Relative Performance of Geometric Search Algorithms for Interpolating Unstructured Mesh Data

Mahdieh Khoshniat[1,2], Gordan R. Stuhne[1], and David A. Steinman[1,2]

[1]Imaging Research Labs, Robarts Research Institute
{mkhosh, gstuhne, steinman}@imaging.robarts.ca
and
[2]Biomedical Engineering Program, The University of Western Ontario, London, Ontario, Canada

Abstract. Interpolation of field data from unstructured meshes requires the potentially expensive identification of the finite element or volume within which the interpolating point lies. A number of geometric search algorithms have been proposed to reduce this expense at the cost of setting up and storing additional search tables. Using tetrahedral finite element models we show that a structured auxiliary mesh (SAM) algorithm can achieve search speeds well in excess of 100,000 points/sec—at least an order of magnitude better than digital tree or nearest neighbour searches—with only modest setup times and storage requirements. Our novel SAM variant is found to offer the best performance per unit of storage, but at the expense of considerable setup times. We conclude that SAM algorithms can be used to provide a flexible "software-only" approach for real-time interpolation and visualization of unstructured mesh data.

1 Introduction

Unstructured meshes are an unavoidable fact of life in the engineering analysis of complex domains and the visualization of the resulting field data. An unstructured mesh is ordinarily defined by a list of nodal coordinates at which the field data is prescribed and/or solved; a table defining how these nodes are connected to form the underlying finite elements or volumes; and the shape functions that define how, for each element, the field data vary between nodes. Interpolation of such piecewise continuous data therefore requires identification of the element or volume within which the interpolating point lies, followed by interpolation using the connected node data and known shape functions. The latter is straightforward, whereas the former—a process often referred to as "geometric searching"—can require fairly sophisticated algorithms in order to be performed in a computationally efficient manner.

To illustrate this, consider a naïve approach in which, for each interpolating point, each and every element is searched. While algorithmically simple, the computational effort obviously scales with the number of elements in the mesh, which in the case of 3D models renders this approach prohibitively expensive. Instead, more sophisticated algorithms have been proposed to reduce the number of elements queried, at the expense of creating and storing additional search tables. Below we review a number of geometric search strategies, including one proposed here for the first time, after which we assess their relative performance for finding random points within non-trivial tetrahedral finite element meshes.

R.E. Ellis and T.M. Peters (Eds.): MICCAI 2003, LNCS 2879, pp. 391–398, 2003.

2 Geometric Search Strategies

2.1 Nearest Neighbour (NN)

In this approach, first suggested by Bercovier [1], the node nearest the interpolating point is first identified, and then only elements surrounding that node are searched. It is easy to see how this can substantially reduce the number of element queries; however, this savings is achieved at the not insubstantial cost of calculating distances from the interpolating point to each node in the mesh. Moreover, the nearest node will not necessarily be connected to the element containing the interpolating point. Thus, it is usually necessary to query a least one extra "layer" of elements surrounding the nearest node to ensure robustness. In addition to its obvious simplicity, an advantage of nearest neighbour searches is the negligible setup and storage requirements: the search table simply consists of a vector of nodes pointing to another vector of connected elements.

2.2 Alternating Digital Tree (ADT)

Digital trees are recursive data structures commonly used for searching and sorting operations. They are constructed by defining a root, and assigning an element to one of two branches based upon whether the bounding box of that element satisfies some geometric condition. By following this procedure for all elements in a list, a tree is built up in such a way that, when searching for the location of an element, each search step ideally reduces the number of elements (M) to be checked by a factor of two, resulting in search times that can scale with O(log M). Digital tree algorithms are particularly attractive for unstructured mesh generation, in that setup times and tree storage requirements are modest—proportional to the number of elements—and thus trees can be easily modified as new elements are introduced. A popular variant is the alternating digital tree (ADT) algorithm [2], which provides well-defined branch criteria for the general case of N-dimensional meshes.

2.3 Structured Auxiliary Mesh (SAM)

First demonstrated by Pissanetzky and Basombrio [3], structured auxiliary meshes (SAMs) are regular Cartesian grids superimposed onto the unstructured domain. For each SAM element, hereafter referred to as a voxel, the overlapping unstructured elements are tabulated in a preprocessing step. (As implemented by these authors, it is the *bounding boxes* of the unstructured elements that are actually used to identify overlap.) Queries are then performed via the trivial identification of the voxel within which the interpolating point falls, followed by testing only those overlapping elements. It is not difficult to see that the relative efficiency of this approach scales with the density of the grid: finer voxels require fewer element queries, but at the price of longer setup times and increased storage requirements, as the search table now consists of a vector of *voxels* pointing to another vector of *overlapping elements*.

