the midpoint of its spine, which we denote as the *central vertex* of the diamond, and to add edges from the central vertex to all other vertices of $\delta$.

Let us consider the collection $\Delta$ of all the diamonds associated with the longest edges of the tetrahedra $T$ arising from longest edge bisection to a cubic domain $\Omega$. The containment relation between the tetrahedra in $T$ induces a parent-child dependency relation over the diamonds in $\Delta$. A diamond $\delta_p$ is a *parent* of another diamond $\delta_c$ if and only if some tetrahedra in $\delta_c$ are created by the splitting of at least one tetrahedron in $\delta_p$. If $\delta_p$ is a parent of $\delta_c$, then $\delta_c$ is called a *child* of $\delta_p$.

The set $\Delta$ with the parent-child dependency relation defined over $\Delta$ can be easily shown to define a partial order relation and thus it can be described as a Directed Acyclic Graph (DAG) (see [2] for a proof), whose nodes correspond to diamonds, and whose arcs are defined by the parent-child dependency relation between diamonds. We call the resulting model a *Hierarchy of Diamonds (HD)*. Due to the regularity of the vertex distribution and the subdivision rule, the DAG associated with $\Delta$ has a fixed structure. Thus, with the exception of diamonds whose spines lie on the domain boundary, $\{0,1,2\}$ diamonds always have $\{3,2,4\}$ parents and $\{6,4,8\}$ children, respectively.

## 4 Supercubes

Since we employ longest edge bisection as the subdivision operation, and diamonds are defined by their spine, it is useful to consider the spatial and hierarchical relationships among the edges of an LEB hierarchy. An analysis of this structure reveals a higher level of symmetry within the hierarchy than that which is apparent at the level of diamonds (see Figure 3). Specifically, each level of resolution is tiled by a repeating pattern of edges arranged in a cubic domain, which we call a *supercube*.

Supercubes can be defined constructively by their set of edges. If we consider an empty cubic domain, the edges of a supercube $\sigma$ are formed by

- the edges joining the center of the cube to its eight corners (see Figure 2a),
- the edges joining the center of each of the six faces of the cube to each of the four face corners and to the center of the cube (adding thirty additional edges, see Figure 2b), and
- the edges joining the midpoint of each of the twelve edges of the cube to the two closest cube corners, the two closest face centers and to the cube's center (adding 60 additional edges, see Figure 2c).

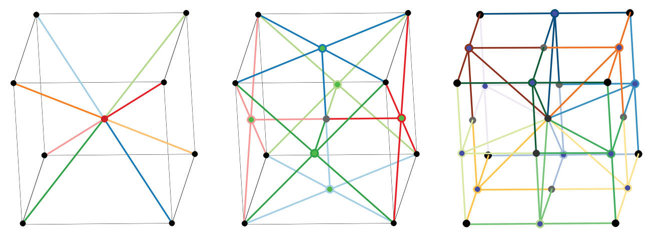


Fig. 2. The edges of a supercube $\sigma$. (a) Eight edges originating from the center of $\sigma$ (b) Six groups of five edges originating from face centers of $\sigma$ (c) Twelve groups of five edges originating from the side centers of $\sigma$. The black edges in (a) and (b) are shown only for context.

To ensure that each edge in an LEB hierarchy is only associated with a single supercube, we adopt the common convention used for octrees [24] that a supercube uses *half-open* intervals, i.e. it contains all internal edges as well as the edges on its lower boundaries but not the edges on its upper boundaries. Thus, we consider any edge of a supercube $\sigma$ whose endpoints are both on an upper boundary of $\sigma$ to belong to a neighboring supercube. Of the 98 edges introduced above, a supercube only contains the 56 edges that satisfy the half-open criteria and consists of: 8 cube diagonals, 24 face diagonals and 24 cube edges. This is illustrated (in 2D) in Figure 3 at three consecutive levels of resolution, where the solid lines are supercube edges, while the dashed lines on the upper boundaries indicate edges belonging to neighboring supercubes due to the half-open interval rule.

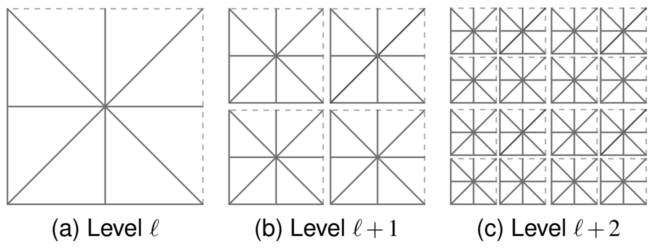


Fig. 3. Supercubes (in 2D) are structured sets of edges tiling each level of resolution within an LEB hierarchy. Three consecutive levels of resolution covering the same domain are shown, containing 1, 4 and 16 supercubes, respectively. Dashed edges are excluded due to the half-open interval rule.

The one-to-one correspondence between edges of an LEB hierarchy and the spines of diamonds provides a unique association from each diamond to a single supercube. This correspondence enables the use of supercubes to associate information with a coherent subset of the diamonds while minimizing the geometric overhead required to access this information. This is facilitated by an efficient encoding for supercubes, as described in the following Section.

## 5 Encoding Supercubes

In this Section, we discuss an encoding for supercubes within a hierarchy of diamonds $\Delta$ built on a grid with $(2^N+1)^3$ points. We assume that all vertices of the diamonds in $\Delta$ have integer coordinates in the range of $[0..2^N]$.

