Constant-Time Navigation in Four-Dimensional Nested Simplicial Meshes*

Michael Lee[1], Leila De Floriani[1,2], and Hanan Samet[1]

[1]Computer Science Department
University of Maryland, College Park, Maryland 20742 (USA)

[2]Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Via Dodecaneso 35, 16146 Genova - Italy

{magus, deflo, hjs}@umiacs.umd.edu

Abstract

We consider a recursive decomposition of a four-dimensional hypercube into a hierarchy of nested 4-dimensional simplexes, that we call pentatopes*. The paper presents an algorithm for finding the neighbors of a pentatope along its five tetrahedral faces in constant time. To this aim, we develop a labeling technique for nested pentatopes that enables their identification by using location codes. The constant-time behavior is achieved through bit manipulation operations, thus avoiding traversing the simplicial hierarchy via pointer following. We discuss an application of this representation to multi-resolution representations of four-dimensional scalar fields. Extracting adaptive continuous approximations of the scalar field from such a model requires generating conforming meshes, i.e., meshes in which the pentatopes match along their tetrahedral faces. Our neighbor finding algorithm enables computing face-adjacent pentatopes efficiently.*

1. Introduction

Time-varying volumetric data sets are sets of points in the three-dimensional Euclidean space describing a scalar field (e.g., pressure, temperature, strength of an electric or a magnetic field) at different instances of time. Time-varying scalar fields arise in engineering, biomedical and other scientific applications, which produce very large data sets by numerical simulations or acquisition. Time-varying fields are often treated as four-dimensional scalar fields by considering time as the fourth dimension [13, 28]. The field can be analyzed by extracting isosurfaces, consisting of tetrahedral cells, which can be visualized at different instants of time. A four-dimensional scalar field can be modeled by decomposing its domain either as a hypercubic grid or as a simplicial mesh with vertices at the data points, obtained by triangulating the former one. Isosurface extraction algorithms, however, are much simpler on simplicial meshes.

Usually, time-varying data sets are very large, and thus, a multi-resolution approach can be suitable for working with them. *Multi-resolution models*, also called *Level-OF-Detail (LOD) models*, have been widely used for describing free-form surfaces, two-dimensional height fields and three-dimensional volume data sets (see [2, 4] for surveys). They implicitly encode a *virtually continuous* set of simplified approximations at different LODs. Adaptive representations are extracted by varying the resolution (i.e., the density of the cells) in different parts of the field domain, or in the proximity of interesting field values. For both two-dimensional (terrain data) and three-dimensional (volume data) scalar fields, defined at regularly-spaced data points, nested meshes, generated by a recursive bisection process of a triangle or of a tetrahedron along its longest edge, have been used because of their capability of producing highly adaptive representations.

In this work, we consider a recursive decomposition of a hypercube into a hierarchy of nested 4-dimensional simplexes, that we call *pentatopes*. We call the resulting hierarchy a *hierarchy of pentatopes*. A hierarchy of pentatopes can be used as the domain decomposition for a four-dimensional scalar field. We address the problem of computing face-neighbors of a pentatope. We propose a neighbor finding algorithm which makes use of a pointer-less representation of a nested simplicial mesh. In such representation, pentatopes are implicitly described as strings of bits, called *location codes*, corresponding to a path from the root of the hierarchy representing the nested simplicial mesh. The algorithm performs bitwise manipulation of the location code of the pentatopes to find neighbors in worst-case

* This work was supported in part by the National Science Foundation under grants EIA-99-00268, IIS-00-86162, and EIA-00-91474, and by Columbia Union College, Takoma Park, Maryland.

constant time. Such technique extends the one developed in [14] for nested tetrahedral meshes.

The major contributions of this paper are:

- An algorithm for neighbor finding on a pointer-less representation of a hierarchy of pentatopes.
- A version of such algorithm which works in *worst-case constant* time.

An application of the proposed neighbor finding algorithm is in extracting adaptive representations of a time-varying scalar field from a multi-resolution model of the field based on a hierarchy of pentatopes. Adaptive representations can be extracted by a top-down traversal of the hierarchy which recursively computes the isosurfaces intersecting each pentatope [27]. This would produce a tetrahedral mesh from which isosurfaces for different time values can be generated. If the nested mesh elements do not properly meet at faces, then the isosurfaces can present cracks. Mesh consistency must be maintained by splitting clusters of face-adjacent pentatopes, at the same time. Such clusters can be efficiently computed through the neighbor finding algorithm described here.

The rest of this paper is organized as follows. Section 2 reviews some related work. Section 3 introduces background notions on nested simplicial meshes. Section 4 describes nested 4-dimensional simplicial meshes, which we call *Hierarchies of Pentatopes (HPs)*. Section 5 introduces a labeling scheme for an HP which enables us to define location codes for pentatopes. Section 6 discusses the problem of generating nested conforming meshes, and states the neighbor finding problem. Section 7 presents a technique for neighbor finding based on location codes. Section 8 describes a worst-case constant time implementation of the neighbor finding algorithm. Concluding remarks are drawn in Section 9.

2. Related work

In this section, we review related work on multi-resolution models based on nested tetrahedral meshes for three-dimensional scalar fields, on neighbor finding approaches, and on modeling techniques for four-dimensional scalar fields.

Multi-resolution models based on nested meshes. Nested tetrahedral meshes have been studied in finite element analysis and in computer graphics for describing scalar fields when the field values are given at the vertices of a regular square grid in 3D space. Examples are tetrahedral meshes generated by the so-called *red/green tetrahedron refinement* technique (see, for instance, [10]), or nested meshes formed by tetrahedral and octahedral elements [9].

Nested meshes generated by recursively bisecting tetrahedra along their longest edge have been introduced for domain decomposition in finite element analysis [11, 16, 20], and they have been applied in scientific visualization [3, 7, 8, 17, 19, 22, 29]. A generalization to arbitrary dimensions is presented in [18]. Nested tetrahedral meshes are either described through hierarchies of tetrahedra, thus representing the containment relation induced by the subdivision process [7, 29], or as a directed acyclic graph in which the nodes represent clusters of tetrahedra which need to be split at the same time and the arcs define the parent/child relation among such clusters [8].