2.4 Structured Auxiliary Mesh with Element Storage (SAMe)

As noted above, the original SAM algorithm simplified the setup process by identifying those unstructured elements whose *bounding boxes* intersected a given voxel. In the case of, say, thin elements oriented obliquely to the cardinal axes, it is not hard to see that the number of voxels intersecting that element will be overestimated, resulting in an increase in storage requirements and the number of element queries. To obviate this we propose here a novel and simple extension of this original SAM algorithm, in which the *actual element boundaries* rather than the element bounding boxes are used to determine voxel intersections. This serves to reduce the storage requirements—only truly overlapping elements are tabulated—but at increased setup times owing to the cost of testing for the intersection of two 3D objects (ie, a cube and a tetrahedron). To moderate this computational overhead, we project each element onto the three cardinal planes, in which case 3D intersection is true only when the (triangular or quadrilateral) tetrahedral projections intersect with the projected voxel in all three directions. As illustrated in Fig. 1, the useful corollary of this is that non-intersection for one 2D projection implies non-intersection in 3D. To continue this idea of *progressively* excluding cases of intersection, for each projection we ask, algorithmically, the following questions:

Fig. 1. Non-intersection of at least one 2D projection is sufficient to exclude 3D intersection.

1. Do any of the element's projected nodes fall within the projected voxel?
2. Do any of the projected voxel's corners fall within the projected element?
3. Do any of the projected element's edges intersect the projected voxel's edges?

As soon as one of these is answered positively for a given projection, we can move directly to the next projection. Conversely, if all answers are "no" for a given projection, then no 3D intersection has occurred.

3 Methods

The models used to test the performance of the geometric search algorithms described above were taken from computational fluid dynamics (CFD) models of arterial blood flow, the real-time and interactive visualization of which ultimately motivates this work. Specifically, two different models were considered: a relatively coarse half-symmetric carotid bifurcation model composed of 83,120 linear tetrahedral elements and 17,470 nodes (hereafter referred to as model 83k, and shown in Fig. 2a); and an anatomically realistic aneurysm model composed of 1,704,392 tetrahedral elements and 311,507 nodes (hereafter referred to as model 1.7M, and shown in Fig. 2b).

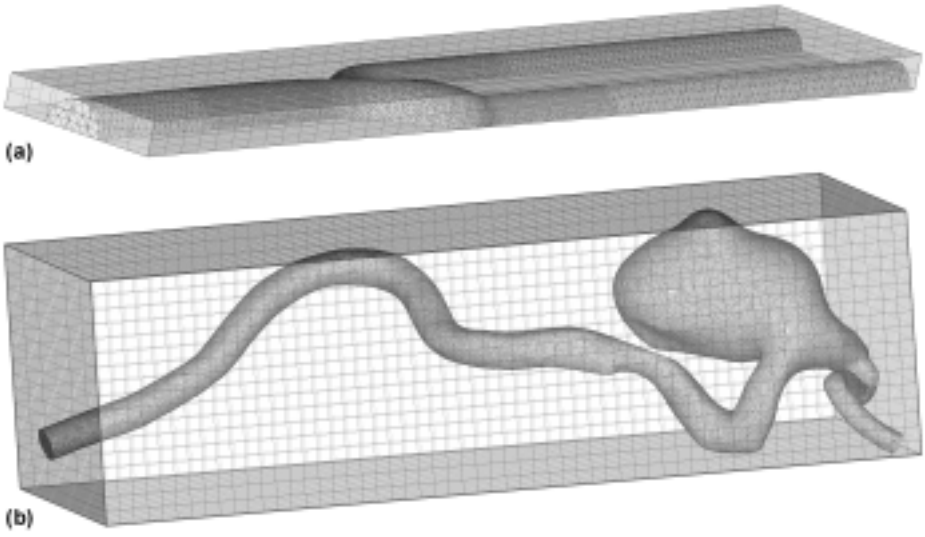

Fig. 2. Tetrahedral finite element meshes used to test geometric search performance, shown with SAMs having relative voxel volumes of 1000: (a) 83k element idealized carotid bifurcation mesh; (b) 1.7M element realistic aneurysm mesh.

Performance was evaluated via the following quantities. **Speed** was defined as the number of interpolating points identified per CPU-second. **Setup time** was defined as the CPU time required for creating the search tables, while **storage** was defined as the RAM required to store the search tables. Speed was calculated by searching for the locations of 1,000,000 points seeded randomly throughout the SAM (ie, mesh) bounding box and then, separately, only within the unstructured mesh itself. For both SAM algorithms the size of the nominally cubic voxels was defined relative to the mean unstructured element volume, i.e., $V_{voxel} = KV_{elem}$, where relative voxel volumes ranging from K=1000 to K=0.01 were tested within the limits of available memory.

