The PR-star Octree: A spatio-topological data structure for tetrahedral meshes

Kenneth Weiss
Dept. of Computer Science
University of Maryland
College Park, Maryland
kweiss@cs.umd.edu

Riccardo Fellegara
Dept. of Computer Science
University of Genova
Genova, IT
riccardo.fellegara@disi.unige.it

Leila De Floriani
Dept. of Computer Science
University of Genova
Genova, IT
deflo@disi.unige.it

Marcelo Velloso
Dept. of Computer Science
University of Maryland
College Park, Maryland
mvelloso@umd.edu

ABSTRACT

We propose the *PR-star octree* as a combined spatial data structure for performing efficient topological queries on tetrahedral meshes. The PR-star octree augments the *Point Region octree (PR Octree)* with a list of tetrahedra incident to its indexed vertices, i.e. those in the *star* of its vertices. Thus, each leaf node encodes the minimal amount of information necessary to locally reconstruct the topological connectivity of its indexed elements. This provides the flexibility to efficiently construct the optimal data structure to solve the task at hand using a fraction of the memory required for a corresponding data structure on the global tetrahedral mesh. Due to the spatial locality of successive queries in typical GIS applications, the construction costs of these runtime data structures are amortized over multiple accesses while processing each node. We demonstrate the advantages of the PR-star octree representation in several typical GIS applications, including detection of the domain boundaries, computation of local curvature estimates and mesh simplification.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures—*Graphs and networks*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations Hierarchy and geometric transformations*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Graphics data structures and data types*

1. INTRODUCTION

There has been an increasing interest in many fields, including geographic informations systems (GIS), finite element analysis, geological modeling, urban modeling and scientific visualization, in discretizing the objects of interest as tetrahedral meshes. Tetrahedral meshes tend to be large, with respect to triangle meshes used as terrain models, since the number of tetrahedra in a mesh is approximatively six times the number of vertices. Advances in sensing technologies have enabled the acquisition of larger scientific datasets with finer resolution, leading to new challenges in allocating resources for storing, processing and visualizing such meshes. However, the faster rate of growth in processing power over memory favors reductions in global memory requirements at the expense of (modest) increases in processing requirements.

Queries involving topological connectivity are fundamental for many tasks which require local *navigation* on the mesh elements. Examples include: visibility and viewshed operations, morphological operations, segmentation and connected component determination, mesh analysis and repair, ray tracing/path following and estimates of the discrete curvature, Laplacian, normal and gradient.

Such operations are typically accelerated through the use of a *topological data structure*, that is, a data structure that supports the efficient reconstruction of a subset of the local topological connectivity of the mesh. Since applications on such datasets can differ greatly in their access patterns to the data, various data structures have been proposed to facilitate specific application-dependent tasks. For example, *line of sight* algorithms are often optimized to compute the adjacencies of the mesh elements, while curvature-based algorithms are more concerned with the set of elements incident to a given point or edge in the mesh.

The major contribution of this work is a new data structure, the *PR-star octree*, in which we obtain the local topological connectivity of a tetrahedral mesh through its spatial locality. In contrast to previous topological data structures, which have focused on the adjacencies or incidences of the mesh elements, we use a spatial data structure on its embedding space to locally reconstruct the optimal application-dependent topological representation at runtime using the sorted geometry available from our spatial index. Thus, the innovative feature of our approach is in computing topology through space: local spatial sorting allows the efficient reconstruction of the local mesh connectivity. Although this increases the cost of a single operation due to the construction of the local data structure, this cost is amortized over multiple accesses to elements within the same region. Moreover, by recovering the memory associated with each local data structure after processing of that part of the mesh has completed, we achieve significant memory savings with respect to global topological data structures.

We demonstrate the effectiveness of this approach through several typical applications on tetrahedral meshes including boundary determination, i.e., computing the tetrahedra which are on the

ACM SIGSPATIAL GIS '11, November 1–4 2011. Chicago, IL, USA

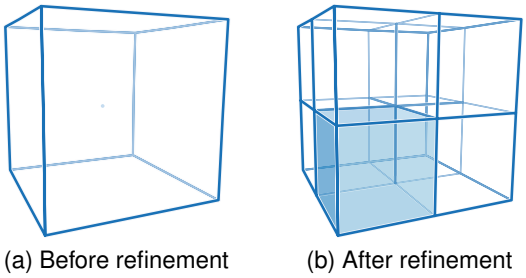


Figure 1: Regular refinement of a *parent* octree node generates eight similar *children* nodes covering its domain.

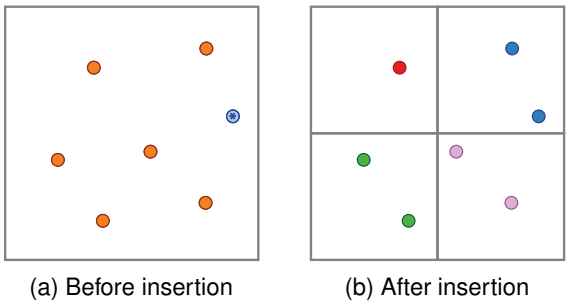


Figure 2: Insertion of a point (blue) into a full leaf node of a PR octree (shown in 2D, with bucket threshold $k_v = 6$) causes the node to refine. (a) before insertion (b) after insertion.

boundary of the mesh, estimating 3D curvature, which is important for meshes discretizing the domain of 3D scalar fields, and mesh simplification.

The remainder of this paper is organized as follows. In Sections 2 and 3, we review background notions and related work. In Section 4, we introduce the PR-star octree and discuss its properties, while in Section 5, we evaluate its storage cost and compare it to a state of the art compact topological data structure. In Section 6, we discuss the implementation of the fundamental queries on the PR-star octree. In Section 7, we describe how to efficiently determine the boundary of the mesh from the PR-star octree, while in Sections 8 and 9, we discuss applications of the PR-star octree to 3D curvature computation and to mesh simplification, respectively. Finally, in Section 10, we draw some concluding remarks.

2. BACKGROUND NOTIONS

The *PR-star octree* combines notions from the PR octree, a spatial data structure over point datasets, with those of the indexed representation for tetrahedral meshes.

2.1 The PR octree

An octree [13, 20] is a hierarchical domain decomposition based on the nested refinement of a cube into eight cubes covering its domain. The containment relation among such cubes defines a hierarchical relationship among the set of nodes in the octree, where a *parent* node's eight *children* in the octree are the cubes generated during its refinement. The *root* of an octree covers the entire cubic domain and is the only node without a parent. Nodes with children (i.e. refined cubes) are referred to as *internal nodes* of the octree, while those without children are referred to as *leaf nodes* of the octree. In the following, we focus on octrees generated by *regular refinement* (i.e. *region octrees*), in which all eight children of a refined node are similar and intersect at the midpoint of their parent's domain (see Figure 1).

A *Point Region octree (PR octree)* [20] is a spatial index on a set of points in a three-dimensional domain. The domain decomposition is controlled by a bucket threshold k_v that determines the maximum number of points that each leaf node of the octree can index. A leaf node is considered *full* when it indexes k_v points. Insertion of a new point into a full leaf node causes the node to split and its indexed points to be redistributed among its eight children nodes (see Figure 2 for an illustration in 2D). As such, the domain decomposition of a PR octree depends only on the bucket threshold k_v, and is independent of the insertion order of its points [20].

