Tools and Techniques for Parallel Grid Generation

Michelle L. Simone
Hugues L. de Cougny
Mark S. Shephard

Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180-3590, USA

Abstract

This paper considers the issues associated with the development of a scalable parallel automatic three-dimensional mesh generator. Emphasis is on the distributed octree structure used to control the parallelization of the mesh generation process. A three-dimensional mesh generator which builds on these structures is outlined and some preliminary results produced by the mesh generator are presented.

Introduction

The introduction of scalable parallel computers is allowing solutions on grids of several millions of elements. As sizes become this large, mesh generation on a serial computer becomes problematic in terms of time and storage. This paper discusses our continuing efforts to address this problem by the development of a parallel mesh generation procedure that will operate on the same computer as the parallel analysis procedures.

Key to the development of an effective parallel mesh generator is balancing the computational effort among the processors. Since the computational effort required to generate the mesh is dependent on the distribution of elements, which is not well understood until the mesh is generated, maintaining load balance on a distributed memory computing environment is difficult. The approach taken here is to employ an underlying octree structure to obtain estimates of the computational work load and to distribute the octree to the processors in an effort to maintain load balance while keeping interprocessor communications to a minimum. Since knowledge of the computational effort required is only determined as the meshing process continues, procedures which allow the redistribution of the octree among the processors are used to maintain load balance as the meshing process continues.

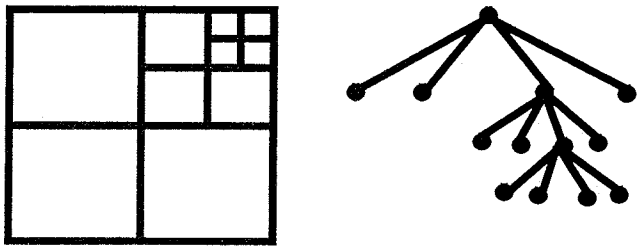
The next section outlines the meshing algorithm and indicates how the octree is used by the mesh generator. The following section focuses on the octree, indicating its structure, how it is distributed to the processors and how it is used to parallelize the meshing process. This is followed by a section which demonstrates its use by the mesh generator.

Outline of Mesh Generation Approach

The grid generation procedure presented here automatically meshes three-dimensional non-manifold objects following the hierarchy of topological entities. That is, the model edges are meshed first, the model faces are meshed second, and the model regions are meshed last. Similar to our previous mesh generation procedures[1], an octree structure is used to provide control over the mesh gradations and the mesh generation process. In the

current work, it is also the structure used to control the parallelization of the meshing process. The central role of the octree in controlling the parallel mesh generation processes has a direct influence on specific aspects of the mesh generation process. In particular, steps in the surface mesh generation are designed to directly use the octree to gain parallel efficiency.

The octree is a spatial data structure used to localize mesh data during mesh generation and control the size and gradation of elements across the geometric model. Its hierarchical structure, shown in the two dimensional quadtree form in Figure 1(b), is based upon parent-child relationships between its members, known as octants. The root octant, the member at the very top of the hierarchy, represents the largest closed space and is constructed to enclose the entire geometric domain of interest; while the terminal octants, those members at the very bottom of the hierarchy, represent the smallest closed spaces whose union is equal to the space occupied by the root octant. A variable level octree is constructed by the mesh generator until it adheres to input mesh specifications on model entities and is consistent with the surface triangulation of the model. Terminal octants are refined until their descendents are of the same order in size as the desired elements in a particular region of the geometric model. In order to obtain a smooth gradation of element sizes across the domain, not more than a one level difference is allowed between terminal octants sharing faces and edges of the octree grid.



(a) quadtree grid (b) quadtree structure

Figure 1 Quadtree data structure used to discretize R^2.

