Space subdivision for Finite Element Meshes

by

Tom Simpson and Mark W. Jones
Department of Computer Science, University of Wales Swansea
Singleton Park, Swansea SA2 8PP, United Kingdom

Abstract

In this paper we present a modification of the octree spatial subdivision technique to enable it to be applied to unstructured data. The data is in the form of a finite element mesh, which results from fluid flow problems, and which needs to be visualised for effective understanding. A straight forward rendering results in huge computational expense unless the data is organised according to some method. An ideal spatial subdivision algorithm is that of the octree, but when applied to unstructured finite element meshes certain problems arise which will be described in the paper. We show that these problems can be overcome by using the presented modification.

Keywords: Spatial subdivision, Octrees, Quadtrees, Finite Element Data, Unstructured data, Visualisation.

1 Introduction

In attempting to render finite element data for visualisation, we encounter the problem that they contain many elements through which rays must be traced for volume rendering or in this case Direct Surface Rendering [Jon96, Jon97, Sim95]. Unless some form of spatial subdivision is used, each ray must be intersected with every element in the dataset. For regular data, a simple decomposition strategy such as octree spatial subdivision may be used very effectively [WG92a]. When applied to irregular data we encounter the problem that elements will have to be duplicated for every node they intersect. This results in extra point inclusion calculations, and wasted storage. This problem was identified by Andrew Glassner within his introduction of octrees to ray tracing [Gla84].

In this paper we present a modification to the octree subdivision strategy which removes the necessity for duplicating elements within every intersected node. We apply this method to typical finite element mesh (FEM) visualisation problems we encounter, and demonstrate that it is an effective spatial organisation technique.

In Section 2 we describe the geometry and topology of FEMs, in particular the FEMs we are visualising. Section 3 reviews the spatial subdivision literature and highlights some of the problems that are encountered, and the lack of information about the subdivision techniques used. The problem cases that FEM data presents are discussed in Section 4, and the new extended octree cover is presented in Section 5. Finally the results are presented in Section 6.

2 Finite Element Data

The domain under consideration is constructed as a finite element mesh, and may be regarded as a structure M containing two fields, the *geometry* field and the *topology* field; $M =< N, E >$. The *geometry* is defined as a set N of points termed *nodes*, where each node n_i has associated with it the following properties :

- n_i^{pos} : A position vector in a world co-ordinate system.

- $n_i^{\{d_1,..,d_q\}}$: A set of data, typically the results of some simulation.

The set of individual data components d_f over all the nodes for a fixed f we term a *datafield*. i.e. for a fixed f between 1 and q, the datafield $= \{n_1^{d_f}, .., n_i^{d_f}\}$. Each datafield will represent all the results for a particular physical property, e.g. pressure, viscosity, stress etc.

The *topology* is a set E of geometric objects termed *elements* or *cells*, which describe the connectivity of the nodes. Triangles and tetrahedra are the most common geometric form. Each element $e_j \in E$ may be regarded as a tuple, $e_j =< n_1, .., n_l >$ where each n_i is a node of N.

3 Existing methods

For the most part, much of the work done into spatial subdivision has been based on regular data [Glo93, Lev90, WG92b, Fan95]. That is data obtained from sources such as CT, MRI, and X-Ray scanners. Others mention irregular data, but provide no clues as to the inclusion predicates used, for example [WJ93]. It is likely, that the data is simply replicated over all intersected nodes. The implications being that memory is wasted, and redundant point inclusions tests may be performed. A good review of spatial subdivision methods is provided by Samet [SW88], who describes a variety of hierarchical data structures for computer graphics. In particular, a method for encoding 2D vector data (of which finite element data is effectively a special case) is described, termed the PM_1 quadtree. The inclusion predicates (or leaf criteria) for a PM_1 quadtree are as follows:

- There can be at most one vertex in an image space.
- If there is a vertex in the image space, then all line segments in the image space must share that vertex.
- If there are no vertices in the image space, then there can be at most one line segment passing through the image space.

This form of quadtree has a number of drawbacks for our purposes. Firstly, the leaf criteria are relatively complicated, in some cases requiring complex line intersection tests to determine inclusion. Secondly, the quadtree encodes the edges and vertices of the data, not the polygons they construct. In the case of finite element domains, it is these polygons, not the vertices that are of interest. Finally, the PM_1 quadtree does not expand easily into three dimensions.