### 5.1 Encoding diamonds

Due to the one-to-one correspondence between diamonds and grid points, a diamond $\delta \in \Delta$ can be uniquely indexed by its central vertex. We consider the binary representation of the coordinates $v_x$, $v_y$ and $v_z$ of central vertex $\mathbf{v}_c$ as

$$\mathbf{v}_c = \begin{bmatrix} v_x & = & x_1x_2\dots x_m & d_{x_1}d_{x_2} & 00\dots0 \\ v_y & = & y_1y_2\dots y_m & d_{y_1}d_{y_2} & 00\dots0 \\ v_z & = & \underbrace{z_1z_2\dots z_m}_{\sigma} & \underbrace{d_{z_1}d_{z_2}}_{\tau} & \underbrace{00\dots0}_{\gamma} \end{bmatrix} \quad (1)$$

The *scale* $\gamma$ of a diamond $\delta$ is the minimum of the number of trailing zeros among its three coordinates. Thus, in any diamond $\delta$ at scale $\gamma$, the rightmost $\gamma$ bits in each of $v_x$, $v_y$ and $v_z$ are zero, but at least one of the bits at position $\gamma+1$ (i.e. $d_{x_2}$, $d_{y_2}$ or $d_{z_2}$ in Equation (1)) is non-zero. The *level* of a diamond $\delta$ at scale $\gamma$ is $\ell = N-\gamma$, and equals the number of $i$-diamond ancestors of $\delta$ in the DAG, i.e. its depth in the DAG modulo 3.

The two bits at positions $\gamma+1$ and $\gamma+2$ in each coordinate of $\mathbf{v}_c$ encode the diamond *type* $\tau$. Since the type $\tau$ is encoded using 2 bits in each dimension, there are $4^3 = 64$ possible values for $\tau$. However, the definition of $\gamma$ precludes the eight cases where $d_{x_2}$, $d_{y_2}$ and $d_{z_2}$ are all zero. Thus, there are 56 valid diamond types, each corresponding to a distinct supercube edge. Finally, the number $i$ of zeros at position $\gamma+1$ of $\mathbf{v}_c$ encodes the congruence *class* of $\delta$.

The final $m = N-(\gamma+2)$ bits in each of the coordinates encode the *origin* of the supercube $\sigma$ containing $\delta$, i.e., its lower-left corner. 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 3D grid that have been scaled by a factor of $2^{\gamma+2}$. Note that scale, type and supercube origin can be efficiently extracted from the binary representation of the central vertex of a diamond through bit shifting operations. Supercubes contain all unique *types* of diamonds within an LEB hierarchy, since the type $\tau$ of a diamond $\delta$ is determined by the supercube edge with which its spine coincides. Thus, a diamond's type $\tau$ encodes the scaled offset of its central vertex from the supercube origin. i.e. $\mathbf{v}_c = \sigma + (\tau \ll \gamma)$, where '$\ll$' indicates a component-wise bit shift operation (see Figure 4).

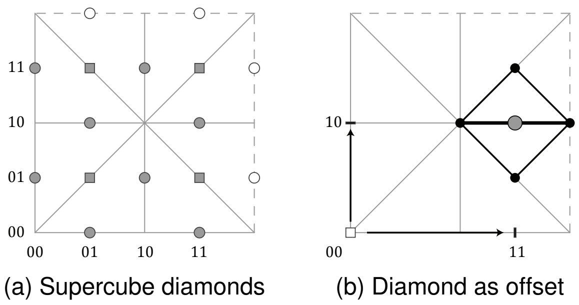


Fig. 4. (a) A diamond's spine coincides with an edge (solid lines) of a single supercube. (b) Alternatively, a diamond's central vertex lies at the midpoint of a supercube edge.

### 5.2 Internal map of a supercube

Since supercubes are intended to efficiently encode information with a subset of the diamonds of an LEB hierarchy, they require some form of bookkeeping to index the encoded diamonds.

A simple approach would be to encode this data as an array with one entry for each of the 56 possible diamonds in the supercube. The data associated with each diamond is then indexed by its type $\tau$, and unencoded diamonds are marked in place as missing. However, this can waste a considerable amount of storage for supercubes with many unencoded diamonds.

A more efficient approach for static representations is to index the encoded diamonds using a bitflag of 56 bits (i.e. 7 bytes) along with a corresponding array with storage only for the encoded diamonds. The data associated with a diamond of type $\tau$ is indexed in the array by the prefix sum of $\tau$ 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 diamonds encoded. Thus, the overhead of this representation is reduced as the average *concentration* of encoded diamonds per supercube increases.

Since this bitflag representation requires a reorganization of the array data every time a diamond is added or removed, we use the array representation during the initial generation of the data, and convert to the bitflag representation immediately thereafter.

### 5.3 Encoding collections of supercubes

We observe that within a given level of resolution $\ell$, supercubes can be uniquely indexed by their origin. However, supercubes from different levels can map to the same origin (see Figure 3).

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 $\ell$ of a supercube to be implicitly encoded within the access structure.

The encoded supercubes at a given level of resolution $\ell$ 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. An MX-Octree [24] is a hierarchical spatial access structure defined on grids where data is only associated with the leaf nodes of the complete tree, and adjacent unencoded elements can be aggregated to reduce the wasted space. The advantage of such a structure for supercubes is that the higher levels of resolution contain fewer elements, and enable quicker access. A third option is to organize the supercubes in a hash table to provide $O(1)$ access.

Thus, the data associated with a diamond $\delta$ with supercube origin $\sigma$, type $\tau$ and scale $\gamma$ can be accessed in three steps. First, the set of supercubes at level $\ell = N-(\gamma+2)$ is located. Next, the supercube $\sigma$ within this set is found. Finally, the data at location $\tau$ within $\sigma$ is found using the internal map of $\sigma$.

### 5.4 Extension to higher dimensions

A closer inspection of Equation (1) reveals that supercubes can be defined in a dimension-independent manner. Thus, the binary representation of the central vertex provides all information for retrieving geometric and hierarchical information as in the 3D case. Consider a regular grid in a $d$-dimensional cubic domain with $(2^N+1)^d$ vertices. With the exception of the $2^d$ vertices at the domain corners, each vertex corresponds to the central vertex of a $d$-dimensional diamond.