Model edges are easily meshed by the placement of nodes along them reflecting the mesh size requirements. Mesh size requirements typically come from: (i) the user by specifying *a priori* sizes for mesh entities classified on some model entities, (ii) the user by specifying a maximum discretization error (maximum relative distance between a model and mesh entity), and (iii) proper mesh gradation. A tree satisfying the 2:1 level of difference rule is used to control mesh gradation. Model faces are meshed in two stages: (i) "coarse" meshing using the Delaunay criterion in the parameter space [2] and (ii) "refinement" using the Delaunay criterion in the real space to satisfy mesh size requirements [3].

Given the surface mesh, the regions are meshed using a combination of meshing templates and element removal procedures. Templet meshing is applied to octants which are interior to the domain which have all corners sufficiently far from any surface triangle. The volume between the surface triangulation and the interior octants is then meshed using a face removal procedure. Each face on the front is removed by connecting it to an existing vertex in its neighborhood using either the Delaunay criterion [5][4], or a classic advancing front method [5] when the Delaunay criterion is not satisfied. The octree structure is used

during the face removal process to efficiently determine the mesh entities the face removal must consider when removing a face.

An Effective Octree Structure for Parallel Mesh Generation

The octree structure controls the parallelization of the mesh generation process. Octants are assigned to the individual processors with the goal of providing each with equal computational load. An individual meshing operation is performed (on processor) only if the required meshing information is available. When the information required to perform the meshing operation is not fully local to the processor, the octree-based procedures obtain that information from the appropriate processors.

Key to the parallel efficiency and scalability of the meshing process is controlling the interprocessor communications during mesh generation. Also key to the scalability of the parallel mesh generator is the ability to distribute the data structures of the mesh and octree to the memories of the various processors.

In a standard octree structure, information in a neighboring octant is obtained by performing a tree traversal [6]. Such traversals are not desirable in the current procedure since they introduce a factor of $O(\log N)$, where N is the number of octants, into the growth rate of the process. Most importantly, each such traversal can introduce multiple interprocessor communications, thus introducing a major communications bottleneck. Traversals to find neighboring information can be eliminated by extending the octree structure to include direct links to neighbors. The addition of this information does increase the size of the octree structure. However, if done in a clever manner, the amount of additional information can be kept small enough such that the increase in the total data storage, octree and mesh, is only 2.5%.

In the presented octree structure, direct links between octants sharing faces of the octree grid are stored and maintained while the tree is constructed. Storage is kept to a minimum by having each octant point either directly to its face neighbor of equal size, or face neighbor of larger size if an equal one does not exist, therefore requiring an additional six pointers per octant. These face neighbor pointers are then used to obtain any desired terminal octant face, edge, and corner neighbors during mesh generation, requiring only a minimal amount of hopping through the face neighbors to get at the needed information. The number of hops required to find an octant's terminal neighbors is proportional to the level differences between the octant and its terminal neighbors. Therefore, in the worst case, the amount of hopping will be proportional to the depth of the tree, $\log N$ (base 8). However, since the primary use of the tree during mesh generation is after it has been adjusted (to have only one level difference), the number of hops required to find any terminal octant neighbor is, in practice, constant.

The retrieval of terminal face, edge, and corner neighbors is easiest to understand by considering a balanced tree. For this case, the retrieval of face neighbors is an $O(C)$ operation because the information is stored in the octant. Finding edge neighbors requires: (i) one hop to one of the face neighbors sharing the edge and (ii) the direct retrieval of that face neighbor's neighbor on that edge. Finding a vertex neighbor requires: (i) two hops to retrieve the edge neighbor and (ii) the direct retrieval of the edge neighbor's neighbor sharing that corner. When the tree is allowed to have a one level difference between its face and edge neighbors, the number of hops to get the terminal face neighbors is bounded by one: one hop over to get the equal face neighbor, then an $O(C)$ operation to get the pointers to the four children on the face since that is stored. The number of hops to find

the terminal edge and vertex neighbors is bounded by three and four respectively.

