Parallel Unstructured Mesh Generation[1]

T. Okusanya
J. Peraire

Computational Aerospace Sciences Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

A strategy for the parallel generation of triangular unstructured meshes is proposed. Several point insertion algorithms originally developed for serial implementation are considered. These algorithms are suitably modified and procedures are developed for parallel execution. Dynamic load-balancing and mesh migration are incorporated to ensure even work distribution during the generation process. After the generation process is completed, the mesh is already partitioned and distributed for the analysis.

Introduction

Unstructured mesh methods for computational fluid dynamics have experienced a rapid growth over recent years and, for the computation of inviscid flows, have achieved a considerable level of maturity. As a result, a number of systems incorporating automatic mesh generators and flow solvers have been built which are currently being used in a semi-production mode by industry and research establishments [4, 6, 10]. The main advantage of the unstructured mesh approach is that, for complex geometries, it allows to significantly reduce the time period associated with the CFD analysis cycle. This feature is especially valuable during the early stages of design of aircraft when a large number of options needs to be analyzed.

The reasons for distributed mesh generation are twofold. The first one is to reduce the computational times required for mesh generation. It is clear that for the large grids required for viscous turbulent flow analysis, mesh generation may become a problem. More important however, are the memory requirements. At present, the size of a computation is determined by the ability to generate the grid on a serial machine. For instance, the generation of a tetrahedral unstructured grid containing 8 million cells requires a serial machine with 512Mb of core memory using the FELISA system [6]. Once generated, this grid needs to

[1]Submitted to the 5^{th} Int. Conf. on Numerical Grid Generation, January 1996

be partitioned and sent to the parallel processors to carry out the flow solution. Hence, a strategy capable of generating a grid in parallel and such that when the grid generation process is finished, the grid is already partitioned and in place for the flow solution, is desirable.

This paper addresses the development of a parallel unstructured mesh generator designed to operate on parallel MIMD machines with distributed memory. Early attempts at parallel mesh generation include Weatherhill [9], Löhner *et al* [14] and Saxena *et al* [7] who in general follow the rule of domain partitioning, independent meshing of the subdomains and finally meshing the inter-subdomain regions. In general, efficient parallel algorithms require a balance of work between the processors while maintaining interprocessor communication to a minimum. A major key to determining and distributing the work load is based on the knowledge of the nature of the type of analysis being performed. For mesh generation, the only knowledge available at the beginning of the process is the initial structure of the geometric model which has little or no relationship to the work required to generate the mesh. Hence these attempts may not be well suited if significant work load imbalance exists.

A more general attempt has perhaps been made by Shephard *et al* [1, 8]. Shephard *et al* discuss the development of an octree-based parallel three-dimensional mesh generator [1] which is later coupled to an adaptive refinement system and a parallel mesh database into the distributed mesh environment [8]. Full data spectrum for all geometric entities are stored in this implementation and hence may be expensive in terms of computational resources.

Mesh Generation Algorithms

The mesh generation strategy is based on a constrained Delaunay triangulation with respect to a discretization of the domain boundary. Two popular point insertion algorithms, originally developed for serial implementation, are considered. The first method due to Watson [2] is based on the circumcircle criterion, whereas the second method due to Green/Sibson [13] utilizes a recursive edge swapping procedure.

The interior point creation strategies of Rebay [15], Chew [12] and Weatherhill [11] have been implemented. In Rebay's algorithm, the new nodes are placed on the Voronoï segment whereas in Chew's and Weatherhill's algorithms, the new nodes are placed on the triangle circumcenters and centroids respectively. The interior points are inserted to meet an element size distribution which is specified by means of a *background mesh* and a *source distribution* [6]. The selected interior point creation stategies require an initial triangulation of the computational domain boundary. This initial triangulation is currently performed a priori by discretizing, using the background mesh, a CAD representation of the domain boundary using Ferguson splines.

Parallel Mesh Generation

In order to provide an environment where the algorithms previously described can be implemented in parallel, a number of data structures and procedures are developed. The parallel mesh generation process is based on a cycle of point insertion and load balancing operations. Points are inserted within each subdomain until a prescribed number of elements have been generated. The mesh is then balanced to ensure a better distribution of the work load. In this implementation, the number of processors coincides with the number of subdomains. The parallel implementation follows a message driven master/slave paradigm which allows for a highly concurrent implementation of the procedures involved.