2.2 Topological relations

An n-dimensional Euclidean simplex in $\mathbb{R}^3$ ($n \leq 3$) is the convex hull of $n+1$ linearly independent points. In particular, a 0-simplex is a *vertex*, a 1-simplex is an *edge*, a 2-simplex is a *triangle* and a 3-simplex is a tetrahedron. A k-dimensional face of an n-simplex σ is defined by any set of $k+1$ vertices of σ ($0 \leq k \leq n$).

A collection of tetrahedra T forms a *tetrahedral mesh* Σ when the interiors of its tetrahedra are pairwise disjoint. A tetrahedral mesh is *conforming* if the intersection of any two tetrahedra σ_1 and σ_2 in T is either empty or is a common triangular face, edge or vertex of σ_1 and σ_2.

Two simplices are *incident* in each other if one of them is a face of the other, while they are *k-adjacent* if they share a k-face. For simplicity, we refer to k-adjacent $(k+1)$-simplices and to vertices incident in a common edge as *adjacent*.

The *(combinatorial) boundary* of a simplex σ is defined by the set of its faces. The *star* of a simplex σ, denoted $St(\sigma)$, is the set of simplices in a tetrahedral mesh Σ that have σ as a face. The *link* of a simplex σ, denoted $Lk(\sigma)$, is the set of all the faces of the simplices in $St(\sigma)$ which are not incident in σ.

Queries on a tetrahedral mesh are often posed in terms of the *topological relations* defined by the adjacencies and incidences of its simplices. Let us consider a conforming tetrahedral mesh Σ and a p-simplex $\sigma \in \Sigma$, with $0 \leq p \leq 3$.

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

Co-boundary relation $R_{p,q}(\sigma)$, with $p < q \leq 3$, consists of the set of q-simplices in Σ incident in σ.

Adjacency relation $R_{p,p}(\sigma)$ consists of the set of p-simplices in Σ that are adjacent to σ.

When the two types of simplex in a topological relation are specified, we can refer to the relation by their simplex types. For example, the tetrahedra in the co-boundary of a given vertex v, $R_{0,3}(v)$ can be referred to as the *Vertex-Tetrahedra* relation of v.

2.3 Indexed tetrahedral meshes

An *indexed tetrahedral mesh* is a common boundary-based data structure for a tetrahedral mesh. It encodes the *Tetrahedron-Vertex* relation, i.e., the relation among a tetrahedron and its four vertices.

It consists of two arrays: The vertices V, encode the geometry of the mesh in terms of their coordinates in $\mathbb{R}^3$, while the tetrahedra T are encoded in terms of the indices in V of their four vertices.

Since the indexed representation explicitly encodes $R_{3,0}$ (i.e. the *Tetrahedron-Vertex* relation), it supports the efficient extraction of the boundary relations, but not the co-boundary or adjacency relations. For example, to find the tetrahedron adjacent to a given tetrahedron along one of its faces, we must (in the worst case) iterate through the entire list of tetrahedra in T.

The Indexed data structure with Adjacencies (*IA data structure*)

[19, 18] extends the indexed representation by explicitly encoding the *Tetrahedron-Tetrahedra* relation $R_{3,3}$, i.e. the tetrahedra adjacent to the four faces of each tetrahedron. An extension to the IA data structure, which we refer to as the *extended IA data structure*, in which a single tetrahedron in the star of each vertex (i.e. a partial *Vertex-Tetrahedron* relation $R^*_{0,3}$) is also encoded, enables the efficient extraction of all topological relations. In our comparisons, we will evaluate the PR-star data structure against this extended version of the IA data structure.

3. RELATED WORK

Hierarchical spatial indexes for points in the 3D Euclidean space are provided by PR octrees and PR kd-trees [20]. In these indexes, the shape of the tree is independent of the order in which the points are inserted, and the points are only indexed by leaf nodes.

Some data structures have been proposed for spatial indexing of *polygonal maps (PM)*, including graphs, planar triangle meshes and tetrahedral meshes. The PMR quadtree [17] is a spatial index for a collection of edges in the plane (not necessarily forming a polygonal map), and can be used for spatial objects in the plane [12]. The class of *PM quadtrees* [21] extends the PR quadtree (which is for points in the plane) to represent polygonal maps considered as a structured collection of edges. There are three variations, namely the *PM1 quadtree*, the *PM2 quadtree* and the *PM3 quadtree*, which differ in the criterion to define the content of a leaf node. They all maintain a list of edges in the leaf nodes. In [5] we have extended the PM2 quadtree to produce a hierarchical spatial index for triangle meshes. In this index, a node is a leaf if and only if either (a) it contains one vertex, and all intersecting triangles are incident in that vertex, or (b) it contains no vertex, and all intersecting triangles have a common vertex lying outside the leaf node.

PM octrees are used to index the boundary of a polyhedral object in space [2, 16, 20]. In particular, PM1-, PM2-, and PM3 octrees have been proposed, where the subdivision rule is similar to the corresponding quadtrees but considers faces instead of edges. In [6], we have developed a collection of spatial indexes for tetrahedral meshes, that we call *Tetrahedral trees*. One tetrahedral tree extends PMR quadtrees to the 3D case, where instead of edges we consider tetrahedra, while the other extends the PM octree to the case of tetrahedral meshes. Our approach is fundamentally and conceptually different from this previous work since the spatial hierarchy is not a spatial index on the mesh (i.e. to support efficient spatial queries such as point location), but it is a tool to support efficient retrieval of topological connectivity (i.e. for topological queries), thus encoding a minimum topology and trading spatial relations for topological ones.

Other approaches utilize a spatial index to reduce the memory requirements for out-of-core [4] or memory intensive mesh processing [7]. Cignoni et al. [4] introduce an external memory spatial data structure for processing large triangle meshes. Whereas, our aim is to enable efficient topological operations on the mesh elements, and thus we allow tetrahedra to be indexed more than once, the aim of [4] is to support compact out-of-core processing of large triangle meshes. As such, they only encode each triangle a single time, and require explicit management of the mesh elements entirely resident in memory (i.e. those with all incident vertices loaded in memory) and the elements that are only partially resident in memory (i.e. those with only a subset of the vertices loaded in memory). Dey et al. [7] use an octree to index a large triangle mesh for localized Delaunay remeshing. Due to the significant overhead associated with their computations, their octrees are very shallow, and contain very few octree nodes (e.g. octrees with only eight nodes are typical in their experiments).

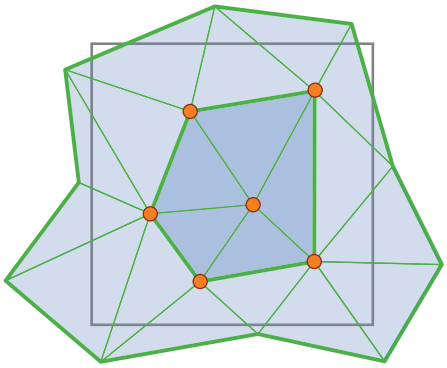


Figure 3: A leaf node in a PR-star octree (shown in 2D with bucket threshold $k_v = 6$) encodes a set of vertices and all tetrahedra incident in those vertices.