All tests were performed on a 1.8 GHz AMD Athlon workstation running Red Hat Linux version 7.3. All code was written in C++, and compiled using gcc version 2.96 with optimization level 3. CPU times for speed and setup were determined using the gprof code profiling utility.

4 Results

The relative performance of the geometric search algorithms is summarized in Fig. 3. For all but the largest voxels tested the SAM algorithms achieved the fastest search speeds, typically more than 100,000 points/sec. With decreasing voxel volume our SAMe algorithm outperformed the conventional SAM algorithm by up to 100%, achieving speeds of more than 1,000,000 points/sec for the finest voxels tested. Both SAM algorithms outperformed the ADT algorithm, which achieved speeds on the order of 10,000 points/sec. The poorest performance was noted for the NN algorithm, which achieved speeds of less than 1000 points/sec.

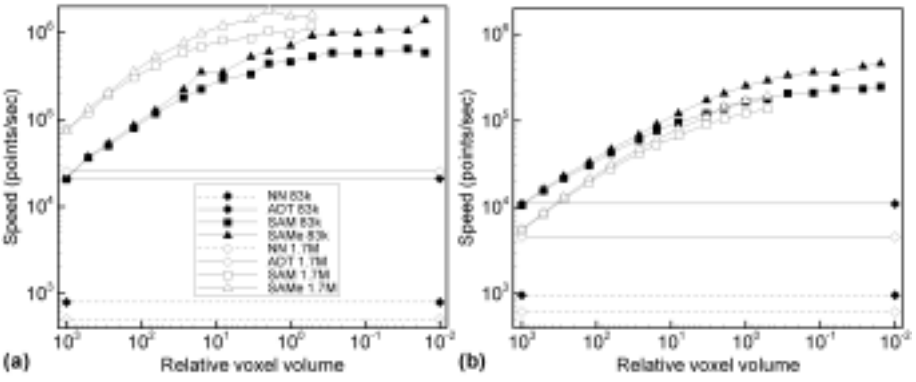

Fig. 3. Search speeds for points randomly seeded within: (a) the mesh bounding box; and (b) the mesh only.

Fig. 3a alone suggests that the ADT algorithm performed similarly for both the coarse and fine meshes, while the SAM and SAMe algorithms were actually faster for the fine mesh. These counter-intuitive observations can, however, be explained by noting that, for the 83k model, the volume of the bounding box was roughly 4× that of the mesh itself, whereas for the 1.7M model the bounding box volume was more than 25× the mesh volume. As a result, the speeds reported in Fig. 3a were effectively biased by the inclusion of many more empty voxels, and hence faster searches on average, for the 1.7M model. As shown in Fig. 3b, in the absence of these empty voxels both the SAM and SAMe algorithms achieved comparable speeds for both models. Similarly for the ADT algorithm search speeds were reduced proportionally according to the amount of "empty space" within the bounding box. Conversely, the NN algorithm saw little change in the search speeds when confined to searching for points within the mesh; this was expected, since NN algorithms must check all nearest neighbours whether points are inside or outside the mesh.

Fig. 4 confirms that increases in search speed are obtained at the cost of increased storage and/or setup times. For the NN and ADT algorithms these were modest: only a few seconds were required to create—and less than 30 Mb to store—the search tables even for the 1.7M element mesh. Storage requirements for the SAM algorithms were comparable to those for the NN and ADT algorithms for large voxel volumes, but increased exponentially with decreasing voxel volume. Our SAMe algorithm required less storage than the SAM algorithm, with savings up to 50% for the finest voxels. These savings, which parallel the increases in speed reported above for the SAMe algorithm, are achieved by storing, and hence searching, only those elements actually intersecting a given voxel. However, as Fig. 4b clearly shows, these advantages were achieved at the substantial expense of setup times on the order of a few minutes versus the few seconds required for the NN, ADT, and SAM algorithms.

Finally, Fig. 5a shows that the SAMe algorithm achieves the best speed:storage ratios; however, for both SAM and SAMe algorithms the optimal ratio occurs for relative voxel volumes ≈10, which, as Fig. 5b shows, correspond to storage requirements on the order of the respective connectivity table. Beyond this "optimal" voxel size, speed increases are achieved with increasingly heavier storage penalties.