1. Data Structure

The mesh is partitioned across processors such that each element belongs to a single processor. Interprocessor information is dealt with by having edges and points common to two adjacent subdomains shared (duplicated). This is not considered wasteful since for typical meshes considered, the surface area to volume ratio is usually quite small. The mesh generation procedures operate on this model of distributed grid representation. The concept of edge and point sharing is embodied in a data structure termed a *front segment* which is defined for each shared interprocessor edge and duplicated on each sharing subdomain. This is depicted in the 2-D context in figure 1. For a given interprocessor interface as shown in figure 1, the associated front on each sharing subdomain contains information regarding

1. The remote processor *Unique Identifier* (UID).
2. The remote identifier of the remote processor front segment.
3. The local vertex identifiers of the local front vertices, and the corresponding remote vertex identifiers of the remote front verrtices.

2. Mesh Migration

The load balancing and element request procedures described below require the transfer of elements between subdomains. The process of element migration from one processor to another is carried out in three stages. The first stage involves verification that the elements to be migrated can be transferred to the destination node. This requires that a *front lock* must be made on all other processors (excluding the sender and receiver) for all the fronts which will be affected by the migration. This is necessary to avoid situations, such as the simultaneous transfer of adjacent elements in different subdomains, in which the proper updating of the shared information cannot be guaranteed.

The second stage involves the transfer of mesh entities between the sender and receiver. This is depicted in figure 2 and involves packing the information regarding the boundary, topology and geometry of the migration element set. In order to determine the boundary and topology of the elements to be transferred, a Breadth First Search (BFS) algorithm is employed.

The third stage involves the update of the front segments on all the processors affected by the transfer excluding the sender and the receiver. In this last stage, the receiver processor issues appropriate messages to unlock the locked fronts on the remote processors which were originally locked by the sender.

The performance of the migration algorithm is illustrated in figure 4 which shows the migration throughput achieved for a typical example on the IBM SP2. The figure includes comparisons for the PVM and MPI message passing libraries and shows the CPU time required to migrate elements between two processors as a function of the number of elements. From this figure, the average throughput is seen to be approximately 5000 elements per second. It should be noted that the developed algorithm is not well suited to a highly fragmented element set as the most time would be spent in the BFS element packing.

3. Load Balancing

The dynamically evolving nature of the unstructured mesh during the mesh generation process creates a load imbalance among the processors. Dynamic load balancing is accomplished by means of *partitioning* and *dynamic redistributing* of the elements . The algorithm developed is based on a modified Coordinate Bisection. The partitioning of the domain into subdomain blocks is done using planes perpendicular to the cartesian axes. Processor assignment of the elements is based on the element centroid coordinates. The number of blocks ($\mathbf{n_X} \times \mathbf{n_Y}$) is based on the number of processors available and in such a way that $\mathbf{n_X}$ is as close to $\mathbf{n_Y}$ as possible. The partition separators are obtained by iteratively solving a linear system of equations derived from a spring analogy. Each block is represented as two linear springs in the x and y directions such that in each direction, the "spring constant" is proportional to the ratio between the number of elements in the block and block dimension along that direction. This linear system is depicted in the 1D context in figure 3. After the partition locations are obtained, the final destination of each element is determined and a migration schedule is set up to redistribute the elements. This algorithm is in some respects similar to the "Strip" partitioning algorithm developed by Vidwans *et al* [16].

A pathological case which is not dealt with easily by this method is a mesh which contains regions of densely clustered elements. In this case, the iterative procedure leads to an oscillatory solution which may require an unreasonable number of iterations to converge. This situation can be remedied by introducing band limiters which limit the minimum and maximum displacements for a partition within each iteration.

Benchmarking of the load balancing algorithm is done by generating a mesh on one processor, random assignment and migration of elements to subdomains followed by load balancing. It should be noted that random distribution represents the worst-case scenario for the load balancing algorithm. Figure 5a shows the CPU time required to balance 50000 randomly distributed elements as a function of the number of processors and figure 5b shows the CPU time required to balance the randomly distributed elements on 8 processors as a function of the number of elements. The total CPU times depicted have been decomposed into processor assignment times and element migration times. Figure 5a shows the expected trend of a decrease in the load balancing time with the number of processors and indicates scalability. The load balancing algorithm however appears to be slightly sensitive to the subdomain partitioning as indicated by the 5 processor (1×5) and 10 processor (2×5) data points of figure 5a. The low times required for the two processor case may be explained by noting that the migration algorithm simplifies considerably when only two processors are involved.