When an adaptive mesh is extracted from a nested representation, the field associated with the extracted mesh (and, thus, the resulting isosurfaces) may present discontinuities in areas of transition. Continuity can be ensured through error saturation [7, 29], thereby implicitly forcing all parents to be split before their descendants, or through neighbor finding [14]. In [3], we have shown that the latter approach exhibits the same performances in terms of computation times as approaches based on error saturation, while generating fewer tetrahedra for the same value of the approximation error.

Neighbor finding algorithms. The earliest results on neighbor finding deal with algorithms to compute adjacent blocks (i.e., neighbors) of greater or equal size in region quadtrees [23] and region octrees [24], described through pointer-based data structures. Subsequently, pointer-less methods for representing such structures have been developed, in which each block B in the decomposition is represented by a *location code*, i.e., by encoding the path from the root of the tree to B as a bit string (e.g., [6]). Pointer-less representations enable finding neighbors of equal size in constant time through bit manipulations involving arithmetic and logical operations.

Worst-case constant time neighbor finding algorithms have been developed for region quadtrees [26], for triangle quadtrees [15], i.e., hierarchical meshes of equilateral triangles generated by splitting a triangle into four, and for hierarchies of right triangles, in which a triangle is split into two triangles [5].

The technique proposed by Hebert [11] computes parents, children, and neighbors in a nested tetrahedral mesh in a symbolic way. Finding neighbors still takes time proportional to the depth in the hierarchy. A neighbor finding technique for nested tetrahedral meshes has been proposed in [14], which computes the face-neighbors of a tetrahedron in worst-case constant time.

Modeling four-dimensional data. The problem of modeling and encoding time-varying scalar fields have been recently considered by some authors [1, 12, 13, 21, 28]. In [12], a loss-less single resolution compression technique is proposed for encoding very large and regularly-sampled 4D data. In [13], the problem of tracking and visualizing local features from a time-varying volumetric data set is considered, based on extracting time-varying isosurfaces and interval volumes using isosurfaces in higher dimensions.

Algorithms for isosurface extraction from a 4D scalar field have also been developed. Extensions of the marching cube algorithm to 4D have been proposed [1, 21], which differ in the number of cases counted for the 4-cube, that is, 272 [21], and 222 [1]. Weigle and Banks [27] have proposed a recursive algorithm for isosurface extraction from simplicial meshes, counting 5 possible different cases for a 4-simplex. The algorithm has been applied in [28] for visualizing unsteady 3D scalar fields.

3. Nested simplicial meshes

A *k-dimensional simplex*, or *k-simplex* σ, for brevity, in $\mathbb{E}^d$ is the locus of the points in $\mathbb{E}^d$ that can be expressed as the convex combination of $k+1$ affinely independent points V_σ. Any q-simplex, with $q < k$, which is generated by a subset of points of V_σ, is called a *face* of σ.

A collection Σ of k-simplexes in $\mathbb{E}^d$, $k = 1, 2, ..., d$, is a d-dimensional *simplicial mesh* if and only if all the faces of simplexes in Σ belong to Σ as well, the interiors of any pair of d-dimensional simplexes belonging to Σ are disjoint, and any k-simplex of Σ, with $k < d$, bounds at least one d-simplex of Σ.

A simplicial mesh Σ is called *conforming* if and only if, for each pair of d-simplexes σ_1 and σ_2 in Σ, the intersection of the boundaries of σ_1 and σ_2 is either empty, or consists of a k-face belonging to the boundary of both σ_1 and σ_2, for some $k < d$. Note that a conforming mesh is the same as a regular simplicial complex in algebraic topology. Moreover, we are interested in simplicial meshes with a manifold domain. Intuitively, a d-dimensional *manifold (with boundary)* M is a subset of the d-dimensional Euclidean space such that each point P of M has a neighborhood homeomorphic (i.e., topologically equivalent) to an open ball, or to an open ball intersected with a plane (if P is on the boundary of M). The use of conforming meshes as decompositions of the domain of a scalar field, which is sampled at a finite set of points on a manifold, provides a way of ensuring at least C^0 continuity for the resulting approximation, without requiring to modify the values of the field at the faces where discontinuities may arise.

A mesh in which the simplexes are defined by the uniform subdivision of a d-simplex into scaled copies of it is called a *nested mesh*. A nested mesh is not necessarily conforming. Note that, in this paper, we consider only nested meshes in which the vertices are on a d-dimensional hypercubic lattice.

A special class of nested simplicial meshes are those generated through d-simplex bisection. The *bisection rule* for a d-simplex σ in a d-dimensional mesh Σ consists of replacing σ with the two d-simplexes obtained by splitting σ at the middle point v_m of its longest edge e and by the hyper-plane defined by v_m and the vertices of σ which are not endpoints of e. This rule is applied recursively to an initial decomposition of the d-dimensional hyper-cubic domain into $d!$ d-simplexes and generates a nested mesh, that we call a *hierarchy of d-simplexes*. The containment relation among the d-simplexes induces a natural tree representation, in which the nodes are d-simplexes and the two children of a d-simplex σ are the two d-simplexes generated by bisecting σ.

4. A hierarchy of pentatopes (HP)

In this section, we consider an instance of a hierarchy of d-simplexes, formed by 4-simplexes, that we call *pentatopes*. We call the resulting hierarchy, a *Hierarchy of Pentatopes (HP)*.