The *type* $\tau$ of a diamond $\delta$ at scale $\gamma$ is thus encoded by the two bits at positions $\gamma+1$ and $\gamma+2$ in the binary representation of the central vertex of $\delta$. There are four possible values for each of the $d$ components of $\tau$, and thus, $4^d$ possibilities for $\tau$. The values at position $\gamma+2$ are unrestricted, but the definition of diamond scale invalidates any case where all the bits at position $\gamma+1$ are zero. Since there are $2^d$ such cases, there are $(4^d-2^d)$ valid $d$-dimensional diamond *types*.

Similarly, since the *class* of a diamond is encoded within the diamond type $\tau$ by the number of zeros in the lower bit of each component, we can determine the number of distinct diamond types in each class in $d$ dimensions. Since there are $\binom{d}{i}$ combinations of $i$ zeros in the $d$ lower bits, and $2^d$ arrangements in the upper bits of $\tau$, there are $2^d\binom{d}{i}$ distinct types of $i$-diamonds in dimension $d$, for $0 \le i < d$.

## 6 Diamond-based Multiresolution Volumes

One of the primary visualization applications of diamond hierarchies has been as a multiresolution model for a volume dataset defined at the vertices of a regularly sampled scalar field, $F$. We call this model a ***D**iamond-based **M**ultiresolution **V**olume (DMV)*.

A multiresolution model [2, 15] is a static structure that consists of a coarse base mesh, a set of atomic modifications and a dependency relation on the modifications defining a partial order. In the case of a DMV, the base mesh is a coarse LEB mesh and the modifications correspond to the diamonds. Since each diamond corresponds to its central vertex, the vertices are ordered according to the dependency relation of a hierarchy of diamonds $\Delta$. Thus, the spatial decomposition and dependency relation of a DMV are obtained from the implicit relationships among the diamonds in $\Delta$, and only the modifications need to be explicitly encoded.

The minimal information encoded in a diamond $\delta$ is given by the scalar value $F(\mathbf{v}_c)$, associated with the central vertex $\mathbf{v}_c$ of $\delta$. In addition to encoding $F(\mathbf{v}_c)$, each diamond usually encodes aggregate information about the field values within the domain of $\delta$, which can be used to accelerate mesh extraction. The error $\varepsilon(\delta)$ associated with diamond $\delta$ encodes the maximum approximation error for any point within the domain of $\delta$, i.e.,

$$\varepsilon(\delta) = \underset{p\in\delta}{Max}(\varepsilon(p)), \tag{2}$$

where $\varepsilon(p) = \left|F(p)-\hat{F}(p)\right|$ 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 $\delta$. The range of values within the domain of $\delta$ is also usually maintained. This is used to cull irrelevant regions during field-based queries [30].

A *full DMV*, which we denote as $\Delta_f$, contains diamonds corresponding to all vertices of a scalar field of dimensions $(2^N+1)^3$. The base mesh of $\Delta_f$ is a single 0-diamond $\delta$ covering the entire cubic domain $\Omega$. The eight corner points of $\Omega$ (i.e. the vertices of $\delta$) are the only points within $\Delta_f$ that do not correspond to diamonds.

A full DMV $\Delta_f$ can be encoded as a three-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 [8].

However, when some of the vertices of a full DMV $\Delta_f$ are unavailable or irrelevant for an intended application, a *partial DMV*, which we denote as $\Delta_p$, can be much more efficient to encode than $\Delta_f$. The base mesh of a partial DMV $\Delta_p$ is a coarse LEB mesh consisting of diamonds from a corresponding hierarchy of diamonds $\Delta$, whose vertices are tagged with values from the scalar field $F$. The diamonds in $\Delta_p$ are a subset of the diamonds of $\Delta$ subject to the *transitive closure* constraint that if a diamond $\delta$ belongs to $\Delta_p$ then all ancestors of $\delta$ belong to $\Delta_p$ as well. Finally, the dependency relation of $\Delta_p$ is the dependency relation of $\Delta$ restricted to the diamonds in $\Delta_p$.

A straightforward representation for $\Delta_p$ is a diamond-based one, where each encoded diamond must explicitly maintain the coordinates of its central vertex in addition to its encoded data. However, due to the transitive closure constraint of the partial DMV model, the encoded diamonds exhibit both a spatial and a hierarchical coherence which can be exploited by clustering the diamonds into supercubes. Since $\Delta_p$ is static, and typically sparse with respect to a corresponding full DMV we represent the internal map within supercubes using the bitflags encoding of Section 5.2. The supercubes at each level are indexed by the coordinates of their origin.

## 7 Encoding an active front

*Selective refinement* is the general operation of extracting a variable-resolution mesh from a multiresolution model [15]. It is defined by an application-dependent predicate called the *Level of Detail (LOD) criterion*, which determines the minimum set of modifications necessary to generate a mesh of the required resolution. The LOD criterion can be based on many factors, including approximation error, field-values (e.g. isosurface extraction), location (e.g. view-dependent and region of interest queries) and perception (e.g. screen-space error). Selective refinement is performed by traversing the DAG describing the multiresolution model either in a top-down manner starting from the base mesh or incrementally from an already extracted mesh. The status of the refinement process is described by a cut of the DAG, called the *active front*, which separates the set of modifications that have been applied from those that have not.