4. Element Request

The selected point insertion algorithms need to be modified slightly to take into account the nature of the parallel mesh generation process. This involves the inclusion of a procedure to obtain the global element set which will be affected by the insertion of a newly created point [2, 13]. This is required by the point insertion algorithms to guarantee the mesh is locally Delaunay. The initiator of the request may be either granted or denied remote elements by any affected processor based on the existence of an overlap between any remote topological entity in use and the requested elements.

The general procedure is initiated given an initial set of local elements $\{\Omega_i{}^r\}$ affected by a newly created point. Request messages are propagated to the subdomains which lie across any processor front elements attached to $\{\Omega_i{}^r\}$. From the point of view of a processor in the request processor set, an ADT tree search [5] is performed on the local element tree to obtain the required elements $\{\Omega^t\}$. In order to maintain mesh consistency, a front lock request must be made on all front elements attached to $\{\Omega^t\}$. If the elements requested are not in use and locks are granted by the remote processors, the request initiator is granted the element set $\{\Omega^t\}$. If this is not the case, the initiator is denied that set of elements and must consider the insertion of a new point.

Parallel Mesh Generation Results

The above described algorithms are incorporated into a parallel mesh generation code and the performance obtained for a typical case is depicted in figure 6(a-d). This example uses the Green/Sibson point insertion and the Rebay

point creation algorithms. Figures 6a and 6c show the mesh generation timings based on average and maximum processor CPU times respectively as a function of the number of processors for 1 million elements without a final fine-resolution load balance. Figures 6b and 6d show the mesh generation timings based on average and maximum CPU times respectively for 1 million elements with a final fine-resolution load balance. The total CPU times have been decomposed into meshing, load balance, element request and polling times. The latter makes reference to the time the processor is idle waiting for messages. All figures show the expected trend for the total generation times and show linear scaling for the actual time spent in mesh generation. As seen from the figures, the time for actual mesh generation decreases with the number of processors but the load balancing time increases with the number of processors. Hence some optimal processor size (element granularity) exists which minimizes the total mesh generation time for a fixed number of elements.

Figures 7(a-d) show a 4 processor example of load balancing during the process of mesh generation. Figures 7a and 7b depict an intermediate stage in the mesh generation about a NACA-0012 airfoil before and after load balancing respectively. Figures 7c and 7d show the final grid before and after load balancing respectively.

Conclusions

An approach to parallel unstructured 2D mesh generation has been developed for operation on distributed memory MIMD computers. The developed algorithms include parallelization of the point insertion schemes and the inclusion of element migration and dynamic load balancing to ensure even work load distribution. The algorithms have been implemented on an IBM SP2 system but are easily portable to other platforms supporting standard communication libraries. Scalability of the mesh generation algorithm has been illustrated based on timings for different processor sizes. The algorithms have been developed for 2D but are extendable to 3D and this is the focus of our current effort.

Acknowledgements

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Computing time on the IBM SP2 was provided by Massachusetts Institute of Technology.