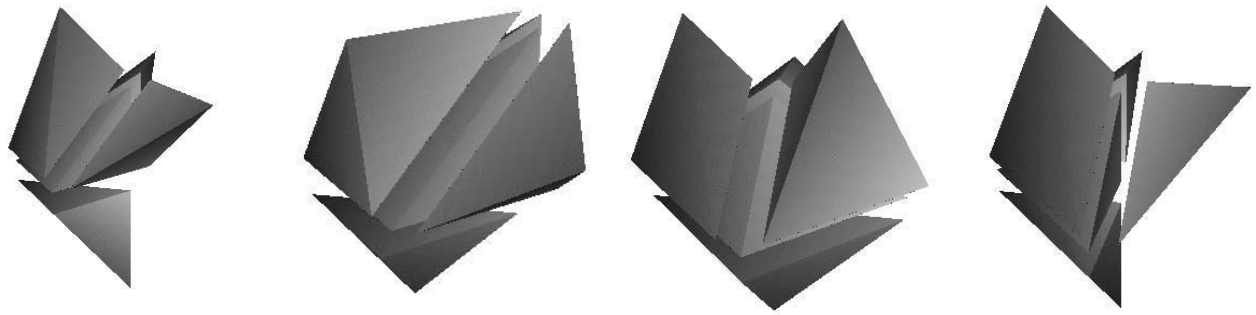
The general decomposition strategy starts with a hypercube, which is subdivided into 24 pentatopes, all sharing an edge which connects a pair of vertices of the hypercube which do not belong to the same face (cube, square or edge) in the hypercube. A pentatope is bounded by 5 0-simplexes (vertices), 10 1-simplexes (edges), 10 2-simplexes (triangles), and 5 3-simplexes (tetrahedra). Two of the five tetrahedral faces of each pentatope σ are contained within one of the eight cubic faces of the hypercube, while each of the remaining three faces is shared by two pentatopes in the subdivision of the hypercube.

The pentatopes at level 0 in an HP result from the initial subdivision of the hypercube. The pentatopes at level $i+1$ are generated by bisecting pentatopes at level i. We need four bisection steps in order to create a pentatope at level i ($i > 3$) which is a factor of two smaller in all directions than its ancestor at level $i-4$. Pentatopes at level i are congruent to their ancestors at level $i-4$ modulus reflections. In other words, the bisection rule generates four classes of congruent pentatopes. This result has been proven by Maubach [16] in the general d-dimensional case: the amount of congruency classes generated through bisection is equal to d, independently of the level of refinement.

For clarity, we describe and classify the four simplicial shapes generated by the bisection process (we denote with h the initial hypercube) as follows:

- *h-pentatope*: pentatope initially generated at level 0 by the subdivision of hypercube h.
- *c-pentatope*: pentatope initially generated at level 1 by splitting an h-pentatope along its longest edge, which is the diagonal of hypercube h.
- *s-pentatope*: pentatope initially generated at level 2 by splitting a c-pentatope along its longest edge, which is the diagonal of a cubic face of hypercube h.
- *e-pentatope*: pentatope initially generated at level 3 by splitting an s-pentatope along its longest edge, which is the diagonal of a square face of hypercube h.

Note that an h-pentatope is then generated at level 4 by the subdivision of an e-pentatope at level 3 along the one among its longest edges, which is an edge of hypercube h.



(a) (b) (c) (d)

Figure 1. Example of (a) an h-pentatope, (b) a c-pentatope, (c) an s-pentatope and (d) an e-pentatope. The figures show the unfolding in 3D space of each pentatope by representing its five tetrahedral faces.

Note that the e-pentatope has three edges with maximum length: the split edge is always the one aligned with the coordinate axis. We will call the *split edge* of a pentatope σ the edge along which σ is split in the recursive subdivision process. In an HP there are h-pentatopes at levels $4j$, c-pentatopes at levels $4j+1$, s-pentatopes at levels $4j+2$, and e-pentatopes at levels $4j+3$, $j = 0, 1, ..., i$.

Figure 1 shows examples of the four possible shapes of a pentatope in an HP: each shape is described by unfolding its five tetrahedral faces.

5. Labeling pentatopes in an HP

Each pentatope σ in a hierarchy of pentatopes, with the exception of those belonging to the subdivision of the initial hypercube, is labeled with one bit, depending on whether σ is the child 0 or child 1 of its parent. In this way, any pentatope in the hierarchy can be uniquely identified through a *location code*. A location code for a pentatope σ in an HP consists of a pair of numbers, in which the first number denotes the level of σ in the tree, while the second number denotes the path from the root of the tree to σ. This path is a sequence of bits each corresponding to a pentatope in the path from the root to σ.

We denote the path-component of the location code of σ as $lc(\sigma)$: $lc(\sigma) = [b_1, ..., b_i]$, where $b_s = 0, 1$ $s = 1, ..., i$. Note that $b_i = 0, 1$, depending on whether σ is labeled as child 0 or child 1 of its parent. In general, $b_s = 0, 1$, $s = 1, ..., i-1$, depending on whether the $(i-s)$-th ancestor of σ is child 0 or child 1 of its parent. Note that from $lc(\sigma)$ we can determine the shape of σ. Let $j = i \bmod 4$: σ is an h-pentatope if $j = 0$, a c-pentatope if $j = 1$, an s-pentatope if $j = 2$, and an e-pentatope if $j = 3$.

Let $\sigma = [v_1, v_2, v_3, v_4, v_5]$ be a pentatope and v_m be the midpoint of the split edge of σ. We denote with σ_0 and σ_1 child 0 and child 1, respectively, of σ. Eight cases arise depending on whether σ is an h-pentatope, a c-pentatope, an s-pentatope, or an e-pentatope, and on the parent-child relations in the hierarchy. Such cases are summarized in Table 1. The table shows the shape of the pentatope which is split, its split edge and the two resulting pentatopes. Note that for a c-pentatope σ, two possible cases arise depending on whether σ is a child 0 or a child 1. For an s-pentatope σ, there are four cases which depend on the parent-child relation between σ and its parent, and between the parent and the grandparent of σ.

6. The neighbor finding problem

The pentatope bisection rule generates nested meshes, that in general are not conforming (see Figure 2 for an example in the case of tetrahedral meshes). To produce a conforming mesh, when bisecting a pentatope σ, all pentatopes that share a common edge with σ, must be split at the same time to guarantee consistency.

We call any set of pentatopes which share their split edge a *diamond*. There are four types of diamonds based on the four choices of orientation of the split edge, that we call *h-diamonds*, *c-diamonds*, *s-diamonds*, and *e-diamonds*, respectively. These four diamonds correspond to the four geometrically similar pentatopes, and each diamond will contain only pentatopes with the same shape:

- an *h-diamond* is a hypercube formed by 24 h-pentatopes (the initial domain subdivision is an h-diamond), all sharing the diagonal of a hypercube, and has 36 tetrahedral and 14 triangle faces.
- a *c-diamond* is formed by 12 c-pentatopes, all sharing the diagonal of a cube (which thus lies on a hyperplane parallel to one of the four coordinate hyperplanes), and has 18 tetrahedral and 8 triangle faces.