Recently, Gurung et al. [11] introduced a compact version of the extended IA data structure, which they call the *Sorted Vertex Opposite Table (SVOT) data structure*. This data structure implicitly encodes the partial Vertex-Tetrahedra relation $R^*_{0,3}$ by rearranging the order of the tetrahedra within the tetrahedra array T, and reduces the size of the boundary relation $R_{3,0}$ by reconstructing it through a traversal along the adjacency relation $R_{3,3}$. Since modifications to the mesh require non-local reconstructions of the associated data structures, this representation is more suitable for static meshes.

4. THE PR-STAR OCTREE

In this section, we introduce the *PR-star octree* as a spatio-topological data structure for a tetrahedral mesh. In contrast to topological data structures or to spatial data structures, our aim is to use the spatial index induced by the octree to support efficient generation of optimal application-dependent topological data structures at runtime.

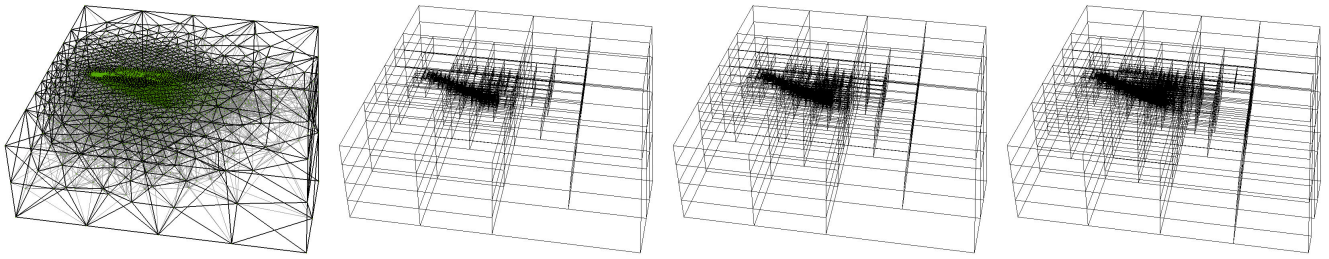
The PR-star octree combines the indexed tetrahedral mesh representation (Section 2.3) with an augmented PR octree (see Section 2.1) that also indexes the set of tetrahedra in the star of its indexed vertices (see Figure 3). Thus, a PR-star octree over a tetrahedral mesh is represented using three entities:

- An array of vertices V, encoding the geometry of the mesh.
- An array of tetrahedra T. Each tetrahedron in T is encoded in terms of the indices of its four vertices within V (e.g. the *Tetrahedron-Vertex* boundary relation $R_{3,0}$).
- An augmented PR octree N, whose leaf nodes index the set of vertices within its domain, as well as the set of all tetrahedra incident in these vertices.

We generate the PR-star octree from a tetrahedral mesh in three phases. First, given a user-defined bucket threshold k_v, we generate a PR octree decomposition on the vertices V of the tetrahedral mesh. Figure 4 shows spatial decompositions over the F117 dataset for different values of k_v. Note that lower values of k_v lead to finer spatial decompositions in octree N.

Next, we insert the tetrahedra of T into the leaf nodes of N that index their vertices. That is, for each vertex v of a tetrahedron t, we find the leaf node n of N that indexes v and add t to its list of tetrahedra. As such, each tetrahedron appears in at least one leaf node of N, and in at most four leaf nodes of N.

Finally, we exploit the spatial locality of N, by reindexing the vertex array V and the tetrahedra array T. We do this by generating



(a) Tetrahedral mesh (b) $k_v = 200$ (c) $k_v = 100$ (d) $k_v = 50$

Figure 4: PR-star octrees built over the F117 tetrahedral mesh (a) with varying bucket threshold values k_v (b-d).

a new sorted vertex array V' in which the vertices are ordered according to a depth-first traversal of the leaves of N. We then remap the vertex indices of T from those of V to those of V'. After this stage, the leaf nodes of N index a contiguous range of vertices from V' (which we will now refer to as V), the internal nodes of N index a contiguous range of vertices from V indexed by its descendants and the tetrahedra in T are consistently indexed according to this new ordering.

This enables a simpler representation for each leaf node of N. Besides the hierarchical information associated with the octree (e.g. pointers to the parent node and to the set of children nodes), each leaf node encodes: the range of vertex indices v_{start} and v_{end} in V, requiring two integers, and a pointer to a list of tetrahedra in T incident in these vertices.

5. EVALUATION OF STORAGE COSTS

We now discuss the storage costs of the PR-star octree representation and compare it to the cost of the extended IA data structure presented in Section 2.3. In our comparisons, we will assume that $|T| \approx 6|V|$, which is a common assumption in scientific data visualization since most of the data sets satisfy this relation.

Since the underlying data structure is the indexed tetrahedral mesh representation, which is required in all cases, we first analyze this fixed component. The geometry of the mesh is associated with the vertices of the mesh and requires three coordinates for each vertex for a total of $3|V|$ space. The tetrahedra in T are encoded through the *Tetrahedron-Vertex* boundary relation $R_{3,0}$ in terms of the indices of their vertices and require four vertex indices per tetrahedron for a total of $4|T|$ space. Thus, the total fixed cost of the indexed tetrahedral mesh representation is

$$4|T| + 3|V| \approx 27|V|. \quad (1)$$

Extended IA data structure.

In the extended IA data structure, the explicit topological information is encoded through the *Tetrahedron-Tetrahedra* relation $R_{3,3}$, requiring a total of $4|T|$ space, and through the partial Vertex-Tetrahedron relation $R^*_{0,3}$ requiring a total of $|V|$ space. Thus, the topological overhead of the extended IA data structure is

$$4|T| + |V| \approx 25|V|, \quad (2)$$

and its total storage requirements are

$$8|T| + 4|V| \approx 52|V|. \quad (3)$$

PR-star octree.

Let us analyze the data structure encoding the PR-star octree. Each node of the octree requires: three pointers for the octree hierarchy (i.e. one parent pointer, one pointer to a set of children and it is pointed to by one parent), two vertex indices, indicating the range of vertices indexed by the octree node, a pointer to a list of tetrahedra incident in these vertices and the number of tetrahedra in this list. This requires a total of $7|N|$ storage for the spatial indexing.

Recall that each tetrahedron in T is indexed by at least one, and by at most four, leaf nodes in N. To determine the total space required for the tetrahedron indices within the PR-star octree, we denote the average number of octree nodes in which a tetrahedron appears as χ, where $1 \leq \chi \leq 4$. Thus, the total storage for the lists of tetrahedra in N is $\chi|T|$. This gives a total topological overhead for the PR-star data structure of $\chi|T| + 7|N|$ space.