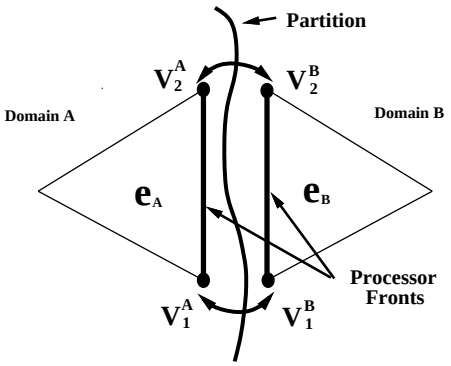


Figure 1: Interprocessor Front

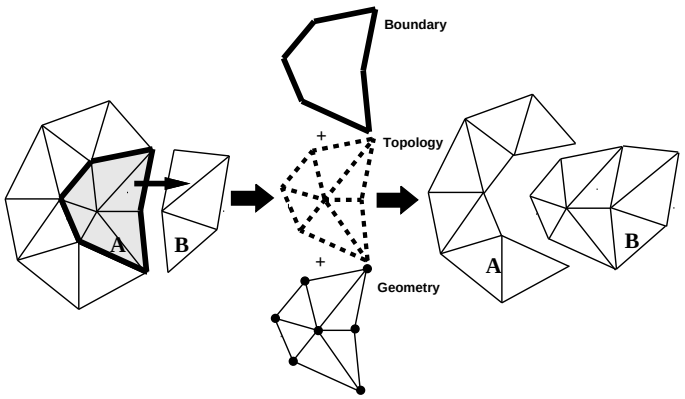


Figure 2: Element Migration Philosophy

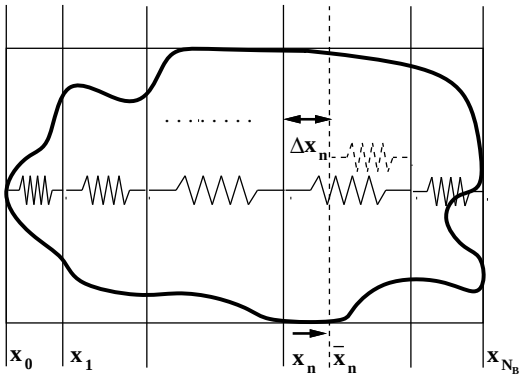