Case	Shape	G	P	Split Edge	Resulting Children
1	h-pentatope			$[v_4, v_5]$	c-pentatopes: $\sigma_0 = [v_m, v_1, v_2, v_3, v_5]$, $\sigma_1 = [v_m, v_1, v_2, v_3, v_4]$
2	c-pentatope		0	$[v_4, v_5]$	s-pentatopes: $\sigma_0 = [v_m, v_1, v_2, v_3, v_5]$, $\sigma_1 = [v_m, v_1, v_2, v_3, v_4]$
3	c-pentatope		1	$[v_2, v_5]$	s-pentatopes: $\sigma_0 = [v_m, v_1, v_5, v_3, v_4]$, $\sigma_1 = [v_m, v_1, v_2, v_3, v_4]$
4	s-pentatope	0	0	$[v_4, v_5]$	e-pentatopes: $\sigma_0 = [v_m, v_1, v_2, v_3, v_4]$, $\sigma_1 = [v_m, v_1, v_2, v_3, v_5]$
5	s-pentatope	0	1	$[v_3, v_5]$	e-pentatopes: $\sigma_0 = [v_m, v_1, v_2, v_3, v_4]$, $\sigma_1 = [v_m, v_1, v_2, v_5, v_4]$
6	s-pentatope	1	0	$[v_3, v_4]$	e-pentatopes: $\sigma_0 = [v_m, v_1, v_2, v_5, v_3]$, $\sigma_1 = [v_m, v_1, v_2, v_5, v_4]$
7	s-pentatope	1	1	$[v_3, v_5]$	e-pentatopes: $\sigma_0 = [v_m, v_1, v_2, v_3, v_4]$, $\sigma_1 = [v_m, v_1, v_2, v_5, v_4]$
8	e-pentatope			$[v_4, v_5]$	h-pentatopes: $\sigma_0 = [v_m, v_1, v_2, v_3, v_4]$, $\sigma_1 = [v_m, v_1, v_2, v_3, v_5]$

Table 1. Table with splitting rules. The second column denotes the shape of the pentatope σ which is split, the third column (G) indicates whether the parent of σ is child 0 or child 1 of the grandparent of σ, the fourth column (P) indicates whether σ is child 0 or child 1 of its parent, the fifth column shows the split edge of σ, the sixth column shows the pentatopes resulting from the split, and their vertices.

- an *s-diamond* is formed by 16 s-pentatopes, all sharing the diagonal of a square (which thus lies on a plane parallel to one of the six coordinate planes), and has 24 tetrahedral and 10 triangle faces.
- an *e-diamond* is formed by 48 e-pentatopes, all sharing an edge aligned with one of the four coordinate axes, and has 72 tetrahedral and 26 triangle faces.

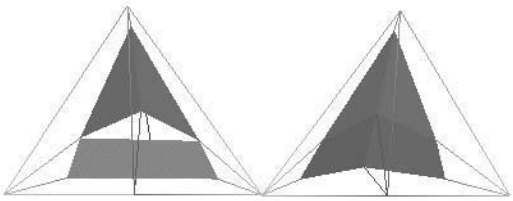


Figure 2. Example of a non-conforming tetrahedral subdivision (on the left) and a conforming one (on the right) along with the corresponding isosurfaces.

Given a diamond D, let us consider the graph G associated with D and embedded on the sphere, defined as follows: the nodes of G are the triangles in D sharing edge e, the arcs of G are the tetrahedra in D sharing edge e, and the faces of G are the pentatopes in D sharing edge e. It is easy to see that each pentatope in D corresponds to a triangular face in G, since each pentatope in D has three tetrahedral faces in common with other pentatopes incident at e.

To produce a nested conforming subdivision, we need to be able to compute efficiently, for each pentatope σ that must be split along an edge e, all pentatopes which belong to the same diamond as σ. Given a pentatope σ and edge e of σ, the problem consists of computing the pentatopes sharing edge e with σ which form a diamond D. This can be performed by traversing the pentatopes incident at e and moving from one pentatope σ to a pentatope adjacent to σ along a tetrahedral face. This process can be viewed as a traversal of the dual graph $G*$ of the graph G associated with diamond D. In graph $G*$, the nodes are the pentatopes in D and the arcs are their tetrahedral faces incident at edge e. Note that each pentatope incident at e has three tetrahedral faces incident at e, and, thus, there are at most three arcs incident in each node of graph $G*$.

The *neighbor finding problem* that we have to solve, can thus be formulated as follows: given a pentatope σ and a tetrahedral face t of σ, find the pentatope σ' sharing face t with σ and having the same shape as σ.

7. The basic neighbor finding algorithm

We denote with f_k, $k = 1, ..., 5$, the tetrahedral face of a pentatope $\sigma = [v_1, v_2, v_3, v_4, v_5]$ defined by all the vertices of σ except v_k. We define five *neighbor types* for a given pentatope $\sigma = [v_1, v_2, v_3, v_4]$ depending on the tetrahedral face shared with σ: the k-neighbor, $k = 1, ..., 5$ of σ is the pentatope σ' that shares face f_k with σ.

The basic approach we use for neighbor finding is similar to that defined in [25] for region quadtrees. The neighbor finding algorithm consists of two steps, which will be described in the following two subsections:

1. Identify the nearest common ancestor of σ and of its k-neighbor σ'.
2. Find the k-neighbor σ': the location code for σ', $lc(\sigma')$, is computed from $lc(\sigma)$ by using the information obtained while finding the nearest common ancestor.

The algorithm requires only a slight modification to take into account the first subdivision of the hypercubic domain into 24 pentatopes.

7.1. First step: locating the nearest common ancestor

Let σ be the given pentatope and σ' the k-neighbor of σ we want to find. We denote the nearest common ancestor of σ and its k-neighbor σ' with σ_A. The objective is to compute, from the location code $lc(\sigma)$ of σ, the location code, $lc(\sigma_A)$, of the nearest common ancestor.

To find σ_A, the hierarchy of pentatopes must be ascended up from σ to σ_A by reversing the path from σ_A to σ. We stop when we identify σ_A. Since we are using a representation based on location codes, the bottom-up retrieval of the nearest common ancestor consists of scanning the bit string in the location code of σ from right to left and deleting the rightmost bit from the bit string at each step. This corresponds to moving to the parent of the current pentatope in the hierarchy. At the end of the process, we obtain the location code of the nearest common ancestor σ_A.