Based on our experiments, summarized in Table 1, we approximate $\chi \approx 2$. Also, since the decomposition is determined based on the vertex bucket threshold k_v we approximate the number of octree nodes as $|N| \approx |V|/k_v$. Since we typically set k_v to be greater than 7, $|N| < |V|$ (we approximate $|N| \approx |V|$ below). Thus, the topological overhead for the PR-star is

$$\chi|T| + 7|N| \approx 2|T| + (7/k_v)|V| \approx 13|V|, \quad (4)$$

and its total space requirement are

$$(4+\chi)|T| + 3|V| + 7|N| \approx 40|V|. \quad (5)$$

We observe that as k_v increases, the topological overhead of the PR-star data structure decreases, i.e. both $|N|$ and χ decrease as k_v increases. For example, when $k_v \approx 10$, the spatial component $7|N|$ requires |V|/1.4 space, while for $k_v \approx \{20, 50, 100, 200, 400, 600\}$, it requires $\{|V|/3, |V|/7, |V|/14, |V|/28, |V|/57, |V|/86\}$, respectively. Thus, in the worst case, when $\chi = 4$ and $k_v \geq 7$, the PR-star requires the same space as the extended IA, while in the best case, $\chi = 1$, the extended IA is larger than the PR-star octree by at least a factor of $3|T|$. On the other hand, as k_v increases, the cost of generating each local data structure (see Section 6) increases, so there is a tradeoff between the number of nodes $|N|$ and the size of each node (represented through parameter k_v).

Based on the estimates $|T| \approx 6|V|$ and $\chi \approx 2$, the PR-star octree incurs approximately half of the topological overhead as the IA ($13|V|$ vs. $25|V|$), and 80% of the total space of the extended IA ($40|V|$ vs. $52|V|$).

Empirically, for tetrahedral meshes of varying complexity, we found these estimates to favor the PR-star even more, as can be seen in Table 1. As expected, the benefits of the PR-star octree increase with the size of the mesh, and for increasing values of k_v.

Table 1: Storage costs for PR-star octree and extended IA representations in terms of the number of vertices $|V|$, the number of tetrahedra $|T|$, the vertex threshold k_v, the number of octree nodes (total $|N|$ and leaf $|N_{leaf}|$), and the average number of leaf nodes in which a tetrahedron is indexed χ. Storage costs in the final four columns are expressed as multiples of $|V|$.

Mesh	$\|V\|$	$\|T\|$	$\|T/V\|$	k_v	$\|N\|$	$\|N_{leaf}\|$	χ	Overhead		Total	
								IA	PR-star	IA	PR-star
F117	48.5 K	240 K	4.94	10	18.0 K	15.5 K	3.30	20.80	16.57	43.59	39.37
				20	9.27 K	8.05 K	2.95		14.66		37.45
				50	4.61 K	4.02 K	2.59		12.82		35.62
				100	2.18 K	1.90 K	2.20		10.91		33.71
POST	108 K	616 K	5.69	20	18.6 K	14.8 K	2.88	23.75	16.42	49.51	42.18
				50	7,58 K	6.22 K	2.48		14.13		39.89
				100	4.45 K	3.62 K	2.28		12.95		38.71
				200	2.27 K	1.93 K	2.09		11.90		37.66
CAMEL_MC	243 K	997 K	4.11	20	51.2 K	41.1 K	2.67	17.45	11.07	36.89	30.51
				50	23.9 K	19.6 K	2.22		9.13		28.57
				100	12.0 K	9.84 K	1.89		7.77		27.21
				200	6.44 K	5.35 K	1.67		6.86		26.31
FIGHTER_2	257 K	1.40 M	5.47	20	49.2 K	42.9 K	2.68	22.88	14.72	47.75	39.60
				50	20.5 K	17.9 K	2.25		12.31		37.19
				100	10.4 K	9.10 K	1.99		10.88		35.76
				200	5.40 K	4.71 K	1.78		9.73		34.61
F16_DENSITY	1.12 M	6.35 M	5.64	200	23.6 K	20.6 K	1.90	23.57	10.70	49.14	36.27
				400	12.6 K	11.0 K	1.71		9.68		35.25
				600	7.92 K	6.91 K	1.60		9.04		34.61
				800	5.94 K	5.18 K	1.54		8.71		34.28

6. GENERATING OPTIMAL LOCAL TOPOLOGICAL DATA STRUCTURES

In this section, we outline a general strategy for computing local topological data structures for the leaf nodes of a PR-star octree which are optimized based on the requirements of the specific application.

We locally process the tetrahedral mesh in a streaming manner by iterating through the leaf nodes of the octree N. For each leaf node n of N, we construct an application–dependent local data structure, which we use to process the local geometry. After we finish processing octree node n, we discard the local data structure and move on to the next node.

The basic ingredients in every topological data structure is the encoding of a suitable subset of the topological relations. Thus, we outline the construction process for a few common example applications.

For example, to build the local *Vertex-Tetrahedra* relation $R_{0,3}$ for the vertices V_n in a node n, we iterate through the vertices of the tetrahedra T_n in n. For each vertex v of a tetrahedron t indexed by n, we add the index of t in T to the list of tetrahedra in the local star of v (see Algorithm 1). Since the indexed vertices are contiguous and there are at most k_v vertices associated with leaf node n of N, our local data structure is an array of size k_v. Each position in the array corresponds to a vertex indexed by n and points to an (initially empty) list of indices from T.

To retrieve the local *Tetrahedron-Tetrahedra* adjacency relations $R_{3,3}$ for all the tetrahedra indexed by leaf node n, we iterate through the four triangular faces of each of its tetrahedra (see Algorithm 2).

Algorithm 1 GENERATEVERTEXSTAR(n)

Require: n is a leaf node in octree N
Require: V_n are the vertices indexed by n
Require: T_n are the tetrahedra indexed by n
Require: Index I_v of vertex $v \in V_n \rightarrow v_{start} \leq I_v < v_{end}$
Ensure: Relation $R_{0,3}$ reconstructed $\forall v \in V_n$
1: **for all** tetrahedra t in T_n (with index I_t in T) **do**
2: **for all** vertices v in t (with index I_v in V) **do**
3: **if** $v_{start} \leq I_v < v_{end}$ **then**
4: Add I_t to list of tetrahedra in star of v

For a manifold tetrahedral mesh in $\mathbb{R}^3$, as we consider here, a triangle can bound at most two tetrahedra. Faces with only a single incident tetrahedron lie on the outer shell of the indexed domain, or on the domain boundary.

If we are interested in the local edge-skeleton (i.e. the 1-skeleton formed by the edges of the mesh) or face-skeleton (i.e. 2-skeleton) of the mesh, we can easily reconstruct the *Edge-Edge* adjacencies or the *Triangle-Triangle* adjacencies using a similar approach.

We can also generate relevant statistics using similar algorithms, such as the number of vertices of a tetrahedron that are indexed by the current node; the number of tetrahedra in the star of a vertex or an edge; the number of triangles in the star of a vertex (or equivalently, the number of edges in its link).

All such algorithms have time complexity that is $O(|T_n|)$ or $O(|V_n|)$, where $|T_n|$ and $|V_n|$ are the number of tetrahedra or vertices indexed by the current octree node n, respectively.