Structure of the Distributed Octree

Since the size of the octree structure grows with the size of the mesh, scalability of the mesh generation process can only be maintained by storing those portions of the mesh and octree, along with appropriate neighbor links, assigned to the processor in the memory of that processor. A Parallel Mesh Database (PMDB) has been developed to support the distribution of the mesh with the required interprocessor links to retrieve off processor mesh adjacency information [7]. PMDB is built on top of the SCOREC Mesh Database [8] and uses MPI [9] to update the mesh structure in the distributed environment.

Likewise, a distributed octree (PODB) supports the distribution of the tree over the memory of the processors. An important requirements is the ability to visit local octants without the need to communicate. Communications bottlenecks are a possibility with the structure in Figure 1 due to the hierarchical nature of the tree. A local root list structure, added on top of the octree structure, is an intermediate storage space containing pointers to local octants whose parents are located on a remote processor. The advantage of such a scheme is that any descendents under an octant with a remote parent can be visited without communication. In order to access any on-processor (local) octant, the mesh generator traverses local subtrees rooted at the local roots without communication. Therefore, the number of local roots should be kept to a minimum. If there is a need to access off-processor octants, messages are sent to the appropriate processors. This kind of traversal requires synchronization. Figure 2 is an example of a distributed binary tree. Octants on processor are represented by a bullet. Local roots are represented by a square. A dashed line is a directed off-processor link to either a parent or child. For localization, the parallel mesh generator links mesh entities in the PMDB with terminal octants in the PODB.

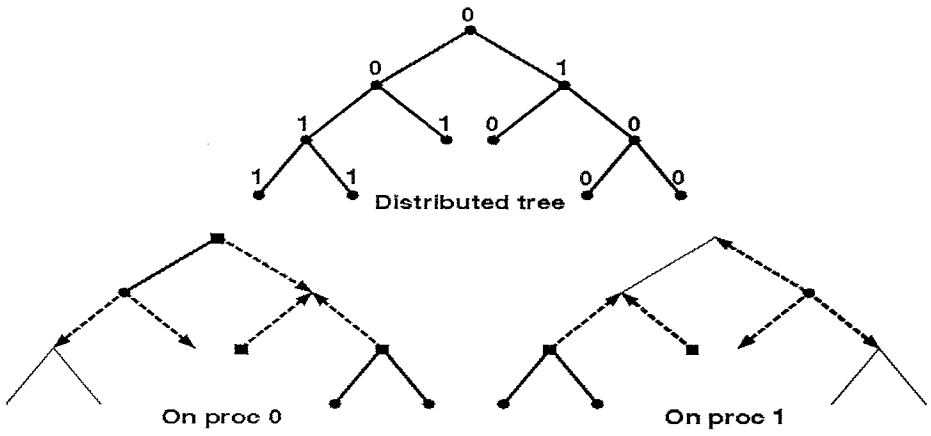


Figure 2 An example of a distributed binary tree.

Tools to Support the Distributed Octree

Specific parallel tools are required to support the application of the distributed data structures during parallel mesh generation. Included in this set of tools are procedures to partition and repartition the domain and migrate the information (tree and mesh).

The purpose of partitioning is to subdivide the domain of interest into sub-domains of equal load. The partitioner makes uses the octree to support that process. The starting point for the tree partitioning procedure is the global root which carries the cost of the entire process. If the cost of an octant is too large with respect to some average cost (say, total load divided by number of processors), the octant is subdivided and its associated cost is passed on to the children. This process of subdividing octants ends when all terminal octants have acceptable costs. In the current tree partitioner, terminal octants are assigned to processors in a depth first search (DFS) visit. Assigning terminal octants in a DFS fashion yields partitions with a high degree of geometric connectivity. This is followed by a consideration of where to migrate the ancestors of the terminal octants. Parents are migrated to the processor where most of their children will reside in order to minimize the length of the local root lists. Another approach is a "divide and conquer" methodology which migrates as the tree is built. The distributed tree resulting from this process is referred to as "partitioning" tree. Note that repartitioning the tree involves the ability to partition the cost starting from an initial partition.