Let $lc(\sigma) = [b_1, \ldots, b_i]$, $b_s = 0, 1$. Bit b_i indicates whether σ is child 0 or child 1 of its parent. The location code of the parent σ_P of σ is $lc(\sigma_P) = [b_1, \ldots, b_{(i-1)}]$. In general, $[b_1, \ldots, b_{(i-q)}]$ is the location code of the q-th ancestor of σ.

Let τ denote the current ancestor of σ we are considering in the tree ascending process towards σ_A. Let $lc(\tau) = [b_1, \ldots, b_r]$, with $r \leq i$. At each step of the traversal, we need

- to identify the neighbor type l of τ we are looking for, i.e., the face f_l of τ that contains face f_k of σ, if we were originally looking for the k-neighbor σ' of σ.
- to establish if τ is the child of the nearest common ancestor σ_A (stopping rule).

To identify the face f_l of τ containing face f_k of σ, we consider the child τ_C of τ and the face f_n of τ_C which contains face f_k. From the location code of τ_C, we can determine whether τ_C is an h-pentatope, a c-pentatope, an s-pentatope or an e-pentatope. Then, face f_l can be determined from face f_n by inverting the labeling rules summarized in Table 1.

Let us consider first some examples. Suppose that τ_C is an h-pentatope and that $n = 4$, i.e., $f_n = f_4$. We invert rule 8, which generates two h-pentatopes from an e-pentatope. Now, $f_4 = [1, 2, 3, 5]$, i.e., f_4 is defined by vertices v_1, v_2, v_3 and v_5 in τ_C. If τ_C is child 0 of τ, then $[1, 2, 3, 5]$ corresponds to vertices v_m, v_1, v_2, v_4 in the subdivision of the parent τ by vertex v_m (which is the midpoint of edge $[v_4, v_5]$). Thus, the face in τ containing face f_4 of τ_C is face $f_3 = [1, 2, 4, 5]$. If τ_C is child 1 of τ, then $[1, 2, 3, 5]$ corresponds to vertices v_m, v_1, v_2, v_5 in the subdivision of parent τ by vertex v_m. The face in τ containing face f_4 of τ_C is again face $f_3 = [1, 2, 4, 5]$.

Suppose that τ_C is an h-pentatope and that $n = 1$, i.e., $f_n = f_1$. Now, $f_1 = [2, 3, 4, 5]$, i.e., f_1 is defined by vertices v_2, v_3, v_4 and v_5 in τ_C. If τ_C is child 0 of τ, then $[2, 3, 4, 5]$ corresponds to vertices v_1, v_2, v_3, v_4 in the subdivision of the parent τ. Thus, the face in τ containing face f_1 of τ_C is face $f_5 = [1, 2, 3, 4]$. If τ_C is child 1 of τ, then $[2, 3, 4, 5]$ corresponds to vertices v_1, v_2, v_3, v_5 in the subdivision of parent τ. Thus, the face in τ containing face f_1 of τ_C is face $f_4 = [1, 2, 3, 5]$.

In general, let σ_0 and σ_1 be two children of a pentatope σ in any of the rules in Table 1. Let $\sigma_0 = [v_m, v_i^0, v_j^0, v_k^0, v_q^0]$ and $\sigma_1 = [v_m, v_i^1, v_j^1, v_k^1, v_q^1]$. Now, if $n = 1$, i.e., $f_n = f_1$ in τ_C, then $f_l = [v_i^0, v_j^0, v_k^0, v_q^0]$ if τ_C corresponds to σ_0, or $f_l = [v_i^1, v_j^1, v_k^1, v_q^1]$ if τ_C corresponds to σ_1. If $n \neq 1$, then let $f_n = [1, a, b, c]$, where a, b and c assume distinct values between 2 and 5. Let v_a, v_b, v_c be the three vertices in $\sigma_0 = [v_m, v_i^0, v_j^0, v_k^0, v_q^0]$ or in $\sigma_1 = [v_m, v_i^1, v_j^1, v_k^1, v_q^1]$ in positions a, b and c, depending on whether τ_C corresponds to σ_0 or to σ_1. Let $[v_p, v_q]$ denote the edge of τ split by v_m. Then, f_l in τ is the face defined by four distinct vertices in the set $\{v_a, v_b, v_c, v_p, v_q\}$. Note that one of the vertices v_a, v_b, v_c is the same as either v_p or v_q.

To identify the nearest common ancestor σ_A of σ and σ', we observe that σ_A is the pentatope which is split by the tetrahedral face f_A containing the k-face f_k of σ. Thus, σ_A is the parent of the ancestor $\sigma*$ of σ bounded by face f_A. This does not have to be verified geometrically, but we can decide whether we need to continue the process or stop based on the shape of the current simplex τ (whether it is a e-pentatope, an h-pentatope, a c-pentatope or an s-pentatope), on the type of neighbor, and the value of the rightmost four bits in the current location code.

From the labeling rules summarized in Table 1, the parent of τ is the nearest common ancestor σ_A if and only if one of the rules reported in Table 2 applies. In Table 2, the second column reports the possible shape types of simplex τ, the third and the fourth column the child type of the parent of τ and of τ, respectively, the fifth column the face f_k along which we are looking for a neighbor (i.e., the neighbor type), the sixth column the condition on the correspondence between f_k and the splitting face in the parent and the seventh column the shape type of the parent. For instance, if τ is a c-pentatope, and we are looking for neighbor along face $f_5 = [1, 2, 3, 4]$, the parent of τ is the nearest common ancestor iff f_5 corresponds to the splitting face $[v_m, v_1, v_2, v_3]$ in the parent of τ.