Algorithm 2 GENERATEADJACENCIES(n)

Require: n is a leaf node in octree N
Require: V_n are the vertices indexed by n
Require: T_n are the tetrahedra indexed by n
Require: Index I_v of vertex $v \in V_n \rightarrow v_{start} \leq I_v < v_{end}$
Ensure: Relation $R_{3,3}$ is *locally* reconstructed $\forall t \in T_n$
(perfectly reconstructed $\forall t \in T_n$ with more than one vertex in n)
1: **for all** tetrahedra t in T_n (with index I_t in T) **do**
2: **for all** vertices v in t (with index I_v in V) **do**
3: Let f_v be the face of t that is opposite vertex v
4: Add I_t to the pair of tetrahedra for face f_v
5: **for all** faces f_v with tetrahedra pair $\{t_1, t_2\}$ for vertex v **do**
6: Set t_1 as adjacent tetrahedron for t_2 opposite vertex v
7: Set t_2 as adjacent tetrahedron for t_1 opposite vertex v

7. BOUNDARY DETERMINATION

Determining the elements on the boundary of a tetrahedral mesh is an important subtask in many applications, including finite element analysis and curvature estimations, since the tetrahedra on the boundary of the mesh require special processing. Additionally, when simplifying a mesh, it is important to preserve the features of the boundary as much as possible.

We describe here a simple local algorithm to differentiate the boundary vertices from the interior vertices. In a manifold mesh, the link of a vertex in the interior of the domain is homeomorphic to a sphere while that of a boundary vertex is not. Since the majority of the vertices are interior to the mesh, we prefer a data structure that classifies vertices in $O(1)$ time after the data structure is constructed, i.e. where we do not need to traverse the data structure's elements in a second pass.

We utilize an *Euler*-type counting scheme to classify the vertices as interior or boundary. For each vertex indexed by an octree node n, we keep track of the faces in its star (which correspond to the edges $\{e\}$ in its link), as well as the tetrahedra in its star (which correspond to the triangles $\{t\}$ in its link). Since we assume that the mesh is manifold, a vertex v is on the domain boundary when $2|e| \neq 3|t|$. Otherwise, it is in the interior of the domain. The motivation is that if vertex v is in the interior of the domain, its triangulated link is homeomorphic to a triangulation of the sphere and thus all the edges in the link are shared by exactly two triangles.

Thus, constructing the data structure is in $O(|T_n|)$ and boundary determination requires $O(|V_n|)$ time per leaf node. When applied to the entire mesh, the algorithm requires $O(\chi|T|+|V|)$ time, where χ is the average number of nodes in which a tetrahedron appears. The former terms comes from the construction phase, while the latter term comes from the boundary determination. Since the storage space is related to the number of tetrahedra indexed by a node, the storage requirements for this algorithm are $O(|T_n|)$ for each node, which can be estimated as $O(k_v)$.

This boundary determination can be locally reconstructed whenever the application requires it, or the results can be stored globally. In the latter case, we can store these in a separate bit array corresponding to V, and requiring an additional $|V|$ bits. Alternatively, we can store the information in place in the tetrahedral array T, using a convention that a negative index for a vertex in T indicates that the corresponding vertex is on the domain boundary.

8. DISCRETE CURVATURE ESTIMATION

In the case of terrain datasets, the elevation values for 2D samples provide an embedding of the triangle mesh discretizing the domain where the samples are distributed into 3D space, thus giving rise to the Triangulated Irregular Network (TIN) model. Volume datasets provide scalar field values at a set of points in 3D space, which are triangulated through a tetrahedral mesh with vertices at these points. The scalar values sampled at the 3D vertices of a tetrahedral mesh describe the embedding of the tetrahedral mesh in 4D space in a similar manner as we have a TIN in 3D space for 2D data. Curvature plays an important role in the morphological analysis of terrains [14], and its 3D counterpart [15] is crucial in understanding the complexity of a volumetric scalar field through its embedding in the 4D space. In the discrete setting, the curvature of a scalar field is typically computed at the vertices of the mesh and is interpolated across the rest of the domain. Such computations involve the geometry in the *1-ring* of a vertex, i.e. the star of a vertex.

In this section, we consider the computation of *discrete distortion*, a recent generalization of Aleksandrov's concentrated curvature [1] to tetrahedral shapes embedded in 4D space [15]. The distortion at a vertex in a tetrahedral mesh embedded in 4D is the difference between the sum of the solid angles in the elevated 4D space from that of its solid angles in the (flat) 3D domain. As such, to calculate the discrete distortion at a vertex, we need to iterate on the tetrahedra in its star, and to know whether the vertex is on the domain boundary (in which case, the sum of solid angles in the 3D domain is 2π, while it is 4π for interior vertices, see [15] for details).

Note that when computing discrete curvature over the entire dataset, we do not strictly need the topological connectivity, since we must process every single tetrahedra in any event. However, many applications require local curvature estimation of spatially coherent vertices. For example, in [8], the vertices in the 1-ring of an edge are checked before and after a remeshing operation to locally optimize a triangle mesh.

In Table 2, we compare the computation time for extracting the *Vertex-Tetrahedra* (VT) relation for all vertices in the mesh represented as a PR-star octree or as an extended IA data structure, as well as computation times for generating the distortion values from both data structures. In the table, we specify absolute timings as well as relative timings. We also specify the different values of the bucket capacity k_v (for the PR-star) on which we have experimented.

Comparing the timings obtained by using the two data structure for VT extraction on all vertices, we notice that extraction of the VT relation from the PR-star is always faster than its extraction from the extended IA data structure. The time requirement is between 50% (for smaller meshes) to 30% (for larger meshes) of that required for the extended IA data structure. This is due to our use of an optimized local data structure that explicitly calculates the VT relation and that we visit each tetrahedron an average of χ times (i.e. the number of leaves which contain it). This is typically much lower than the four times a tetrahedron is visited in the extended IA data structure.

In contrast to the simple operation of building the star, the distortion computation requires complicated geometric processing. For example, we need to calculate the trihedral angle of the three faces of a tetrahedron incident to the given vertex. By considering this computation, in addition to the cost of traversing the octree, the PR-star requires 30% to 50% more time than the extended IA for distortion computation on smaller meshes. For medium sized meshes, the timings for the two structure are almost equivalent and for larger meshes, the PR-star is approximately 20% faster that the IA. Moreover, since all computations are done locally on the PR-star octree, it requires significantly less memory and resources as the the mesh size increases.

Table 2: Timing comparisons (TIME column represents seconds, while (%) column shows relative times) for extracting the *Vertex-Tetrahedra (VT)* relation and for computing the distortion from the extended IA and the PR-star octree for various vertex threshold values (k_v) on several tetrahedral meshes.

Mesh	Structure	k_v	VT		Distortion	
			Time	(%)	Time	(%)
F117	IA	–	0.12	–	0.27	–
	PR-star	10	0.07	**55**	0.35	**131**
		20	0.06	**51**	0.35	**129**
		50	0.06	**46**	0.34	**127**
		100	0.05	**41**	0.38	**125**
POST	IA	–	0.23	–	0.57	–
	PR-star	20	0.15	**65**	0.87	**151**
		50	0.13	**59**	0.86	**149**
		100	0.13	**56**	0.85	**148**
		200	0.12	**53**	0.85	**147**
CAMEL_MC	IA	–	0.94	–	2.65	–
	PR-star	20	0.32	**34**	2.51	**95**
		50	0.29	**31**	2.51	**94**
		100	0.27	**28**	2.47	**93**
		200	0.25	**27**	2.45	**92**
FIGHTER_2	IA	–	1.16	–	2.12	–
	PR-star	20	0.46	**39**	2.12	**100**
		50	0.42	**36**	2.08	**98**
		100	0.39	**34**	2.05	**97**
		200	0.37	**32**	2.03	**96**
F16_DENSITY	IA	–	6.25	–	11.10	–
	PR-star	200	1.77	**28**	9.37	**84**
		400	1.68	**27**	9.28	**84**
		600	1.63	**26**	9.21	**83**
		800	1.60	**26**	9.18	**83**

9. MESH SIMPLIFICATION

One of the major difficulties in processing tetrahedral meshes is the large size of such meshes. In many situations, the meshes are generated by a uniform process that is not adapted to the underlying properties of the shape, and thus, the mesh is oversampled. Thus, a typical preprocessing step for many downstream applications is to simplify the mesh by reducing the number of its vertices and tetrahedra. A common simplification approach is to iteratively apply the *edge collapse operator* to the mesh elements. This consists of contracting an edge of the mesh to a vertex. In the *half-edge collapse* operation [3], an edge $e = (v_1, v_2)$ is collapsed to one its extreme vertices, e.g. v_1.