Octant migration procedures support the arbitrary mobility of octants needed by the partitioning and repartitioning algorithms. Octants targeted for relocation, by the partitioner or repartitioner, are migrated with octant migration procedures which are responsible for updating the tree connectivity across multiple processors. The migration of a set of octants from a given processor to a destination processor occurs in three stages. In the first stage, all requests are sent out to destination processors. The destination processors, upon receiving the requests, allocate space dynamically for each octant which will be relocated to the processor. Destination processors respond back to requesting processors with the new location of each octant which will be moving. In the second stage, each migrating octant, which now knows its new location, updates all its relatives (parents, children, and face neighbors), as to where it will be moving. This requires either a local visit to a relative if it is on the same processor as the migrating octant, or a communication update if the relative is located on another processor. Each relative is modified to point to the new octant address. Once all the relatives have been told of the new octant address, the third stage actually moves the octant data from the old location to the destination location, freeing the memory for the old location. Octant migration is accompanied by mesh migration [10] which sends the associated mesh entities to the appropriate processors.

Construction of Distributed Octree for Parallel Mesh Generation

The construction of the tree during the mesh generation process occurs in two stages: (i) a tree refinement process, whereby any terminal octant is refined until the sizes of all terminal octants are consistent with required mesh sizes on the model boundary and (ii) an adjustment process which enforces no more than one level difference between terminal octants and their terminal face and edge neighbors for purposes of controlling the mesh gradation and allowing application of meshing templates. The ability to partition both the tree and mesh data allows both steps to occur in parallel with very little communication on the partition boundaries. Previous efforts to perform these steps in parallel were not possible since the tree was duplicated on all processors [11][12].

Step (i) calls for the construction of a variable level octree which adheres to input mesh specifications on model entities and is consistent with the resulting triangulation of the model boundary. In the surface mesher, the cost considered for balancing is related to: (i) imposed mesh size and (ii) maximum discretization error (relates to curvature) on some model entities. This cost is determined by sampling the curvature on model entities. A

"partitioning" tree is constructed using the above defined cost. This tree is refined according to the mesh size requirements imposed by the user on model entities. Typically, each processor considers mesh sizes at sample points on the model faces it is responsible for meshing. A tree level is obtained for a mesh size (log (*rootsize/meshsize*) (base 2)) and is used to refine the tree to a proper level. The terminal octants that contain each sample point are gathered in $O(\log N)$ (base 8) time. Any such terminal octant is refined if its level is less than the level coming from the required mesh size at the sample point.

The adjusting process refines the tree further. A one level difference between octants sharing faces and edges is enforced during this process to control smoothness of the mesh gradations and allow templates to be applied at a later stage. The adjustment involves visiting each terminal octant, and refining it, if necessary. In order to avoid revisiting the same octant multiple times, terminal octants are sorted into buckets corresponding to their level number. Such an operation requires one complete traversal through the tree in $O(N)$ time. Adjusting begins by looping over all terminal octants beginning with the second highest bucket, corresponding to the second highest level number, retrieving the terminal neighbors of a given octant in, at worst, $O(\log N)$ time, and refining the octant recursively if any of its edge and face neighbors are more than one level smaller in size. Since the work required to adjust the tree is, at worst, proportional to $N \log N$, efficient parallelism requires an equal number of terminal octants on each processor. Thus, the cost of each terminal octant is equally weighted. The tree is repartitioned if the cost associated with the tree is not well balanced. However, the work required to partition, that is, compute cost values, and migrate is typically not justified for this particular step.

Some communication is required during the adjusting process to retrieve off processor terminal face and edge neighbors of octants on the partition boundary. The amount of communication needed depends on the size of the partition boundaries and the number of terminal octants on those boundaries. In order to avoid communicating frequently, the approach taken is to first adjust each part of the tree with the local tree neighbor data available on the memory of each processor, synchronize once, exchange all the needed neighborhood information on the partition boundaries, then re-adjust to correct for off processor level violations.