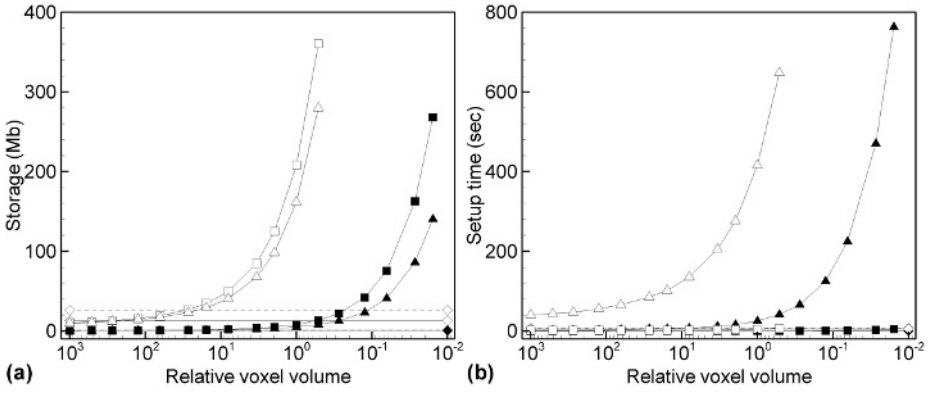

Fig. 4. (a) Storage requirements and (b) setup times for search tables. See Fig. 3 for legend.

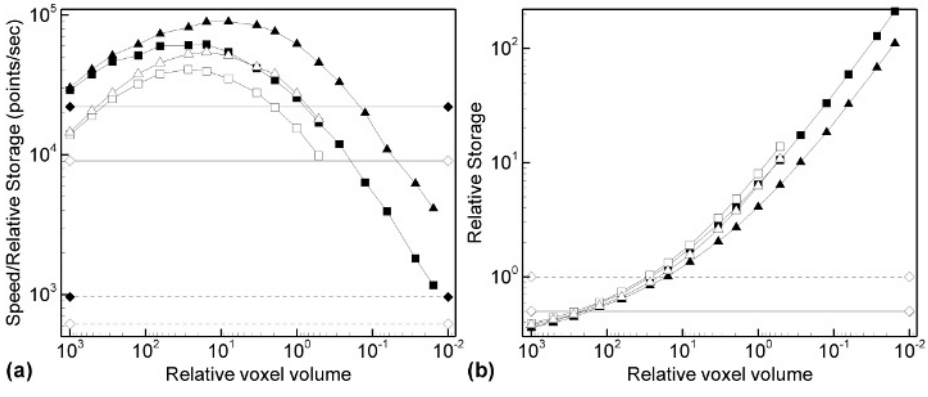

Fig. 5. (a) Speed normalized to relative storage, and (b) relative storage, defined as the memory required to store the search table divided by the memory required to store the respective mesh's connectivity table. See Fig. 3 for legend.

Discussion

Our results demonstrate that structured auxiliary mesh (SAM) search algorithms outperform nearest neighbour (NN) and alternating digital tree (ADT) search algorithms for the case of points randomly seeded around and/or within an unstructured mesh, independent of unstructured mesh density. Given the demonstrated ability to identify the location of more than 100,000 points/sec, and notwithstanding the additional CPU time required for interpolating the field data itself, these finding suggest that it should be possible to interactively slice through high resolution unstructured field data in real-time. To demonstrate this, we interpolated the velocity field of the 1.7M aneurysm model onto a 120×84 oblique slice using the SAM and SAMe algorithms each with a relative voxel volume of 8. The resulting velocity field data, shown in Fig. 6,

were interpolated within 30 and 20 msec, respectively, well within the 30-60 Hz required for real-time interaction.

Of course such real-time interpolation can be achieved through the use of hardware rendering techniques, but these are not always available or portable across architectures. Furthermore, they are typically restricted to simple element types such as linear tetrahedra. Although it is usually possible to decompose any type of finite element or volume into tetrahedra [4], this can compromise the accuracy of the interpolation, especially for higher-order elements. Instead, the element testing routines underlying the SAM algorithms can obviously be extended to handle any element type, at the potential loss of speed for complex or higher-order elements. Combined with the relative ease with which search speed, storage and setup time can be traded-off by adjusting the voxel volume, this makes SAM algorithms ideally suited to such software-based rendering. In this context even the few minutes of setup time required for the SAMe algorithm may be tolerated relative to the times typically spent interacting with such data. Moreover, the long setup times can be avoided by storing the setup tables for a default voxel volume along with the unstructured mesh, so that each time data is loaded search tables need only be reread, not recreated.