In this section, we show how to implement the half-edge collapse operator in the PR-star data structure. Half-edge collapse for an edge $e := \{v_1, v_2\}$ is a local operator, whose implementation requires an efficient retrieval of the star of the two vertices v_1 and v_2 and the star of edge e. For any edge whose extreme vertices are both indexed by an octree node, all of this information (star of both vertices) is available. Our simplification algorithm incrementally simplifies the local tetrahedral mesh within each octree node by successively collapsing edges within the node until a quality threshold is achieved. In our current implementation, we consider a simple geometric quality of the edge (e.g. its length), and collapse from the vertex with the higher vertex index into the vertex with the lower vertex index. More advanced quality criteria, such as a priority based ordering of edge-collapses that incorporate higher-order geometry, such as curvature or quadric error [10] should be easy to incorporate.

Our algorithm imposes some restrictions on edge collapses [22]. The *boundary condition* ensures that the domain boundary is preserved as much as possible. We collapse only edges whose extreme vertices are both on the interior of the mesh, or both on its boundary. The *geometric condition* ensures that the orientation of its tetrahedra are maintained during collapses. That is, we prevent tetrahedra from flipping, since this would create invalid tetrahedra. Finally, the *topological condition* (also referred to as the *link condition*) ensures that the topological type of the mesh is preserved. In our case, this ensures that non-manifold situations cannot arise in the dataset.

The half-edge collapse of edge $e := (v_1, v_2)$ into vertex v_1 is performed as follows:

- Delete all the tetrahedra in $St(e)$, the star of edge e.
- Add the tetrahedra from $St(v_2)$ to $St(v_1)$, and remove the tetrahedra in the star of edge e:
$$St(v_1) = St(v_1) \cup St(v_2) - St(e).$$
- Modify the geometry of the vertices in $St(v_2)$, by replacing v_2 with v_1.
- Clear the memory for $St(v_2)$ and delete vertex v_2.

Our simplification algorithm on the PR-star octree performs a few iterative stages. We first apply local simplifications to each octree node and update the spatial index. Next, we compact the tetrahedral mesh, and rebuild the spatial index. These three stages are repeated until we remove at least an edge during the local simplification.

The local simplification algorithm operates on each leaf node n in the octree N. We insert into a local queue Q all edges $e := (v_1, v_2)$ such that v_1 and v_2 are both indexed by octree node n and e is shorter than a predetermined threshold. For each edge in queue Q, the simplification criteria listed above (depending on boundary, geometry and topology) are checked, and if these are satisfied, edge e is simplified.

To determine the effect of the local edge collapse algorithm with respect to the same algorithm on the entire dataset, we compare the performance on a PR-star octree with practical values of k_v with that of a PR-star with k_v set to infinity. In the latter case, the octree N will have only a single leaf node (i.e. the root of N). In Table 3, we compare the number of tetrahedra removed, the computation times and the memory requirements to complete the simplification procedure. As a proxy for the memory requirements, we consider the maximum number of edges in the simplification queue Q.

We have conducted a series of experiments in which the stopping criterion is based on the approximate percentage of tetrahedra removed relative to the original mesh (e.g. 25%, 50% and 75%). Comparing the results, we observe that for small meshes with relatively low values of k_v, many edges are contained in two octree nodes and are therefore not eligible for simplification. For example, on the F117 dataset with $k_v = 20$, we were only able to remove 65% of the tetrahedra compared to the global data structure ($k_v = \infty$). However, for larger meshes and larger values of k_v, this was less of a problem, and only 10-20% of tetrahedra were blocked by our local algorithm.

Considering both timings and the percentage of tetrahedra removed we can see that practical PR-star octrees perform worst when the mesh is relatively small and we want to remove relatively few tetrahedra. In this case, there are more tetrahedra shared between multiple nodes, and the cost of building the index is more significant

Table 3: Comparison of timings (in seconds), the number of removed tetrahedra and memory requirements for our local half-edge simplification for PR-star octree with different vertex bucket thresholds. Comparisons are to the PR-star octree consisting of a single octree node ($k_v = \infty$). The memory column considers the maximum size of the simplification edge queue as a proxy for the number of bytes.

		Approximately 25% of tetrahedra removed					Approximately 50% of tetrahedra removed					Approximately 75% of tetrahedra removed				
mesh	k_v	**removed**	(%)	**timings**	(%)	**memory**	**removed**	(%)	**timings**	(%)	**memory**	**removed**	(%)	**timings**	(%)	**memory**
F117	∞	68.7 K	–	0.59	–	64.7 K	119 K	–	1.06	–	262 K	175 K	–	1.91	–	747 K
	10	34.7 K	**51**	0.91	**154**	74	60.4 K	**51**	1.05	**98**	106	81.5 K	**47**	1.65	**86**	128
	20	44.9 K	**65**	0.90	**151**	133	78.2 K	**66**	1.24	**116**	268	112 K	**64**	2.05	**107**	390
	50	57.6 K	**84**	1.03	**174**	316	101 K	**85**	1.20	**113**	508	145 K	**83**	2.59	**136**	915
	100	59.9 K	**87**	0.89	**150**	452	106 K	**89**	1.43	**134**	1.2 K	159 K	**91**	2.47	**129**	2.3 K
POST	∞	159 K	–	0.93	–	94.4K	330.8K	–	1.94	–	430 K	449 K	–	4.48	–	1.34 M
	20	151 K	**95**	1.61	**173**	210	315 K	**95**	1.96	**101**	286	435 K	**97**	2.99	**67**	388
	50	156 K	**98**	1.26	**136**	456	325 K	**98**	2.19	**113**	791	443 K	**99**	3.70	**83**	929
	100	157 K	**98**	1.27	**136**	625	326 K	**99**	2.09	**107**	1.4K	450 K	**100**	4.12	**93**	1.9 K
	200	157 K	**99**	1.24	**133**	1.2 K	332 K	**100**	2.65	**136**	3.1K	451 K	**101**	4.49	**100**	4.5 K
CAMEL_MC	∞	291 K	–	14.59	–	1.49M	539.8K	–	25.64	–	3.33 M	746 K	–	37.96	–	5.39 M
	20	237 K	**82**	7.94	**54**	557	422 K	**78**	12.93	**50**	566	551 K	**74**	15.20	**40**	569
	50	263 K	**91**	7.94	**55**	1.6 K	474 K	**88**	10.94	**43**	1.8 K	642 K	**86**	15.16	**40**	1.8 K
	100	274 K	**94**	8.58	**59**	3.2 K	497 K	**92**	12.24	**48**	4.4 K	674 K	**90**	16.06	**42**	3.6 K
	200	284 K	**97**	7.66	**52**	6.8 K	515 K	**95**	12.03	**47**	6.9 K	704 K	**94**	16.93	**45**	8.1 K
FIGHTER_2	∞	362 K	–	6.76	–	860.1K	821 K	–	14.19	–	3.17 M	1.15 M	–	20.80	–	6.35 M
	20	265 K	**73**	8.96	**132**	313	633 K	**77**	12.68	**89**	618	899 K	**78**	17.21	**83**	618
	50	295 K	**82**	8.04	**119**	1 K	684 K	**83**	12.72	**90**	1.3 K	997 K	**86**	19.92	**96**	1.5 K
	100	314 K	**87**	7.06	**104**	1.9 K	733 K	**89**	12.46	**88**	2.4 K	1.04 M	**90**	21.69	**104**	2.8 K
	200	317 K	**88**	7.07	**105**	2.9 K	753 K	**92**	14.19	**100**	5.6 K	1.08 M	**93**	22.94	**110**	6 K
F16_DENSITY	∞	1.61 M	–	43.13	–	4.05M	3.24M	–	71.86	–	8.86 M	4.94 M	–	134.62	–	21.0 M
	200	1.40 M	**87**	45.62	**106**	5.6 K	2.93 M	**91**	56.92	**79**	5.7 K	4.50 M	**91**	89.42	**66**	6.2 K
	400	1.48 M	**92**	46.75	**108**	9.5 K	2.97 M	**92**	61.98	**86**	10.4 K	4.64 M	**94**	93.72	**67**	12.9 K
	600	1.50 M	**93**	41.92	**97**	16.6 K	2.99 M	**92**	58.88	**82**	18.3 K	4.67 M	**94**	94.97	**71**	19.5 K
	800	1.50 M	**93**	36.84	**85**	21.3 K	3.04 M	**94**	55.43	**77**	23.8 K	4.69 M	**95**	93.57	**70**	24 K