Surface Mesh Generation in Parallel

Parallelism is proposed at the model entity (vertex, edge, face), or sub-entity level. That is, a processor is responsible for the complete meshing of any model entity, or portion of a model entity, it has been assigned to. Scalability with respect to the number of mesh faces generated requires the ability to split model faces. Splitting model faces without modifying the geometric model requires the presence of an additional database referred to as the model database. The driving process for model face splitting is similar to the "partitioning" tree building, that is, a model face is split if it carries too high a cost. The cost carried by a model face is defined as the expected number of mesh faces that the surface mesher will generate. This can be obtained from the information contained in the sample points. A model face is assigned (for meshing) to the processor that has most of the terminal octants in its path. Model vertices and edges bounding a model face are assigned to the same processor as the face. Once assignment of model entities is complete, model entities are migrated accordingly.

Model boundary meshing is hierarchical, that is, model entities are meshed in vertex, edge, and face order. Once distributed, surface meshing can be conducted without any

communication and synchronization. Model vertices are meshed by creating vertices at the location of model vertices. Given a model edge, model edge meshing starts by creating a mesh edge discretizing the whole entity. Then, considering the imposed mesh sizes, the mesh edge is refined if necessary. Model faces are meshed using a Delaunay approach in the parameter and real spaces. Given a model face, a coarse mesh is obtained using the Delaunay criterion in the parameter space. The mesh vertices in the coarse mesh come from the discretizations on bounding model entities. The coarse mesh is refined in the real space using the Delaunay criterion for surface triangles, as seen in [3]. Basically, mesh edges are considered in turn and refined (if necessary) by inserting vertices at the middle. Once all model entities have been meshed, mesh entities (vertex or edge) on mesh partition boundaries are linked together as required by the PMDB.

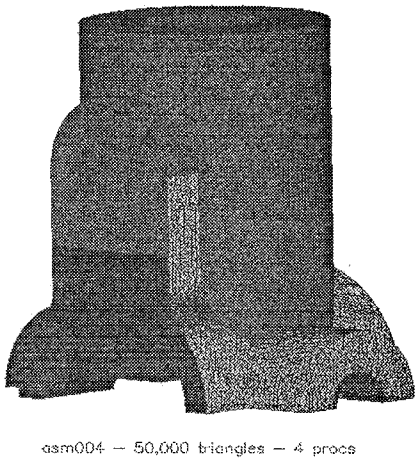


Figure 3 Example of surface mesh generation on 4 processors

Region Mesh Generation in Parallel

Region meshing proceeds after the surface triangulation is available. The approach taken is to apply templates to the interior terminal octants far from the surface triangulation for fast region meshing, then use an advancing front technique to complete the meshing process. Faces on the resulting front are removed using either a Delaunay or classic advancing front procedure. Templates and face removals are supported by an underlying octree structure. It is not necessary to construct a new tree for the region mesher, as the one constructed by the surface mesh generator is sufficient.

In order to support the localization needed by the face removal procedure, the region mesher establishes links from the terminal octants to the surface mesh faces they partially or fully enclose. Additionally, the application of templates requires identification of terminal octants located at a sufficient distance from the surface triangulation. Since both the tree and the mesh are distributed, these steps can now be performed in parallel with very little communication on the partition boundaries. Previous efforts to perform these steps in parallel were not possible since the tree was duplicated on all processors [11][12]. Assuming the face removal stage is computationally the most expensive of the region meshing, the load carried by a processor for region meshing can be defined approximately as the number of front faces. This load corresponds to the load used in surface mesh gen-

eration, therefore, it is expected that the current surface mesh distribution will lead to a balanced region mesher.