When selective refinement is performed on a DMV, the active front describes a conforming tetrahedral mesh $\Sigma$, that we call the *current* *mesh*, which can be used to visualize an approximation of the dataset, e.g. to extract an isosurface or for direct volume rendering. In general, a diamond in a current mesh $\Sigma$ will not contain all of its tetrahedra, and thus diamonds need to track which of their tetrahedra are present in $\Sigma$ (see Figure 5a for an example in 2D, where, e.g., the blue diamonds contain only one of their two triangles). A diamond in the active front cannot be subdivided until all of its parents have been subdivided and, thus, all of its tetrahedra are present in $\Sigma$.

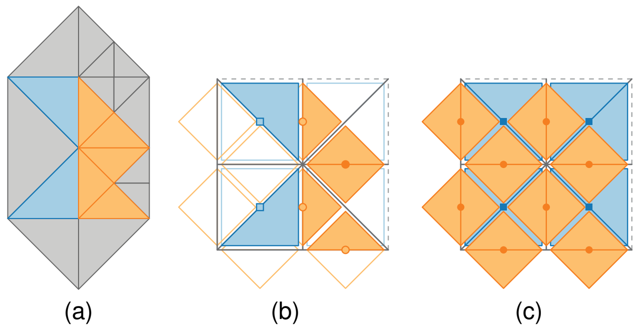


Fig. 5. (a) A portion of a *current mesh* $\Sigma$ (in 2D). (b) Highlighted triangles from $\Sigma$ map to triangles in the supercube $\sigma$. Gray triangles map to other supercubes. Diamonds in $\Sigma$ rarely contain all their triangles. (c) The set of all triangles (tetrahedra in 3D) in a supercube overlap.

We have observed [26] that in a hierarchy of diamonds, a diamond with $k$ parents has $2k$ tetrahedra, and that, upon subdivision, each parent of a diamond $\delta$ contributes a pair of tetrahedra to $\delta$. Thus, since *0-diamonds* have three parents, *1-diamonds* have two parents and *2-diamonds* have four parents, bitflags with three, two and four bits, respectively, are sufficient to track the subdivided parents of diamonds as well as their tetrahedra in $\Sigma$. Since a diamond cannot be subdivided until all of its parents have been subdivided, the former property can be used to accelerate the extraction process, while the latter property can be used to visualize the extracted mesh $\Sigma$.

A straightforward representation for encoding an active front utilizes a hash table of diamonds. Each diamond $\delta$ 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. 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.

However, there is a considerable amount of coherence among the tetrahedra in $\Sigma$ that a diamond-based representation of an active front cannot exploit. Due to the LEB subdivision rule, neighboring tetrahedra in $\Sigma$ can differ by at most one level of refinement, so the presence in $\Sigma$ of a tetrahedron from diamond $\delta$ often indicates the presence of tetrahedra from neighboring diamonds in the hierarchy. We therefore propose a supercube-based representation for active fronts extracted from a DMV. Recall that a supercube indexes 56 diamonds, of which

- 8 are 0-diamonds, each with 6 tetrahedra,
- 24 are 1-diamonds, each with 4 tetrahedra and
- 24 are 2-diamonds, each with 8 tetrahedra.

Thus, each supercube indexes up to 336 tetrahedra. Note, however, that these tetrahedra overlap (see Figure 5c), but tetrahedra in a conforming mesh cannot overlap. Due to the containment relation among the tetrahedra in the hierarchy, the presence of a tetrahedron $\mathbf{t}$ in $\Sigma$ precludes the presence of its parent tetrahedron and both of its children tetrahedra from $\Sigma$. Thus, in practice, a supercube in $\Sigma$ contains significantly fewer than the 336 possible tetrahedra (see Figure 5b).

Since tetrahedra contribute to diamonds of $\Sigma$ in pairs, we can track the tetrahedra due to a single supercube in the active front $\Sigma$ using (336/2) bits = 21 bytes. Our proposed supercube-based representation for an active front therefore requires 27 bytes of overhead per supercube: 6 bytes to index its origin and 21 bytes of bookkeeping.

## 8 Results

In this Section, we evaluate the effectiveness of supercubes as a partial DMV representation and as an active front representation across a testbed of volume datasets of resolution up to $512^3$. All experiments were run on a 2 GHz Intel Core 2 Duo laptop with 4 GB of RAM.

We first consider when it is appropriate to represent a dataset using a partial diamond hierarchy. This is measured in terms of the *density* of the dataset, i.e. the percentage of samples from a full hierarchy that are retained in the partial representation. Next, we analyze when supercube-based representations of a partial hierarchy 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.

Clustered representations typically incur some computational overhead related to the extra level of indirection required to index the data. We thus compare the runtime performance of the DMV representations during selective refinement queries. Since the efficiency of a selective refinement query depends on the representation of its associated active front, we conclude by comparing the supercube-based and diamond-based active front representations of Section 7 in terms of performance and storage costs.

We begin by introducing some notation. Let $\Delta_f$ denote the full DMV, containing $n_f = (2^N+1)^3$ diamonds and let $\Delta_p$ denote the desired partial DMV, containing $n_p$ diamonds. $\Delta_p$ can be encoded using a diamond-based partial DMV $\Delta_d$ or a supercube-based representation $\Delta_s$, whose $n_p$ diamonds are clustered into $n_s$ supercubes. Finally, let $b_\delta$ 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 $b_\sigma$ 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 1 with respect to an ideal representation $\Delta_p$, 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. $\Delta_f$ 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, $\Delta_d$ encodes only the $n_p$ diamonds but must also explicitly encode the spatial coordinates of each diamond. Finally, the overhead in $\Delta_s$ can be attributed entirely to the $n_s$ supercubes.

| Representation | Cost | Overhead |
|---|---|---|
| $\Delta_p$ | $n_p * b_\delta$ | 0 |
| $\Delta_f$ | $n_f * b_\delta$ | $(n_f - n_p) * b_\delta$ |
| $\Delta_d$ | $n_p * b_\delta + n_p * b_v$ | $n_p * b_v$ |
| $\Delta_s$ | $n_p * b_\delta + n_s * b_\sigma$ | $n_s * b_\sigma$ |

Table 1. Storage requirements and overhead, in bytes, for the various DMV representations. Overhead is relative to $\Delta_p$.