A better solution, which directly addresses the problem of the inclusion predicate for irregular grids is provided by Neeman [Nee90]. In this method, a median cut algorithm is used to partition the data. The position of the cut plane is adjusted to minimise the number of elements intersected by the plane. However, regardless of how well the plane is positioned, some elements will always be split, and these are then replicated on either side of the cut. Furthermore, testing for element/plane intersection is an additional overhead which may cause problems for large datasets. More importantly however, Neeman still bases the inclusion predicate on the area of the actual element. The implication of this is that the algorithm gives poor decomposition for highly irregular grids, such as used in many CFD calculations. It is therefore suggested that the algorithm works best for datasets with *regular* connectivity. A similar method, based on median cut for volume rendering of sparse regular data, is described by Subramamian and Fussel [SF90].

4 Problem Cases (FEM)

We now concern ourselves solely with spatial subdivision applied to irregular, finite element domains. Our problem lies in determining how far (or to what level) the spatial subdivision has progressed through the dataset. That is, how many elements in E are contained within each node of the subdivision. For regular data such a test is trivial, with the move to irregular, and in particular finite element domains, the predicate "is an element contained within a node?" becomes more obscure. In particular, we can isolate three problem cases associated with a traditional octree, and inclusion predicates for Finite Element Data. We describe each case using the diagram below illustrating a triangular finite element mesh, constructed of two elements spanning four quadrants of a quadtree.[1]

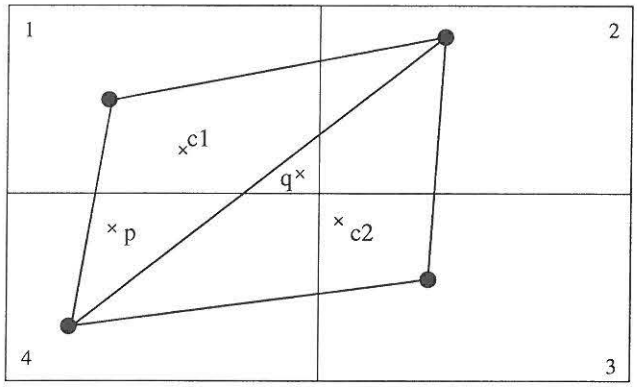

Figure 1: *A simple finite element dataset within a quadtree structure.*

4.1 Inclusion predicate 1

For the first case we consider using the centroids of the elements to test for element/node inclusion. This simple predicate has a number of advantages. Firstly, it is a simple boolean test, hence quick. This is a primary concern since the datasets we are considering typically may have many hundreds of thousands of elements. An overly complicated test will most likely take too long to construct the tree and therefore be unusable in this context (*cf.* The PM_1 quadtree predicates). Secondly, the element is referenced by only one point, thus ensuring it is entered in the tree once only. Again, more complex inclusion predicates do not have this property, giving rise to data duplication and slower timings when the tree is traversed later on.

The centroid predicate falls down though on one simple case, illustrated in figure 1. Consider the element with centre c_1, and a point to be located positioned at p. With a quadtree built on element centroids, this element will only be referenced in quadrant 1, though it clearly extends into quadrants 2 and 4. Therefore, in this form, the centroid method is flawed.

4.2 Inclusion predicate 2

In the second case, rather than basing the inclusion predicate on element centroids, we instead use the nodes of the mesh structure. This resolves the flaw of the centroid method since the element centred on c_1 can now be seen in all three quadrants it intersects. Using nodes though presents many more problems than the centroid method. Firstly, it too has a flaw, shown by the

[1] All the examples expand directly to 3D and an octree structure.

element centred on c_2, and the test point q. It can be seen that in this case, though the element intersects quadrant 1, it has no node, or centroid there, hence is unreferenced and the test will fail. Secondly, for the most part, we are interested in the elements, not simply the nodes, of a finite element domain. Traversal of a structure based on the nodes requires a linked-list structure stored at the nodes listing the connected elements. This is far from ideal.

4.3 Inclusion predicate 3

The seemingly obvious solution to both these problems is to simply use element bounding boxes, rather than centroids or nodes to define the element/node inclusion predicate. For normal irregular data, this is a common method and presents no problems. With irregular data, and its high degree of shared element boundaries, a problem case does however arise. Consider Figure 2, here a structure of elements are based around and share a central node. The dotted lines indicate the boundaries of the quadtree nodes at this level. The important point to note is that the bounding boxes of all the elements in this diagram will be contained in the inner four quadrants of the tree *regardless* of how far we subdivide. Though the diagram is obviously designed to show a worst case scenario, it is far from rare and proves to be a real problem with an implementation.