7.2. Step 2: Computing the neighbor and its location code

In the second step, we need to find the k-neighbor σ' of σ, and we identify it through its location code $lc(\sigma')$. Find-

Case	Shape	G	P	Sibling Face	Condition	Shape of the Parent
1	c-pentatope			$f_5 = [1,2,3,4]$	f_5 corresponds to $[v_m, v_1, v_2, v_3]$	h-pentatope
2	s-pentatope		0	$f_5 = [1,2,3,4]$	f_5 corresponds to $[v_m, v_1, v_2, v_3]$	c-pentatope
3	s-pentatope		1	$f_3 = [1,2,4,5]$	f_3 corresponds to $[v_m, v_1, v_3, v_4]$	c-pentatope
4	e-pentatope	0	0	$f_5 = [1,2,3,4]$	f_5 corresponds to $[v_m, v_1, v_2, v_3]$	s-pentatope
5	e-pentatope	0	1	$f_4 = [1,2,3,5]$	f_4 corresponds to $[v_m, v_1, v_2, v_4]$	s-pentatope
6	e-pentatope	1	0	$f_5 = [1,2,3,4]$	f_5 corresponds to $[v_m, v_1, v_2, v_5]$	s-pentatope
7	e-pentatope	1	1	$f_4 = [1,2,3,5]$	f_4 corresponds to $[v_m, v_1, v_2, v_4]$	s-pentatope
8	h-pentatope			$f_5 = [1,2,3,4]$	f_5 corresponds to $[v_m, v_1, v_2, v_3]$	e-pentatope

Table 2. Table with inverse rules. The first column shows the case, the second column reports the possible shape types of simplex τ, the third and the fourth column the child type of the parent of τ and of τ, respectively, the fifth column the face f_k along which we are looking for a neighbor (i.e., the neighbor type), the sixth column the condition on the correspondence between f_k and the splitting face in the parent and the seventh column the shape type of the parent.

ing σ' requires descending the hierarchy of pentatopes from the nearest common ancestor σ_A by reflecting the path from σ_A to σ.

The location code $lc(\sigma')$ of σ' is obtained from the location code $lc(\sigma)$ of σ by just inverting the bit in position p corresponding to the child of the nearest common ancestor σ_A. If $lc(\sigma) = b_1, b_2, ..., b_i$ and b_p is the bit corresponding to the child of σ_A, then $lc(\sigma_A) = [b_0, b_2, ..., b_{(p-1)}]$. Thus, all the bits on the left of p are unchanged in the location code of σ'. The bits in $lc(\sigma)$ on the right of b_p are also not affected because of the way the pentatopes are labeled. The labeling technique ensures that face-adjacent pentatopes of the same shape are reflections of each other. This process works regardless of the original neighbor type which we are trying to find, since the location code of the nearest common ancestor of σ and σ' is the only information needed in order to determine which bit needs to be inverted.

8. Neighbor finding in constant time

In this section, we describe how to perform neighbor finding in worst-case constant time. For the sake of simplicity, we consider only the case in which the input pentatope σ is an h-pentatope. In the case that σ is not an h-pentatope, we just need to move up in the hierarchy by at most four levels: either we find the nearest common ancestor (and, thus, the k-neighbor we are looking for) in a maximum of four steps (changes in level), or we find an h-pentatope, since the four shapes are cyclic on four levels. Note that h-diamonds are hypercubes with cubic faces subdivided into tetrahedra.

In Subsection 8.1, we examine the rules for finding the nearest common ancestor of σ and of its k-neighbor σ' in order to determine the bit to be changed for finding σ'. In Subsection 8.2, we describe how such rules can be applied to the location code of σ to generate the location code of σ' in worst-case constant time.

8.1. Finding the five neighbors of an h-pentatope

Let $lc(\sigma) = [b_1, \ldots, b_i]$. We partition $lc(\sigma)$ into groups of four bits starting from the rightmost bit b_i: four bits correspond to four consecutive levels in the hierarchy. Let us consider the rightmost four bits in the location code of σ, that we denote as B_1, B_2, B_3, B_4, from left to right. Bit B_4 corresponds to σ (an h-pentatope), B_3 to the parent σ_1 (an s-pentatope) of σ, B_2 to the grandparent σ_2 (a c-pentatope) of σ, and B_1 to the third ancestor σ_3 (an e-pentatope) of σ. Table 3 summarizes the results of neighbor finding by just looking at these last four bits (see the first column): it specifies how the corresponding four bits change, if the neighbor can be determined by considering B_1, B_2, B_3, B_4, or the neighbor type to look for in the next group of four bits to the left of B_1, B_2, B_3, B_4 in the location code of σ, otherwise. Table 3 is generated by inverting the subdivision rules introduced in Section 5.

As an example, we describe how the rules for neighbors of type 5 and 4 are derived. The neighbor of type 5 of an h-pentatope σ (i.e., along face f_5) is always the sibling of σ, since an h-pentatope is generated by splitting an e-pentatope (see rule 8 in Subsection 5) and face f_5 of σ is the splitting face $[v_m, v_1, v_2, v_3]$ in the parent of σ. Finding the sibling is simply a matter of inverting the bit B_4 in the location code.

The neighbor of type 4 of an h-pentatope σ (i.e., along face f_4) is always outside the h-diamond defined by the h-pentatope which is the fourth ancestor τ of σ and by those sharing their longest edge with τ. Face f_4 of σ is always contained in either face f_4 or face f_5 of the fourth ancestor τ. Let σ_1, σ_2,σ_3, and σ_4 denote the parent, grandparent, third and fourth ancestor of σ, respectively. Face $f_4 = [1,2,3,5]$ in σ is contained in face $f_3 = [1,2,4,5]$ of e-pentatope σ_1, since $f_4 = [1,2,3,5]$ corresponds to sub-face $[v_m, v_1, v_2, v_4]$ in σ_1 when split by vertex v_m, if σ is child 0 of σ_1 or to sub-face $[v_m, v_1, v_2, v_5]$ in σ_1 when split

Current Bits	Neighbor 5	Neighbor 4	Neighbor 3	Neighbor 2	Neighbor 1
0000	0001	Cont 4	Cont 3	0100	0010
0001	0000	Cont 4	Cont 3	0101	Cont 1
0010	0011	Cont 4	Cont 3	Cont 2	0000
0011	0010	Cont 4	Cont 3	Cont 2	Cont 1
0100	0101	Cont 4	1100	0000	Cont 2
0101	0100	Cont 4	1101	0001	0111
0110	0111	Cont 4	1110	Cont 1	Cont 2
0111	0110	Cont 4	1111	Cont 1	0101
1000	1001	Cont 5	Cont 1	Cont 2	1010
1001	1000	Cont 5	Cont 1	Cont 2	Cont 3
1010	1011	Cont 5	Cont 1	1110	1000
1011	1010	Cont 5	Cont 1	1111	Cont 3
1100	1101	Cont 5	0100	Cont 3	Cont 2
1101	1100	Cont 5	0101	Cont 3	1111
1110	1111	Cont 5	0110	1010	Cont 2
1111	1110	Cont 5	0111	1011	1101