Using this notation, we define the *density* $\mathscr{D} = n_p/n_f$ of the dataset as the ratio of retained diamonds in $\Delta_p$ compared to $\Delta_f$. Also, we define the *concentration* $\mathscr{C} = n_p/n_s$ of the dataset as the average number of diamonds per supercube. We note that $\mathscr{C} \in [1, 56]$ since we only encode supercubes that contain at least one diamond.

By rearranging the equations in Table 1 and substituting terms for $\mathscr{D}$ and $\mathscr{C}$, we can compare the representations. $\Delta_s$ is more compact than $\Delta_f$ when

$$\mathscr{D} < \frac{b_\delta}{b_\delta + (b_\sigma/\mathscr{C})},$$

$\Delta_d$ is more compact than $\Delta_f$ when

$$\mathscr{D} < \frac{b_\delta}{b_\delta + b_v}$$

and $\Delta_s$ is more compact than $\Delta_d$ when $\mathscr{C} > b_\sigma / b_v$. 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 $\Delta_p$ (third column of Table 1). While $\Delta_d$ has a constant overhead of $b_v$ bytes per diamond, the overhead in $\Delta_s$ is related to $\mathscr{C}$ as $(b_\sigma / \mathscr{C})$ bytes per diamond.

As a concrete example, let the size of each refinement be $b_\delta = 4$ bytes as in [8]. Further, assume vertices are encoded in 6 bytes as three `unsigned shorts`, then $b_v = 6$ bytes. Finally, let $b_\sigma = 17$ 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, $\Delta_s$ is more compact than $\Delta_f$ and $\Delta_d$, respectively, when $\mathscr{D} < \frac{4}{(4+17/\mathscr{C})}$, and when $\mathscr{C} > 17/6$. The gray curves in Figures 6 and 7 separate the half-spaces in which $\Delta_s$ is more compact than $\Delta_f$ by the constant to its right. For example, when $\mathscr{C} = 17$ and $\mathscr{D} = .2$, $\Delta_f$ requires four times as much space to encode as $\Delta_s$.

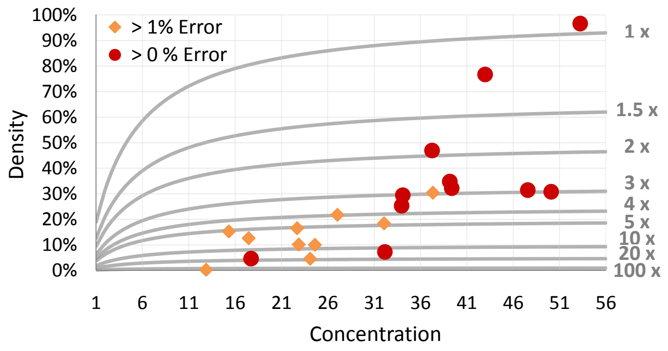


Fig. 6. Density and concentration of partial DMV datasets of Table 2 extracted from a complete DMV with uniform error greater than 0% (red) and 1% (orange). Gray curves indicate the factor by which a supercube-based partial DMV $\Delta_s$ is more compact than a full DMV $\Delta_f$.

As a first application, consider a partial DMV $\Delta_p$ generated from a full DMV $\Delta_f$ by retaining all diamonds whose error is greater than a given threshold $\varepsilon$. Since $\Delta_p$ corresponds to a partial hierarchy of diamonds, it must also retain all ancestors of the retained diamonds. This ensures the transitive closure of the diamond dependency relation (as described in Section 6). When $\varepsilon = 0$, this generates a lossless encoding of $\Delta_f$, i.e. $\Delta_f$ can be reconstructed from $\Delta_p$ without any error.

Table 2 lists the number of elements, and storage costs in a zero-error partial DMV $\Delta_p$ for various datasets as well as their density $\mathscr{D}$ and concentration $\mathscr{C}$. These datasets are plotted on Figure 6 for $\varepsilon = 0\%$ and $\varepsilon = 1\%$. We observe that some datasets, such as `Fuel` and `Aneurism` are extremely sparse, and achieve a 12.2 times and 17.3 times reduction in storage requirements, respectively, compared to $\Delta_f$. 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 $\Delta_f$. However, even for these datasets, the size of $\Delta_s$ is close to that of $\Delta_f$ (requiring 4% more and 19% less space, respectively), whereas $\Delta_d$ 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 $\Delta_s$ compared to $\Delta_f$ (see last column in Table 2).

Since $b_v$ (6 bytes) is 1.5 times as large as $b_\delta$ (4 bytes), the overhead associated with $\Delta_d$ compared to the ideal representation $\Delta_p$ is 150%. In contrast, the overhead of $\Delta_s$ is related to the concentration of the supercube clustering, and averages around 12% across all datasets. Thus, the 2.25 times savings achieved by $\Delta_s$ compared to $\Delta_d$ is entirely due to the difference in storage overhead.