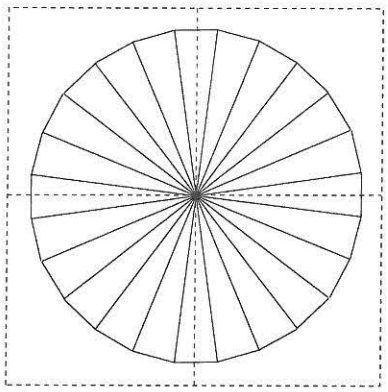

Figure 2: *A scenario when the bounding box predicate causes problems.*

5 An octree cover for FEM Data

In this section we introduce a structure which modifies both quadtrees and octrees to allow their application to irregular, finite element data. This method is based primarily on the centroid test described above, since this is the most computationally efficient inclusion predicate. Once the element has been placed in a node according to its centroid, we modify the node structure to account for the problem cases.

5.1 The node structure

In traditional space subdivision, the domain under consideration is split at each level into equal sized nodes, however, we have seen that for the centroid method this fails to reference the complete extent of included elements. We resolve this by relaxing the constraint of equal sized nodes on a level, and allowing the modification of the extent (size) of the node. In the direct surface rendering method, rays are cast into the data, and their intersection with the surface

is determined. In the case that each element only occurs in a normal size node, the problem described in Section 4.1 occurs. In other words, the ray will *miss* elements that have a centroid in one node, but extend into neighbouring nodes, when it is traced through those neighbouring nodes. This modification provides a method by which a ray can be tested with all elements that intersect the node it is being traced through. The algorithms that is used can be summarised by the following piece of pseudo-code.

```
build_tree (data, node_orig)

thisnode.list = {}
thisnode.type = internal
thisnode.extent = node_orig.extent
for each element e in data
        if e is in node_orig.extent then
                numfound++
                thisnode.list = thisnode.list + e
                if e.extent > node_orig.extent then
                        extend(thisnode.extent)
                endif
        endif
endfor
if numfound < nodeing_threshold then
        thisnode.type = leaf;
        return (thisnode)
else
        for each subdivision i node_orig_sub of node_orig
                thisnode.next[i] =
                                build_tree(data, node_orig_sub)
        endfor
endif
return(thisnode)
```

At each level of the subdivision process, a list of candidate elements is searched for inclusion within the node. In the traditional octree building process, once more than a predetermined number of elements has been found within a node, the node can be immediately subdivided without performing any more checks. This algorithm does not stop searching for elements once it has found more than the required threshold since there is a need to find the extent of each node. If the algorithm was to stop searching once the required threshold had been reached and started subdividing, elements could be located later in the process that had larger extents than nodes higher in the octree.

Figure 3 shows the same problem cases with the new extent information at the subdivision nodes. Considering again problem case 1; the location of p outside quadrant 1. We can now see that p is contained within the extended boundary of quadrant 1 (dotted line $ex1$), and the normal boundaries of quadrant 4. A search on this point will now match with the element centred on c_1 when quadrant 1 is traversed.

With problem case 2, the search point q is now contained within the extended boundary of quadrant 3 (dotted line $ex3$), hence the search succeeds with the element centred on c_2 being found. Finally, since the structure has been built on one point per element (the element centre) which is not shared, problem case 3 never arises.

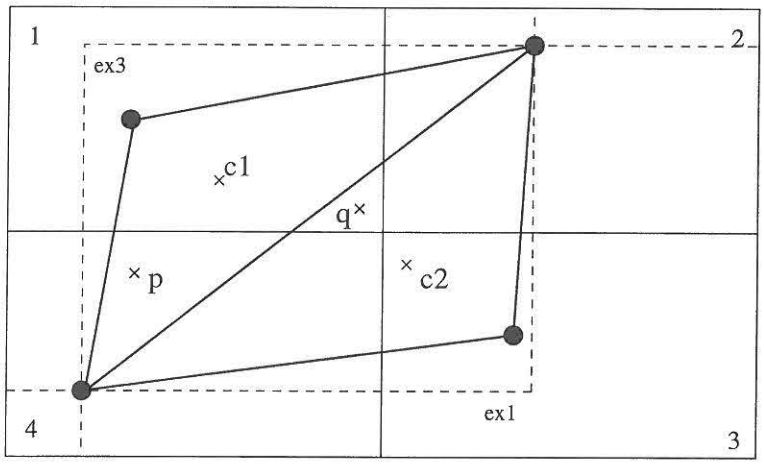

Figure 3: *The solution of the problem cases via extended nodes.*

A side effect of the extended nodes is that some degree of overlap will exist, hence a number of points will fall into one or more nodes. This gives rise to a very small increase in traversal times since points are still only being compared to elements in a very localised area.