Table 3. The table indicates how to proceed at each level when searching for the neighboring pentatope.

by vertex v_m, if σ is the child 1 of σ_1 (by applying rule 8). Face f_4 in σ_1 is contained in face $f_2 = [1, 3, 4, 5]$ of s-pentatope σ_2, in all four cases defined by rules 4-7. Face f_2 in σ_2 is contained in face f_1 of c-pentatope σ_3 (by applying rules 2 and 3). Finally, face f_1 in σ_1 is contained in face f_4 of σ_4 if σ_3 is child 0 of σ_4, face f_1 in σ_1 is contained in face f_5 of σ_4 if σ_3 is child 1 of σ_4 (by applying rule 1). Note that, in this latter case, σ_4 is an h-pentatope and that the 4-neighbor of σ is just the sibling of σ_4.

When the neighbor can be identified by looking at the four bits B_1, B_2, B_3, B_4, we need to invert bit B_4 for neighbor of type 5, bit B_1 for neighbor of type 3, bit B_2 for neighbor of type 2, and bit B_3 for neighbor of type 1. We need to continue the search on the next group of four bits in the following cases:

- Neighbor of type 4: continue if $not(B_1)$
- Neighbor of type 3: continue if $not(B_2)$
- Neighbor of type 2: continue if $not(B_1 = B_3)$
- Neighbor of type 1: continue if $not(B_2 = B_4)$

Note that, for neighbor of type 4, the location code is always modified beyond the current four bits, but if B_1, the neighbor is the sibling of the fourth ancestor σ_4 of σ, thus we simply need to invert the first bit on the left of B_1 in the location code. If $not(B_1)$, then we need to continue searching using the same neighbor type.

For the remaining three neighbor types (3, 2, and 1), the neighbor type at the next level is determined by the combination of bits B_1 and B_2, namely:

- $not(B_1)\ and\ not(B_2)$: continue with the same neighbor type
- $not(B_1)\ and\ B_2$: if we are looking for a neighbor of type 2, we continue with a neighbor of type 1 and vice versa
- $B_1\ and\ not(B_2)$: if we are looking for a neighbor of type 3, we continue with a neighbor of type 1 and vice versa
- $B_1\ and\ B_2$: if we are looking for a neighbor of type 3, we continue with a neighbor of type 1; if we are looking for a neighbor of type 2, we continue with a neighbor of type 3; if we are looking for a neighbor of type 1, we continue with a neighbor of type 2.

We can apply the rules described in Table 3 to each group of four consecutive bits in the location code of σ by proceeding right to left. We would do bit operations to identify the different bit patterns, but this is still a sequential search which does not achieve a constant time behavior. To this aim, we need to be able to predict the neighbor type we will be looking for in all groups of four bits at the same time, thus avoiding an iterative process.

8.2. Techniques for constant-time neighbor finding

We now present techniques that make use of the carry property of addition to find a neighbor without specifically searching for the nearest common ancestor. In particular, we replace the step-by-step process sketched in the previous subsection, by an arithmetic operation that takes constant time instead of time proportional to the depth of the tree. The algorithms make use of bit manipulation operations which can be implemented in hardware using a few

machine language instructions. Of course, the constant time bound arises because the entire bit string which identifies the path in the location code is assumed to fit in one computer word. This, however, allows us to deal with data sets containing up to 256 points in each of the four dimensions, or over 10^9 total points.

Since our goal is to use bit operations in order to find the nearest common ancestor, we need to identify which bit patterns indicate that the nearest common ancestor is farther up in the tree (beyond the current set of four bits). Being able to identify these patterns in a logical notation, as in Section 8.1, allows for a conversion into bit operations that perform the same logical function over the entire location code in a fixed number of operations. This is regardless of the actual length of the location code, since bit operations can be performed over an entire computer word in a single operation.

Identifying positions in the location code where we need to continue searching for the nearest common ancestor can be done simultaneously for all sets of four bits as described below. Let us denote with b_p the bit in position p in the location code of σ

- Neighbor of type 4: continue if $not(b_{4q+1})$
- Neighbor of type 3: continue if $not(b_{4q+2})$
- Neighbor of type 2: continue if $not(b_{4q+1} = b_{4q+3})$
- Neighbor of type 1: continue if $not(b_{4q+2} = b_{4q+4})$

where $q = 0, 1, \ldots, i/4$.

This assumes that the neighbor type does not change from level to level. Since neighbors of types 3, 2, and 1 change depending on bits b_{4q+1} and b_{4q+2}, we need to capture those changes in another mask, called the *neighbor mask*, in order to determine which neighbor type to use in each group of four bits. This information needs to be available before processing the location code so that all operations can occur simultaneously.

For a given starting neighbor type, two bits in the neighbor mask can be used to determine the appropriate neighbor type for each set of four bits (b_{4q+1},b_{4q+2},b_{4q+3},b_{4q+4}) in the location code. We simply code a neighbor of type 3 as 01, type 2 as 10, and type 1 as 11, in our neighbor mask. When starting with neighbor type 3, we use bits m_{4q+1} and m_{4q+2} in the neighbor mask. Likewise, we use bits m_{4q+3} and m_{4q+4} to capture the state when starting with neighbor type 2. We could use an additional two bits in the neighbor mask for the case when we are starting with neighbor type 1, but these two bits would always contain the only remaining neighbor choice for each set of four bits in the location code, and so we can save memory by avoiding this redundant information.

The complete logical expression that determines when to continue when starting with neighbor type 3 is given below.

$$
\begin{aligned}
&(not(m_{4q+1})\ and\ m_{4q+2}\ and\ not(b_{4q+2}))\ or\\
&(m_{4q+1}\ and\ not(m_{4q+2})\ and\ not(b_{4q+1} = b_{4q+3}))\ or\\
&(m_{4q+1}\ and\ m_{4q+2}\ and\ not(b_{4q+2} = b_{4q+4}))
\end{aligned}
$$

Notice that this expression combines the logical parts given previously for each of the three neighbor types (3, 2, and 1). The complete logical expressions for neighbor types 1 and 2 are similar. We simply substitute the appropriate bits for m_{4q+1} and m_{4q+2} in the expression for neighbor type 3.