Partial hierarchies 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.

| Dataset | N | $n_f$ | $n_p$ | $n_s$ | $\mathscr{D}$ | $\mathscr{C}$ | $\Delta_f$ | $\Delta_d$ | $\Delta_s$ | $\Delta_f/\Delta_s$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Fuel | 6 | 275 K | 19.9 K | 620 | .07 | 32.2 | 1.05 | .19 | .09 | 12.2 x |
| Neghip | 6 | 275 K | 129 K | 3.46 K | .47 | 37.2 | 1.05 | 1.23 | .55 | 1.91 x |
| Plasma | 6 | 275 K | 265 K | 4.98 K | .97 | 53.2 | 1.05 | 2.53 | 1.09 | .96 x |
| Hydrogen | 7 | 2.15 M | 545 K | 16.0 K | .25 | 34.0 | 8.19 | 5.19 | 2.34 | 3.50 x |
| Buckyball | 7 | 2.15 M | 1.65 M | 38.3 K | .77 | 43.0 | 8.19 | 15.7 | 6.90 | 1.19 x |
| Aneurism | 8 | 17.0 M | 791 K | 44.6 K | .05 | 17.7 | 64.8 | 7.54 | 3.74 | 17.3 x |
| Tooth | 8 | 17.0 M | 5.23 M | 104 K | .31 | 50.1 | 64.8 | 49.9 | 21.7 | 2.99 x |
| Engine | 8 | 17.0 M | 5.34 M | 112 K | .31 | 47.6 | 64.8 | 50.9 | 22.2 | 2.92 x |
| Head | 8 | 17.0 M | 5.47 M | 139 K | .32 | 39.4 | 64.8 | 52.1 | 23,1 | 2.80 x |
| Bonsai | 8 | 17.0 M | 5.00 M | 147 K | .29 | 34.1 | 64.8 | 47.7 | 21.5 | 3.02 x |
| Foot | 8 | 17.0 M | 5.90 M | 151 K | .35 | 39.2 | 64.8 | 56.3 | 25.0 | 2.59 x |

Table 2. DMVs generated based on uniform field error with $\varepsilon = 0$. File sizes for $\Delta_f$, $\Delta_d$ and $\Delta_s$ are listed in MB (1 MB = $1024^2$ B). All datasets are plotted on Figure 6 (red circles).

We can thus generate an isovalue-based partial DMV $\Delta_p$ from $\Delta_f$, where all diamonds whose range intersects the predetermined isovalue(s) are retained, while those not intersecting the isovalue(s) are only retained if they are ancestors of the required diamonds. $\Delta_p$ can then be queried using selective refinement to extract adaptive tetrahedral meshes and isosurfaces. This model thus trades fidelity in regions away from the desired isosurface for storage and extraction efficiency of the desired isosurface(s).

Table 3 lists the number of elements and the storage requirements for the three DMV representations for each isovalue-based dataset (the values of $N$ and $n_f$ can be found in Table 2). The density and concentration of these datasets are plotted in Figure 7. We observe that these extracted partial DMVs are indeed sparse with respect to $\Delta_f$, 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 DMVs of these datasets require an average of 25 times less storage than their corresponding full DMVs. In fact, the largest dataset $\texttt{Xmas}_{\{868\}}$ (orange square in Figure 7) requires only 1.3% of the samples of $\Delta_f$ and is over 65 times more compact.

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

We note that, when more than one isovalue is desired (as in the set of `Engine` datasets on the bottom of Table 3, and the corresponding colored rhombuses in Figure 7), there is also a significant amount of coherence among the active cells of distant isovalues (e.g. 58 and 170). Thus, the supercube-based representation for $\texttt{Engine}_{\{58,100\}}$ requires only 15% more storage space than either of the individual datasets $\texttt{Engine}_{\{58\}}$ or $\texttt{Engine}_{\{100\}}$, and has a higher concentration than either of them. This advantage is increased in $\texttt{Engine}_{\{58,100,170\}}$ as the samples from a third isovalue are added, where the density only increases by 3% and the supercube concentration increases by .8.

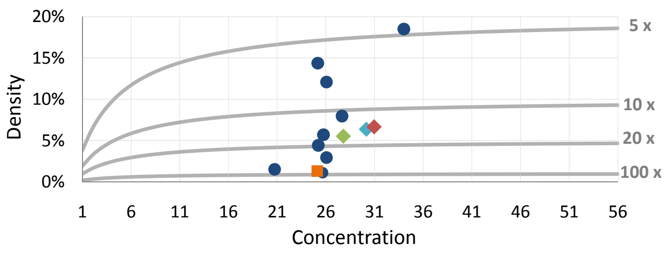


Fig. 7. Density and concentration of partial DMV datasets of Table 3 containing all diamonds intersected by the specified isovalue(s).

We now compare the runtime performance of the three DMV representations, $\Delta_f$, $\Delta_d$ and $\Delta_s$, 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

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

Table 3. DMVs generated based on specific isovalue(s). File sizes are listed in MB (1 MB = $1024^2$ B). All datasets are plotted on Figure 7.

| | Diamond-Based ($\Sigma_d$) | | | Supercube-Based ($\Sigma_s$) | | |
|---|---|---|---|---|---|---|
| | Min | Max | Average | Min | Max | Average |
| $\Delta_f$ | 252 K/s | 343 K/s | 317 K/s | 298 K/s | 354 K/s | 324 K/s |
| $\Delta_d$ | 223 K/s | 333 K/s | 301 K/s | 263 K/s | 340 K/s | 306 K/s |
| $\Delta_s$ | 237 K/s | 327 K/s | 296 K/s | 276 K/s | 331 K/s | 300 K/s |

Table 4. Selective refinement performance of the three DMV representations using a diamond-based active front representation $\Sigma_d$ and a supercube-based active front representation $\Sigma_s$, in terms of the minimum, maximum and average number of diamonds visited per second.

the two active front representations introduced in Section 7. Recall that the active front of a selective refinement query on a diamond hierarchy corresponds to a tetrahedral mesh called the *current mesh* $\Sigma$. This mesh can be represented using a diamond-based representation, which we denote as $\Sigma_d$, or through a supercube-based representation which we denote as $\Sigma_s$.