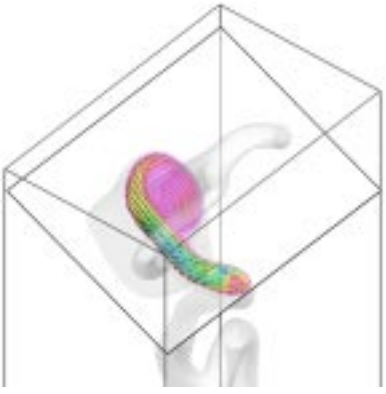

Fig. 6. Isospeed contours and velocity vectors interpolated onto an oblique slice through the 1.7M aneurysm model. Only every other vector is shown for clarity.

Although we have argued that SAM algorithms are ideally suited to the interactive visualization applications that motivate our work, it is important to recognize that SAM algorithms will not necessarily outperform NN and ADT algorithms for other applications involving geometric searching. For computations in the Lagrangian frame (e.g., particle tracking, methods-of-characteristics [5-8]), stability and/or accuracy restrictions may require the use of relatively small time-steps to integrate along a trajectory, in which case it is usually sufficient—and perhaps most efficient [9]—to merely check the current and neighbouring elements, obviating the need for the expensive distance calculations that otherwise compromise the efficiency of NN algorithms. In the context of mesh generation, SAM algorithms are also not necessarily superior to algorithms that have been developed for these purposes, such as the ADT algorithm, since digital trees are more easily updated as new elements are added [2].

Finally, we note that further trade-offs between search speed and storage requirements for the SAM algorithms may be possible. Minev and Ethier [6] suggested a storage strategy that reduces the size of the vector of pointers to each voxel's contents from the total number of voxels to twice the number of voxels actually intersecting the unstructured mesh. For the 1.7M model this would reduce the storage requirements by a factor or two for the smallest voxels tested here, but at the expense of potentially significant reduction in search speed owing to the introduction of a binary search step. (A similar trade-off may also be achieved by combining digital tree and SAM approaches [3].) Information about how the unstructured mesh was generated could also be exploited to optimize trade-offs between storage and speed, at the loss

of the algorithm's general applicability [10]. For example, although not attempted here, it might be possible to at least partially align voxel volumes along internal unstructured element boundaries for the octree-based meshes tested here.

Conclusions

SAM algorithms provide significantly faster geometric searches compared to digital tree and especially nearest neighbour algorithms. Relative voxel volumes of 5-10 appear to provide the best trade-off between search speed and storage requirements. Our novel variant of the SAM algorithm offers 1.5-2× improvements in storage and speed compared to the conventional SAM algorithm, but does so at the expense of an order-of-magnitude increase in setup times. We conclude that SAM algorithms are ideally suited to real-time, software-based interpolation of unstructured field data.

Acknowledgments. This work was supported by a grant from the Natural Sciences & Engineering Research Council of Canada (RGPIN 249746-02). D.A.S. was supported by a New Investigator Award from the Heart & Stroke Foundation of Canada. M.K. was supported by the CIHR Group in Vascular Imaging.

References

1. Bercovier, M., Pirroneau, O., Sastri, V.: Finite elements and characteristics for some parabolic-hyperbolic problems. Appl Math Model. 7 (1983) 89–96.
2. Bonet, J., Peraire, J.: An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems. Int J Numer Meth Eng. 31 (1991) 1–17.
3. Pissanetzky, S., Basombrio, F.G.: Efficient calculation of numerical values of a polyhedral function. Int J Numer Meth Eng. 17 (1981) 231–237.
4. Dompierre, J., Labbe, P., Vallet, M., Camarero, R., How to subdivide pyramids, prisms and hexahedra into tetrahedra, 8th International Meshing Roundtable, Lake Tahoe, 1999.
5. Buscaglia, G.C., Dari, E.A.: Implementation of the Lagrange-Galerkin method for the incompressible Navier-Stokes equations. Int J Numer Meth Eng. 15 (1992) 23–26.
6. Minev, P.D., Ethier, C.R.: A characteristic/finite element algorithm for the 3-D Navier-Stokes equations using unstructured grids. Comput Meth Appl Mech Eng. 178 (1999) 39–50.
7. Coppola, G., Sherwin, S.J., Peiro, J.: Non-linear particle tracking for high-order elements. J Comput Phys. 172 (2001) 356–380.
8. Tambasco, M., Steinman, D.A.: On assessing the quality of particle tracking through computational fluid dynamic models. J Biomech Eng. 124 (2002) 166–75.
9. Kenwright, D.N., Lane, D.A.: Interactive time-dependent particle tracing using tetrahedral decomposition. IEEE Trans Vis Comput Graph. 2 (1996) 120–128.
10. Mucke, E.P., Saias, I.A., Zhu, B.: Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. Comput Geom. 12 (1999) 63–83.