Using the above expression, we can identify all positions in the location code which should propagate a carry. This makes finding the location of the nearest common ancestor as simple as a single addition, where the final carry determines which bit gets inverted in order to get the neighboring pentatope.

9. Concluding remarks

We have considered a decomposition of a hypercube into nested four-dimensional simplexes that we called pentatopes, thus generating a hierarchy of pentatopes (HP). We have developed a labeling technique for nested pentatopes which enables us to identify a pentatope through its location code. We have shown how face-neighbors of a pentatope can be extracted by manipulating location codes, and we have proposed a neighbor finding algorithm which works in worst-case constant time. The constant-time behavior is achieved by using bit manipulation operations.

We have considered the application of a hierarchy of pentatopes to the multi-resolution representation of four-dimensional scalar fields. Our aim is to develop tools for extracting approximated, simplified, representations of a time-varying volumetric data sets from its multi-resolution representation. Such simplified representations are based on adaptive meshes of smaller size with respect to the mesh at full resolution, and having a resolution varying in different parts of the field domain, or in the proximity of interesting field values, according to user requirements. The meshes extracted from a multi-resolution representation must be conforming as so as to avoid discontinuities in the corresponding field approximation. Generating conforming nested meshes requires computing diamonds composed of pentatopes which must be split at the same time. Diamonds can be extracted from the HP by finding the neighbors of a pentatope along one of its five tetrahedral faces, and this can be done efficiently by using the neighbor finding algorithm proposed here.

A hierarchy of pentatopes, when used as the domain decomposition of a four-dimensional scalar field, does not need to be explicitly stored either in a pointer-based representation or through location codes.

Current and future work includes developing algorithms for generating an HP from time-varying volumetric data sets, for extracting isosurfaces from an HP at variable resolution as well as a tool for visualizing and analyzing such isosurfaces at different time frames.

Acknowledgments

This work has been partially supported by project *MACROGeo: Algorithmic and Computational Methods for Geometric Object Representation* funded by the Italian Ministry of Education, University, and Research (MIUR).

References

[1] P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in higher dimensions. In *Proceedings IEEE Visualization 2000*, pages 267–273. IEEE Computer Society, Oct. 2000.

[2] E. Danovaro, L. De Floriani, P. Magillo, and E. Puppo. Data structures for 3D Multi-Tessellations: an overview. In H. Post, G. Bonneau, and G. Nielson, editors, *Proceedings Dagstuhl Scientific Visualization Seminar*. Kluwer Academic Publishers, 2002.

[3] L. De Floriani and M. Lee. Selective refinement on nested tetrahedral meshes. In G. Brunnett, B. Hamann, H. Mueller, and L. Linsen, editors, *Geometric Modeling for Scientific Visualization*, pages 329–344. Springer Verlag, 2003.

[4] 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 in Geometric Modeling*, Lecture Notes in Mathematics, Springer Verlag, Berlin (D), 2002.

[5] W. Evans, D. Kirkpatrick, and G. Townsend. Right-triangulated irregular networks. *Algorithmica*, 30(2):264–286, 2001.

[6] I. Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, 25(12):905–910, Dec. 1982.

[7] T. Gerstner and M. Rumpf. Multiresolutional parallel isosurface extraction based on tetrahedral bisection. In *Proceedings 1999 Symposium on Volume Visualization*. IEEE Computer Society, 1999.

[8] B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings IEEE Visualization 2002*, Boston, MA, Oct. 2002. IEEE Computer Society.

[9] G. Greiner and R. Grosso. Hierarchical tetrahedral-octahedral subdivision for volume visualization. *The Visual Computer*, 16:357–369, 2000.

[10] 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 '97*, pages 387–394, Phoenix, AZ, Oct. 1997.

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

[12] L. Ibarria, P. Linstrom, J. Rossignac, and A. Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum*, 22(3), 2003.

[13] G. Ji, H. Shen, and R. Wenger. Volume tracking using higher dimensional isosurfacing. In J. v. W. G. Turk and R. Moorhead, editors, *Proceedings IEEE Visualization 2003*, pages 209–216. IEEE Computer Society, Oct. 2003.

[14] M. Lee, L. De Floriani, and H. Samet. Constant-time neighbor finding in hierarchical tetrahedral meshes. In *Proceedings International Conference on Shape Modeling & Applications*, pages 286–295, Genova, Italy, May 2001.

[15] M. Lee and H. Samet. Navigating through triangle meshes implemented as linear quadtrees. *ACM Transactions on Graphics*, 19(2):79–121, Apr. 2000.

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

[17] M. Ohlberger and M. Rumpf. Hierarchical and adaptive visualization on nested grids. *Computing*, 56(4):365–385, 1997.

[18] V. Pascucci. Slow growing subdivision (SGS) in any dimension: towards removing the curse of dimensionality. *Computer Graphics Forum*, 21(3), 2002.

[19] 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, Oct. 2000. IEEE Computer Society.

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

[21] J. 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.

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

[23] H. Samet. Neighbor finding techniques for images represented by quadtrees. *Computer Graphics and Image Processing*, 18(1):37–57, Jan. 1982.

[24] H. Samet. Neighbor finding in images represented by octrees. *Computer Vision, Graphics, and Image Processing*, 46(3):367–386, June 1989. Also University of Maryland Computer Science TR-1968.

[25] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.

[26] G. Schrack. Finding neighbors of equal size in linear quadtrees and octrees in constant time. *CVGIP: Image Understanding*, 55(3):221–230, May 1992.

[27] C. Weigle and D. Banks. Complex-valued contour meshing. In *Proceedings IEEE Visualization 1996*, pages 173–180. IEEE Computer Society, Oct. 1996.

[28] C. Weigle and D. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *Proceedings IEEE Visualization 1998*, pages 103–110. IEEE Computer Society, Oct. 1998.

[29] 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 '97*, pages 135–142, Phoenix, AZ, Oct. 1997.