Figure 3: Linear 1D Load Balance System

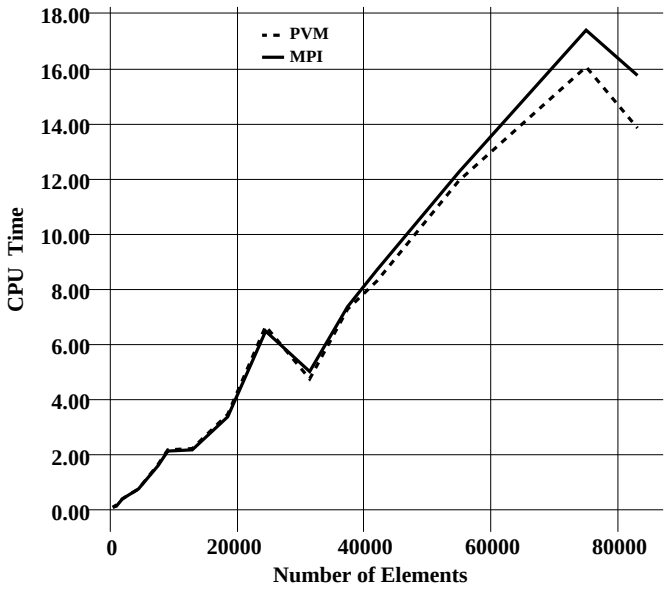
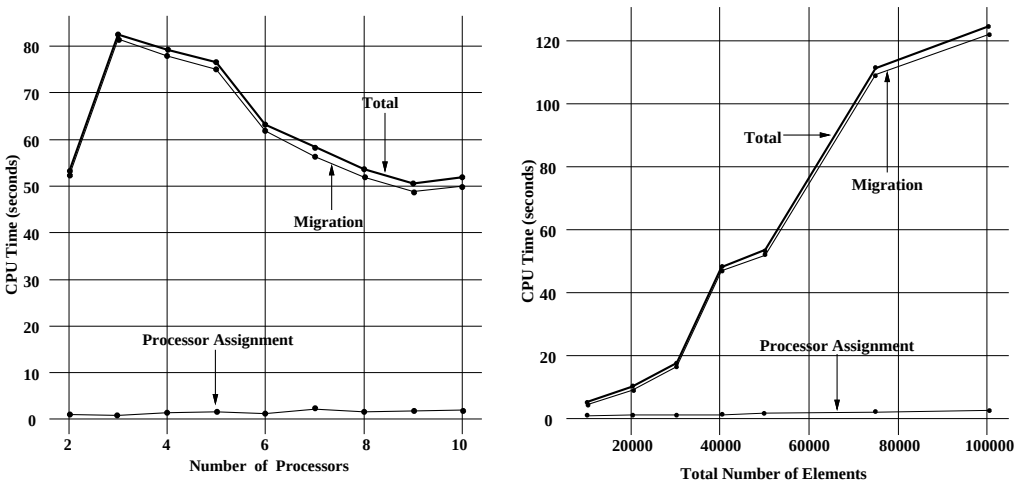
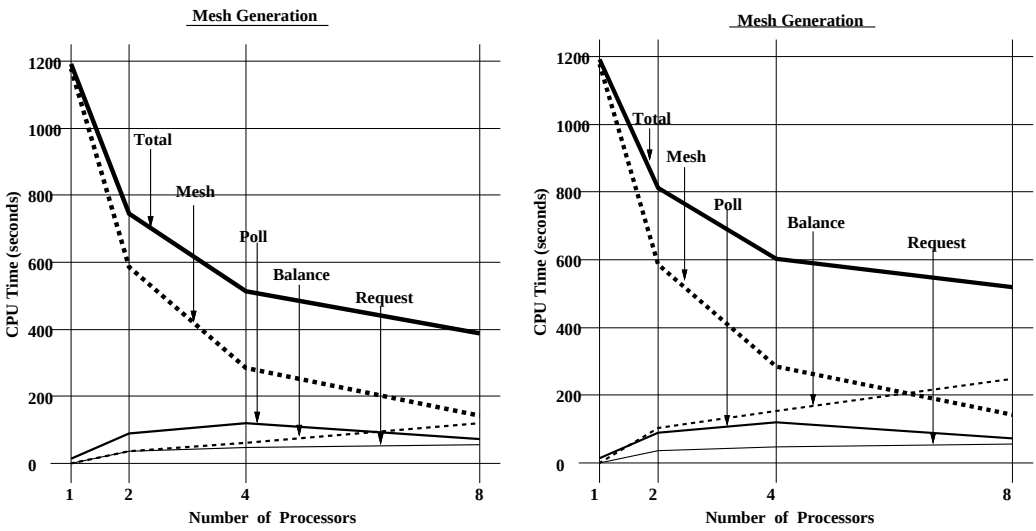