Inserting mesh faces into the tree is performed by looping over each mesh face, obtaining its bounding box, and traversing down the tree to locate the terminal octants which interfere with the box. Locating the terminal octants in the path of a face's bounding box requires (i) traversing the local subtrees rooted at the local root list and (ii) communicating to inform the off processor neighbors about the mesh faces close to the partition boundaries. In practice, each mesh face is inserted into the local tree. Once all mesh faces have been inserted locally, one synchronization is performed to exchange all the mesh data on the partition boundary so that neighboring off processor octants can know about the off processor mesh entities they enclose. Current efforts are focused on alternative methods which use the adjacency information in the tree and PMDB in order to avoid traversing.

Classifying terminal octants begins with terminal octants which contain the boundary of the model. Given that each boundary octant has been identified in previous steps, classification begins from a "seed" boundary octant. A ray firing technique is used to classify a corner of the seed octant as either interior or exterior to the surface mesh of the geometric model. Once the status of the corner is known, the classification of the corner propagates to the terminal neighbors sharing that corner, if they are not already classified. A recursive procedure then classifies terminal octants using corner neighbor information. The recursion ends when every terminal octant has been classified.

Once interior terminal octants have been identified, templates are applied to each interior terminal octant located a sufficient distance from the surface triangulation. Template meshing is fast because no mesh validity checks are required, and by construction, adjacent elements are guaranteed to be compatible. The selection of the appropriate template is determined by inspecting the octant neighbors surrounding the edges and faces of a given interior terminal octant. Once the appropriate template has been found, the octant is meshed in hierarchical fashion.

Some communication is required during this process. Since the selection of templates requires knowledge of the number of neighbors on each of an octant's face and edges, meshing octants on the partition boundary requires a communication step to find the number of remote terminal face and edge neighbors. Since the number of communications would be proportional to the number of terminal octants on the partition boundary, the approach taken is to perform one synchronization step to retrieve all the non-local neighbor data that will be needed to support all local template meshing on each processor. The drawback to this approach is a higher storage expense. However, the amount of extra storage, which is proportional to the number of terminal octants on the partition boundaries, should be small. With all the needed data available, each processor can mesh all its interior local terminal octants completely in parallel. A final synchronization is necessary to set off processor links in the PMDB, the details of which are located in [7].

After templates are applied to interior terminal octants, face removal procedures are responsible for meshing the region between the triangulation of the interior terminal octants and the surface triangulation. Face removal procedures depend upon tree neighborhoods to supply information regarding the mesh front data in a given region of space (the space enclosed by a set of terminal octants). More specifically, tree neighborhoods provide face removal with a suitable list of mesh vertices with which to remove each mesh face on the front. In addition, tree neighborhoods are responsible for obtaining the appro-

priate mesh data, needed for mesh validity checks, to determine if the connection of a front mesh face to a vertex is acceptable. When the required tree neighborhoods are available on each processor, face removal is able to form elements, in the partitions, without any communication. For mesh faces too close to partition boundaries, however, this will not be the case. In such a situation, a connection to a target is postponed until knowledge of the entire target neighborhood is made available by a repartitioning step. The approach taken is to process as many face removals as possible on each processor using only the local neighborhood information available, *i.e.,* no synchronizing or exchange of data is required. When the processors are no longer able to mesh any more elements as a result of lack of required neighborhood data, a repartition step must be performed in order to move data from a heavily loaded processor to its least loaded neighbor. Figure 4 shows the faces remaining on the front after the first round of face removal. A repartitioner is needed to complete the mesh.

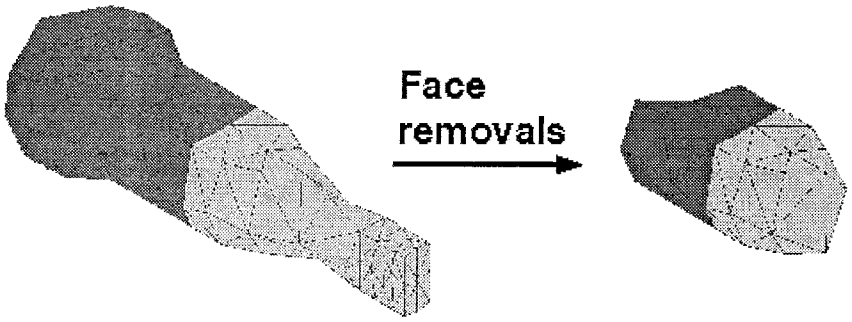


Figure 4 Initial face removal step on two processors.