Since selective refinement queries depend on the specific LOD 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 4, we present the aggregate results over our testbed of 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 $\varepsilon$ and containing a particular isovalue $\kappa$. As a partial DMV for this query, we use the datasets generated in Table 3.

For these experiments, we implemented $\Delta_d$ using a hash table from the central vertex of each diamond in $\Delta_d$ 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 $\Delta_d$ 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 $\Delta_d$ requires an average memory overhead of 36% compared to the values listed in Table 3. Similarly, for $\Delta_s$, 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 5.3). We found the load factor to average 75% (with a standard devaiation of 11%) across the datasets, thus, requiring an average memory overhead of 33% compared to the values listed in Table 3.

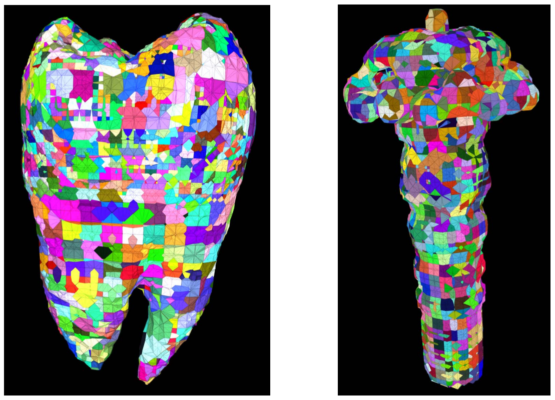
We evaluate the performance of the DMV representations (i.e. the rows of Table 4) by comparing the average number of diamonds processed per second. Recall that due to the query type and the transitive closure of $\Delta_d$ and $\Delta_s$, all three representations process the same set of diamonds and yield the same result. We first observe that all three representations yield similar performance results of about 300,000 diamonds per second. $\Delta_f$ is the fastest DMV representation, since it can directly access its diamonds using the array location of their central vertices. $\Delta_d$ is approximately 5-6% slower than $\Delta_f$, due to its use of indirect hashing, while $\Delta_s$ is around 7.5% slower than $\Delta_f$. Thus, despite its required extra processing, such as extracting the supercube origin and diamond type and the prefix sum calculation, supercube-based $\Delta_s$'s performance is within 2% that of diamond-based $\Delta_d$.

Next, we evaluate the relative performance of the two active front representations $\Sigma_s$ and $\Sigma_d$ by comparing columns 4 and 7 (AVERAGE) of Table 4. Thus, the supercube-based active front representation $\Sigma_s$ is, on average, 1-2% more efficient than the diamond-based active front representation $\Sigma_d$. Although this is not a significant difference, we note that the addition (removal) of any diamond to (from) $\Sigma_d$ incurs a memory allocation (deallocation), whereas, due to the supercube clustering, such allocations (deallocations) are rarer for $\Sigma_s$.

Finally, we evaluate the sizes of the two active front representations. Recall that the supercube-based active front representation $\Sigma_s$ requires 27 bytes overhead per supercube. Over the entire test, $\Sigma_s$ averaged 26.5 tetrahedra per supercube (with a standard deviation of 3.1). Thus, the supercube-based active front $\Sigma_s$ incurs an overhead of around 1 byte per tetrahedron in the active front. In contrast, the diamond-based active front representation $\Sigma_d$ requires 7 bytes overhead per diamond. Over the entire test, $\Sigma_d$ 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 $\Sigma_s$ and $\Sigma_d$ using hash tables, (analogously to our indexing of the partial DMVs above). Across all datasets, we achieved an average load factor of 74% for $\Sigma_d$ (with a standard deviation of 15%), and thus the hash-indexed $\Sigma_d$ incured a memory overhead of around 36%. The average load factor for $\Sigma_s$ was 72% (with a standard deviation of 11%), requiring an average overhead of 39%. Thus, a supercube-based active front representation $\Sigma_s$ can be used to extract an equivalent mesh from a DMV as a diamond-based representation $\Sigma_d$ in slightly less time and using less than half the storage.

Figure 8 illustrates the clustering of tetrahedra in a supercube-based active front by the color of their isosurface triangles.



(a) Tooth, $\varepsilon > 10\%$ (b) Fuel, $\varepsilon > .03\%$

Fig. 8. Isosurfaces extracted from DMV models. Triangles are colored by their embedding supercube. (a) Tooth dataset with uniform error $\varepsilon > 10\%$. A supercube-based active front representation $\Sigma_s$ has an average of 26.9 tetrahedra per supercube yielding an average overhead of 1 byte per tetrahedron. The diamond-based active front $\Sigma_d$ has an average of 2.9 tetrahedra per diamond yielding an overhead of 2.4 bytes per tetrahedron. (b) Fuel dataset with uniform error $\varepsilon > .03\%$. $\Sigma_s$ has an average of 25.6 tetrahedra per supercube yielding an average overhead of 1.05 bytes per tetrahedron. $\Sigma_d$ has an average of 3.3 tetrahedra per diamond yielding an overhead of 2.12 bytes per tetrahedron.

## 9 CONCLUDING REMARKS

We have proposed the *supercube* as a high-level primitive for compactly encoding nested tetrahedral meshes generated through longest edge bisection. Supercubes represent the basic unit of symmetry within the hierarchy of diamonds model, and encode all distinct types of diamonds within the hierarchy (up to scaling factors). Supercube clustering provides an efficient means of associating information with a subset of the diamonds within an LEB hierarchy by exploiting the spatial and hierarchical coherence within the dataset.

We have demonstrated that several visualization datasets 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 DMV 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 (see the end of Section 8), our partial DMV representations can be inflated by as much as 40%. This can reduce the benefits of a supercube-based DMV $\Delta_s$ to a full DMV $\Delta_f$, when their relative differences are less pronounced. However, when the relatives sizes are more significant, a hash-indexed $\Delta_s$ still provides a significant advantage over $\Delta_f$. Additionally, we have demonstrated that hash-indexed DMVs and active front representations based on diamonds suffer from similar or worse overhead than their supercube-based counterparts.