relative to the simplification algorithm. Performance improves significantly for larger meshes, where the auxiliary data structures (the queues) are significantly smaller and their generation costs have a much smaller effect.

In summary, the experiments show that using around 0.1% the memory, and restricting edges to belong entirely to the current octree node we are able to remove most of the same edges (i.e. $\approx$ 80–90%) in similar or less time.

In Figures 5 and 6 we show some details of two simplified meshes obtained using our half edge collapse procedure. The images show the original mesh and the simplified meshes with 75%, 50% and 25% of the tetrahedra.

10. CONCLUDING REMARKS

We have presented the *PR-star octree* as a novel spatial data structure optimized for performing efficient topological queries on large tetrahedral meshes. Each leaf node of the PR-star octree encodes the minimal amount of information necessary to locally reconstruct the topological connectivity of its indexed elements. We have demonstrated the advantages of the PR-star octree representation in several applications, including detection of the domain boundaries, computation of local curvature estimates and mesh simplification.

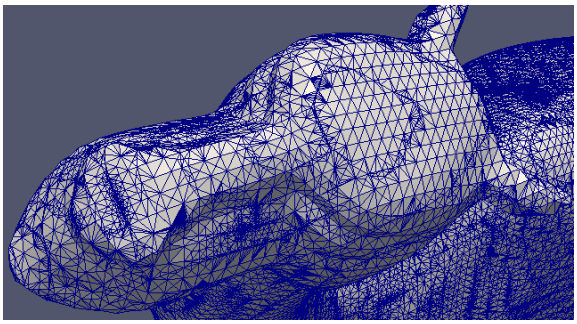
In contrast to previous spatial indexing approaches, the PR-star is not optimized for spatial queries such as point location. Rather, it is optimized for spatially coherent topological queries on the dataset. In contrast to topological data structures on tetrahedral meshes, we are not forced into a choice of an initial data structure (possibly optimized for a specific task) that must be used for all future applications. The spatial indexing enables the construction of the locally optimal data structures for the given application. Due to the spatial locality of successive queries in typical GIS applications, the construction costs of these optimal local data structures at runtime are amortized over multiple accesses while processing each node. The experimental results clearly indicate the benefits of the PR-star octree data structure for large tetrahedral meshes. As the mesh size increases, the memory requirements of our data structure decrease with respect to the an optimal data structure for the given task and with respect to general state of the art topological data structures.

Although our reference implementation uses a pointer-based octree, it would not be difficult to switch to a linear octree representation [9] (i.e. to remove the internal nodes). Since the number of internal nodes is about $1/8$ the number of leaf nodes, and the overhead for the number of octree nodes scales with the inverse of the bucket threshold k_v, this change will not likely save us that much space (see Table 1), but may save us time in traversing the octree during operations.

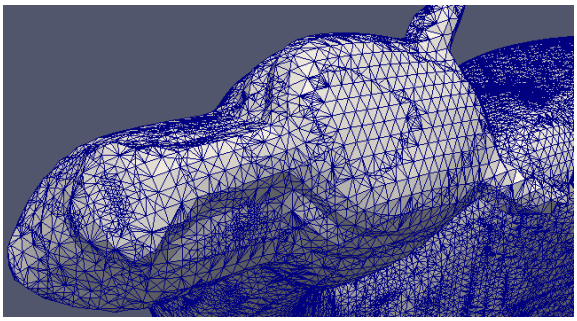
In our future work we are planning to investigate ways of tuning the value of the bucket capacity k_v to better fit the dataset. Additionally, algorithms that cache the expanded local data structures for adjacent octree nodes can help us process geometry spanning multiple octree nodes. For example, this would help us simplify edges whose vertices are located in separate nodes. This will enable the removal of the same edges as the single-node PR-star octree while retaining the memory benefits associated with PR-star octrees with higher values of k_v.

Acknowledgments

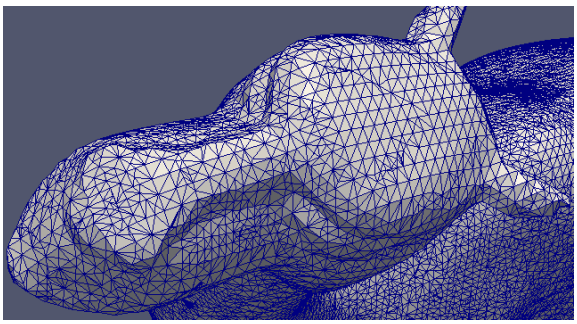
We would like to thank Paola Magillo for many helpful discussions and the anonymous reviewers for their helpful comments and suggestions. This work has been partially supported by the National