Repartitioning needs to be considered when all front faces require off-processor tree neighborhoods. No more than three repartitioning steps are needed (in between face removal steps). The first two steps basically repartition the domain remaining to be meshed using the tree so that the assumptions for the third step are met. The third partitioning step assumes that the domain to be meshed is decomposed into independent (disconnected) sub-domains of "small" sizes. A sub-domain is assumed to be of "small" size if its size is of the order of the sizes of the bounding faces. If the sub-domains to be meshed are "small", a localizing tree is not needed to mesh them, which greatly simplifies the process of repartitioning.

Closing Remarks

This paper has presented a set of structures and techniques for scalable parallel automatic mesh generation of three-dimensional geometries as defined in a solid modeling system. Key ingredients to effective parallization of the mesh generation process in a scalable manner are:

- The use of a distributed octree decomposition of the geometric domain for controlling the parallel processes and localization of the meshing process.
- The extension of the octree structure to include direct neighbor pointers which eliminate the need for tree traversals when looking for information in a neighboring octant.
- Tools to allow the efficient parallel redistribution of the octree and associated mesh information to maintain load balance as the meshing process progresses.

Acknowledgment

The authors would like to acknowledge the support of NASA Ames Research Center under grant NASA-AMCS NCC2-9000, the Office of Naval Research through grant N00014-95-1-0892 and the Army Research Office through grant DAAH04-95-1-0091.

References

1 M. S. Shephard and M. K. Georges, "Automatic three-dimensional mesh generation by the finite octree technique," *Int. J. Numer. Meth. Engng.*, **32(4)**, 709-749, 1991.

2 P. L. George and F. Hecht and E. Saltel, "Automatic mesh generator with specified boundaries," *Comp. Meth. Appl. Mech. Engng.*, **92**, 269-288, 1991.

3 L. P. Chew, "Guaranteed-quality mesh generation for curved surfaces," *Proc. of Ninth Annual ACM Symp. on Comp. Geom.*, ACM Press, 274-280, 1993.

4 S. H. Lo, "Delaunay triangulation of non-convex planar domains," *Int. J. Numer. Meth. Engng.*, **28**, 2695-2707, 1989.

5 P. Moller and P. Hansbo, "On advancing front mesh generation in three dimensions," *Int. J. Numer. Meth. Engng.*, **38**, 3551-3569, 1995.

6 H. Samet, Applications of Spatial Data Structures, Addison-Wesley, 1990, pgs. 57-110.

7 C. Ozturan, H. L. de Cougny, M. S. Shephard and J. E. Flaherty, "Parallel adaptive mesh refinement and redistribution on distributed memory machines," *Comp. Meth. Appl. Mech. Engng.*, **119**, 123-137, 1994.

8 M. W. Beall, "Mesh data structures for advanced finite element applications," submitted for publication, *Int. J. Numer. Meth. Engng.*

9 Document for a Standard Message-Passing Interface, *Message Passing Interface Forum*, University of Tennessee, CS-93-214, November, 1993.

10 C. Ozturan, "Distributed environment and load balancing for adaptive unstructured meshes," Ph. D. Thesis, Computer Science Dept., Rensselaer Polytechnic Institute, Troy, NY, 1995.

11 H. L. de Cougny, M. S. Shephard, C. Ozturan, "Parallel three-dimensional mesh generation," *Comp. Syst. in Engng*, Vol. 5, No. 4-6, 311-323, 1994.

12 H. L. de Cougny, M. S. Shephard, C. Ozturan, "Parallel three-dimensional mesh generation on distributed memory mimd computers," submitted for publication, *Engineering with Computers*, 1995.