In future work, we intend to focus on the indexing structure for diamonds and supercubes in a partial hierarchy. As mentioned in Section 5.3, since the set of supercubes at a given level of resolution tile the plane, an MX-octree is a promising representation for the spatial access structure. Specifically, a pointerless MX-octree [24] should achieve the storage goals of Tables 2 and 3. Alternatively, since $\Delta_d$ and $\Delta_s$ contain static spatial data, a perfect spatial hash [14] can be generated to yield significantly lower overhead than a standard hash table.

We are also currently looking into modifications of the supercube representation for higher dimensional datasets such as time-varying volume data where the internal map of each supercube must reflect the increased number of diamonds.

## Acknowledgements

We would like to thank the anonymous reviewers for their many helpful suggestions. This work has been partially supported by the National Science Foundation under grant CCF-0541032 and by the MIUR-FIRB project SHALOM under contract number RBIN-04HWR8. The Buckyball model is courtesy of AVS Inc. The Plasma model is courtesy of Visual Computing Group, National Research Council, Pisa, Italy. All other datasets are courtesy of volvis.org.

## References

[1] 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.

[2] L. De Floriani and P. Magillo. Multi-resolution 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.

[3] 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, *Proc. IEEE Visualization*, pages 81–88, Phoenix, AZ, October 1997. IEEE Computer Society.

[4] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings SIGGRAPH'00 Conference*, pages 249–254, New Orleans, LA, July 2000. ACM Press.

[5] T. Gerstner. Multi-resolution visualization and compression of global topographic data. *GeoInformatica*, 7(1):7–32, 2003.

[6] T. Gerstner and R. Pajarola. Topology-preserving and controlled topology simplifying multi-resolution isosurface extraction. In *Proceedings IEEE Visualization 2000*, pages 259–266, 2000.

[7] T. Gerstner and M. Rumpf. Multiresolutional parallel isosurface extraction based on tetrahedral bisection. In *Proceedings Symposium on Volume Visualization*, pages 267–278. ACM Press, 1999.

[8] 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, 2002.

[9] B. Gregorski, J. Senecal, M. Duchaineau, and K. Joy. Adaptive extraction of time-varying isosurfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):683–694, 2004.

[10] D. J. Hebert. Symbolic local refinement of tetrahedral grids. *Journal of Symbolic Computation*, 17(5):457–472, May 1994.

[11] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, 2002.

[12] 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.

[13] 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.

[14] S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics (TOG)*, 25(3):579–588, 2006.

[15] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, San Francisco, 2002.

[16] S. Marchesin, J. Dischler, and C. Mongenet. 3D ROAM for scalable volume visualization. *IEEE Symposium on Volume Visualization and Graphics*, pages 79–86, 2004.

[17] J. M. Maubach. Local bisection refinement for *n*-simplicial grids generated by reflection. *SIAM Journal on Scientific Computing*, 16(1):210–227, January 1995.

[18] V. Mello, L. Velho, and G. Taubin. Estimating the in/out function of a surface represented by points. *Symposium on Solid Modeling and Applications 2003*, pages 108–114, 2003.

[19] V. Pascucci. Slow Growing Subdivisions (SGS) in any dimension: towards removing the curse of dimensionality. *Computer Graphics Forum*, 21(3):451–460, 2002.

[20] V. Pascucci. Isosurface computation made simple: Hardware acceleration, adaptive refinement and tetrahedral stripping. *Eurographics/IEEE TVCG Symposium on Visualization (VisSym)*, pages 293–300, 2004.

[21] R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. In *ACM Computer Graphics, (Proc. SIGGRAPH 2001)*, pages 47–56, Los Angeles, CA, August 2001. ACM Press.

[22] M. Rivara and C. Levin. A 3D refinement algorithm for adaptive and multigrid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.

[23] T. Roxborough and G. Nielson. Tetrahedron-based, least-squares, progressive volume models with application to freehand ultrasound data. In *Proceedings IEEE Visualization 2000*, pages 93–100. IEEE Computer Society, October 2000.

[24] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Elsevier, 2006.

[25] S. Takahashi, Y. Takeshima, G. Nielson, and I. Fujishiro. Topological volume skeletonization using adaptive tetrahedralization. *Geometric Modeling and Processing, 2004. Proceedings*, pages 227–236, 2004.

[26] K. Weiss and L. De Floriani. Multiresolution Interval Volume Meshes. In H.-C. Hege, D. Laidlaw, R. Pajarola, and O. Staadt, editors, *IEEE/EG Symposium on Volume and Point-Based Graphics*, pages 65–72, Los Angeles, California, USA, 2008. Eurographics Association.

[27] K. Weiss and L. De Floriani. Sparse Terrain Pyramids. In *Proc. of the 16th ACM SIGSPATIAL Int. Conference on Advances in Geographic Information Systems*, pages 115–124, New York, NY, USA, 2008. ACM.

[28] K. Weiss and L. De Floriani. Diamond Hierarchies of Arbitrary Dimension. *Computer Graphics Forum*, 28(5):1289–1300, 2009.

[29] 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.

[30] J. Wilhelms and A. Van Gelder. Topological considerations in isosurface generation extended abstract. *Proceedings of the 1990 workshop on Volume visualization*, pages 79–86, 1990.

[31] Y. Zhou, B. Chen, and A. Kaufman. Multi-resolution tetrahedral framework for visualizing regular volume data. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization '97*, pages 135–142, Phoenix, AZ, October 1997. IEEE Computer Society.