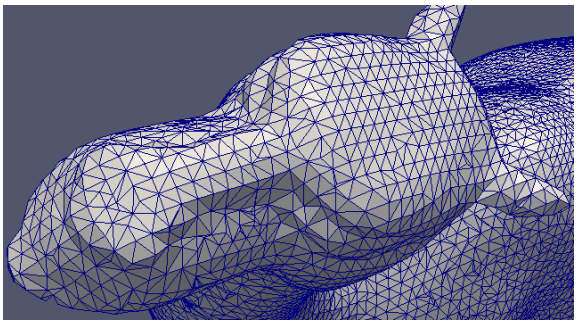
(a) Original mesh



(b) 25% of tetrahedra removed

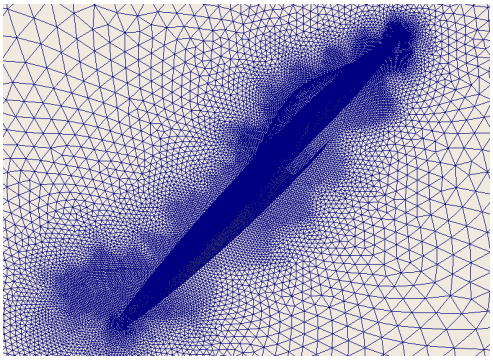


(c) 50% of tetrahedra removed

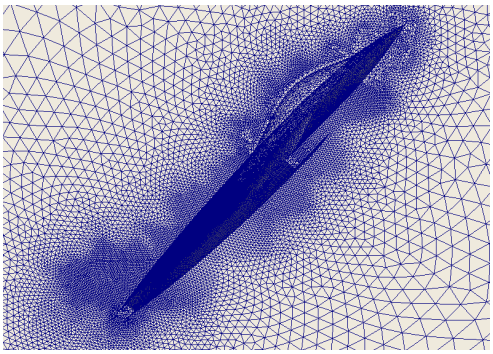


(d) 75% of tetrahedra removed

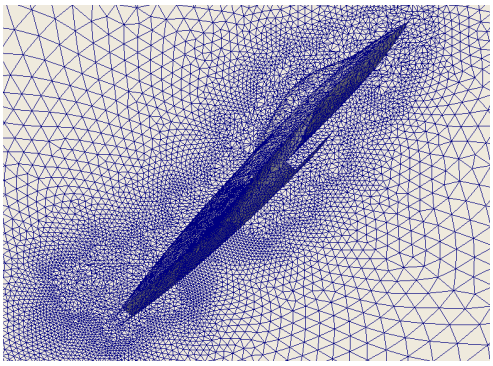
Figure 5: Simplified versions of CAMEL_MC tetrahedral mesh after applying our local edge-collapse procedure. Original mesh (a) and mesh after removing 25% (b) 50% (c) and 75% (d) of the tetrahedra.



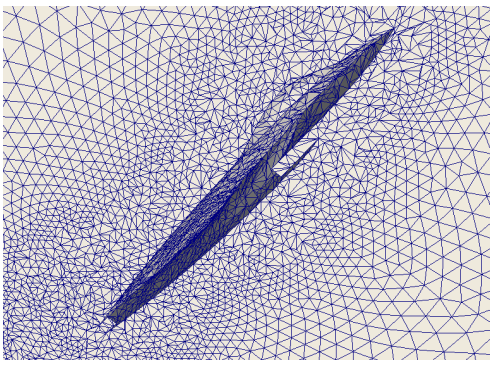
(a) Original mesh



(b) 25% of tetrahedra removed



(c) 50% of tetrahedra removed



(d) 75% of tetrahedra removed

Figure 6: Simplified versions of FIGHTER_2 tetrahedral mesh after applying our local edge-collapse procedure. Original mesh (a) and mesh after removing 25% (b) 50% (c) and 75% (d) of the tetrahedra.

Science Foundation under grant number IIS-1116747, and by the Italian Ministry of Education and Research under the PRIN 2009 program.

Volumetric datasets are courtesy of the AIM@SHAPE repository (CAMEL_MC and POST), the volvis repository (F16_DENSITY), C. Silva (FIGHTER_2) and R. Haimes (F117).

11. REFERENCES

[1] P. Aleksandrov. *Topologia Combinatoria*. Torino, 1957.

[2] I. Carlbom, I. Chakravarty, and D. Vanderschel. A hierarchical data structure for representing the spatial decomposition of 3D objects. *IEEE Computer Graphics and Applications*, 5(4):24–31, 1985.

[3] P. Cignoni, L. De Floriani, P. Magillo, E. Puppo, and R. Scopigno. Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):29–45, January-February 2004.

[4] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, 2003.

[5] L. De Floriani, M. Facinoli, P. Magillo, and B. Dimitri. A hierarchical spatial index for triangulated surfaces. In *Int. Conf. on Computer Graphics Theory and Applications (GRAPP)*, pages 86–91, 2008.

[6] L. De Floriani, R. Fellegara, and P. Magillo. Spatial indexing on tetrahedral meshes. In D. Agrawal, P. Zhang, A. El Abbadi, and M. Mokbel, editors, *Proceedings ACM SIGSPATIAL GIS*, GIS '10, pages 506–509, New York, NY, USA, 2010. ACM.

[7] T. Dey, J. Levine, and A. Slatton. Localized delaunay refinement for sampling and meshing. *Computer Graphics Forum*, 29(5):1723–1732, 2010.

[8] N. Dyn, K. Hormann, S.-J. Kim, and D. Levin. Optimizing 3D triangulations using discrete curvature analysis. In T. Lyche and L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces: Oslo 2000*, Innovations in Applied Mathematics, pages 135–146. Vanderbilt University Press, Nashville, TN, 2001.

[9] I. Gargantini. Linear octtrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, 20(4):365–374, 1982.

[10] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings ACM SIGGRAPH*, pages 209–216, 1997.

[11] T. Gurung and J. Rossignac. SOT: A compact representation for tetrahedral meshes. In *Proceedings SIAM/ACM Geometric and Physical Modeling*, SPM '09, pages 79–88, San Francisco, USA, 2009.

[12] G. Hjaltason and H. Samet. Speeding up construction of quadtrees for spatial indexing. *VLDB*, 11(2):109–137, 2002.

[13] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, June 1982.

[14] M. Mesmoudi, L. De Floriani, and P. Magillo. Morphological analysis of terrains based on discrete curvature and distortion. In W. Aref, M. Mokbel, H. Samet, M. Schneider, C. Shahabi, and O. Wolfson, editors, *Proceedings ACM SIGSPATIAL GIS*, pages 415–418, Irvine, CA, USA, 2008.

[15] M. Mesmoudi, L. De Floriani, and U. Port. Discrete distortion in triangulated 3-manifolds. *Computer Graphics Forum*, 27(5):1333–1340, 2008.

[16] I. Navazo. Extended octree representation of general solids with plane faces: Model structure and algorithms. *Computer & Graphics*, 13(1):5–16, 1989.

[17] R. Nelson and H. Samet. A population analysis for hierarchical data structures. In *Proc. ACM SIGMOD Conference*, pages 270–277, 1987.

[18] G. M. Nielson. Tools for triangulations and tetrahedralizations and constructing functions defined over them. In G. Nielson, H. Hagen, and H. Müller, editors, *Scientific Visualization: Overviews, Methodologies and Techniques*, chapter 20, pages 429–525. IEEE Computer Society, 1997.

[19] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(1):56–102, January 1993.

[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 2006.

[21] H. Samet and R. Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics*, 4(3):182–222, 1985.

[22] O. G. Staadt and M. Gross. Progressive tetrahedralizations. In *Proceedings IEEE Visualization*, pages 397–402, Research Triangle Park, NC, 1998. IEEE Computer Society.