The problem the algorithm poses is that if naively implemented, the requirement for an exhaustive search at each octree node would slow the method down to unusable speeds. In response to this we also concentrated on the building of the octree.

5.2 Building the tree

Clearly, we need some method which allows exhaustive searching of the complete dataset while not causing a large overhead in inclusion tests. A simple method is to construct a linked list at each node listing the included elements. This list is passed to the child nodes as the candidate list of elements, and so on for each level in the octree hierarchy. This technique is quick (tested on a DEC Alpha 2100, over 12,000 elements were built into a tree in under a second), but some additional memory is required to store an extra copy of the dataset. This memory overhead can be overcome by storing an octree code for each element, but this results in slower timings.

Alternatively a bottom up approach can be used. Each element is examined in turn, and it is decided at which leaf node the element would have to be stored. Any additional information about the extent the node should have to cover the element is also stored. Once the placement for all elements has been calculated, the tree is then traversed in order to determine which nodes can be combined into larger nodes and still fulfil the requirement of containing less than a specified number of elements within the node.

6 Conclusions

The modified octree building method was coded using C on a Dec Alpha 2100, and was used to accelerate ray/element intersections for direct surface rendering. An example of such a rendering for a 3D flow problem is shown in Figure 4. The algorithm was tested on a 257250 element mesh with 357911 nodes to produce an image of dimensions 400×400. With the modified octree, the final image took 35 seconds to produce, and without using an octree the image took about 10 hours. This demonstrates the usefulness of using a spatial subdivision

technique over no technique at all. We have as yet not been able to compare the performance with the standard implementation of the octree, which would include repeated elements.

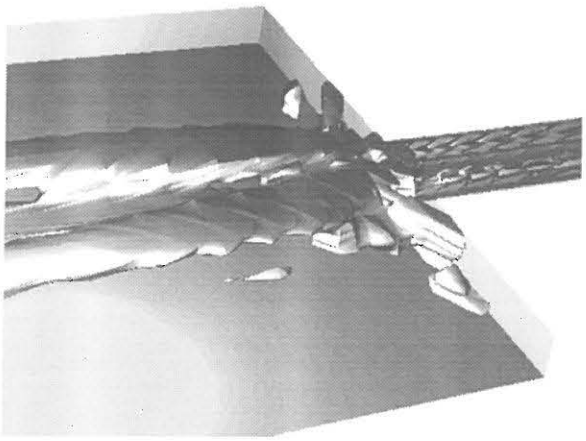

Figure 4: *Direct surface rendering of a 3D expansion flow.*

In this paper we have presented some of the problems that are encountered when attempting to visualise irregular data. We have shown that the inclusion of irregular elements in any spatial subdivision can be a computationally expensive operation if the problems mentioned in Section 4 are to be avoided. At the expense of complex inclusion tests and duplicating elements at various nodes throughout the tree, these problems can be resolved. For large datasets, such overheads can be restrictive. The modification we have presented in Section 5 allows each element to appear just once in the octree reducing memory costs. The simple centroid based inclusion predicate allows computationally inexpensive building of the search structure.

References

[Fan95] Shiaofen Fang. An efficient volume rendering algorithm by octree projection. *unknown*, 1995.

[Gla84] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[Glo93] A. Globus. Octree optimization. *unknown*, 1993.

[Jon96] M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5), December 1996.

[Jon97] M. W. Jones. An efficient shadow detection algorithm and the direct surface rendering volume visualisation model. *Submitted to UK Eurographics Conference 1997*, March 1997.

[Lev90] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[Nee90] Henry Neeman. A decomposition algorithm for visualising irregular grids. *Computer Graphics*, 24(5):49–56, November 1990.

[SF90] K. R. Subramamian and D. S. Fussel. Applying space subdivision techniques to volume rendering. *IEEE Computer Graphics and Applications*, pages 150–159, 1990.

[Sim95] T. Simpson. Direct surface rendering for irregular, quadratic, volume data. Technical Report CSR-4-96, University of Wales, Swansea, 1995.

[SW88] H. Samet and Robert E. Webber. Hierarchial data structures and algorithms for computer graphics. *IEEE Computer Graphics and Applications*, pages 49–69, May 1988.

[WG92a] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[WG92b] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[WJ93] D. M. Weinstein and C. R. Johnson. Hierarchical data structures for interactive volume visualisation. Technical Report UUCS-95-012, Dept. of Computer Science, University of Utah, Salt Lake City, UT 84112, USA, 1993.