Figure 4: Mesh Migration Throughput



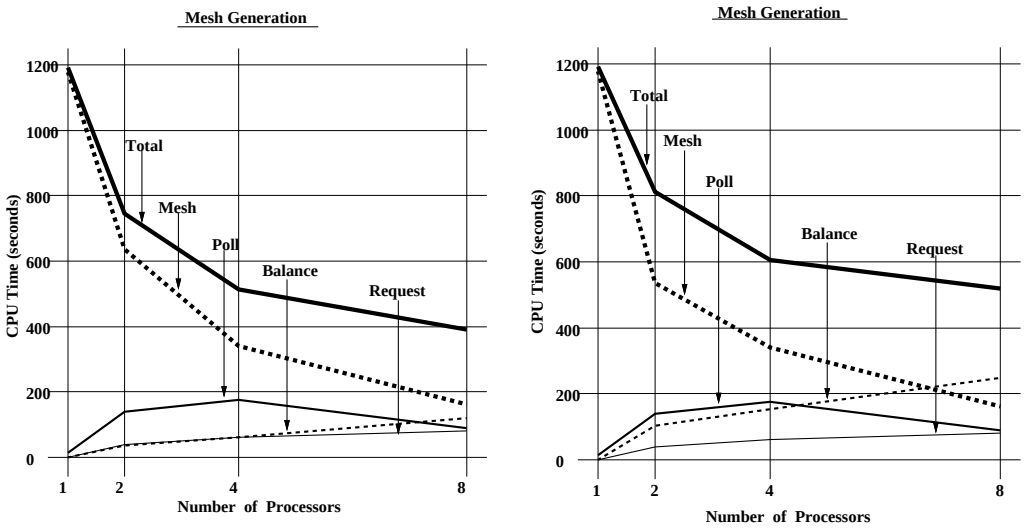
(a) 50000 Elements (b) 8 Processors

Figure 5: Load Balancing: Random Element Assignment



(a) Avg w/o Final Balance

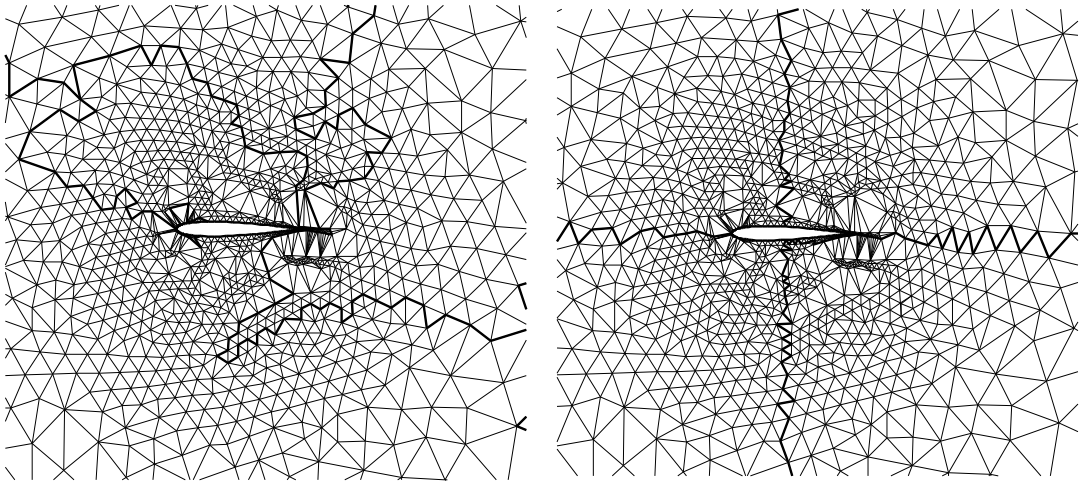
(b) Avg w/ Final Balance



(c) Max w/o Final Balance

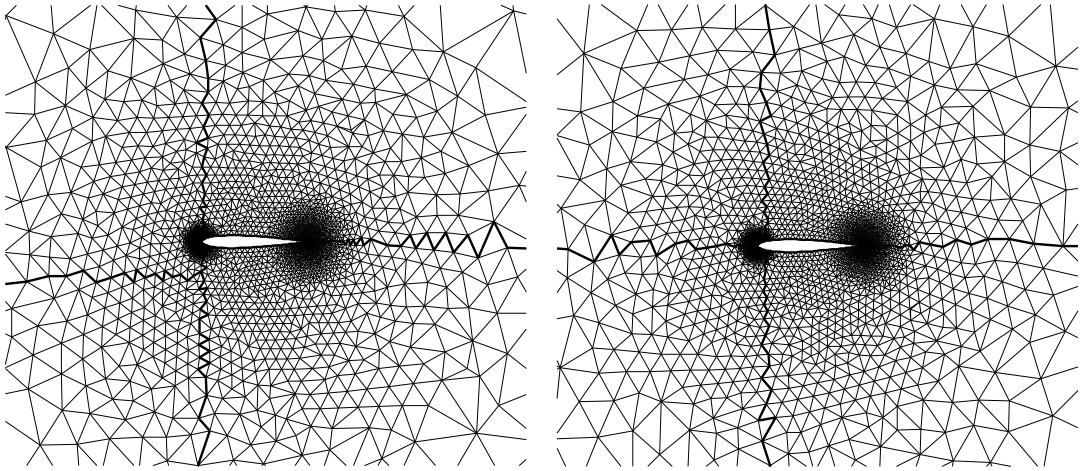
(d) Max w/ Final Balance

Figure 6: Mesh Generation of 1 Million Elements



(a) Midway Unbalanced

(b) Midway Balanced



(c) Final Unbalanced

(d) Final Balanced

Figure 7: 2D Parallel Mesh Generation - Four processor illustrative example

References

[1] H.L. de Cougny, M.S. Shephard, and C. Ozturan. Parallel Three-Dimensional Mesh Generation on Distributed Memory MIMD Computers. Technical Report SCOREC Report #7, Rensselaer Polytechnic Institute, 1995.

[2] D.F.Watson. Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes. *The Computer Journal*, pages 167–171, 1981.

[3] Farin Gerald. *Curves and Surfaces for CAGD.* Academic Press, Inc., 1988.

[4] A. Jameson. Computational Algorithms for Aerodynamic Analysis and Design. *Applied Numerical Mathematics*, pages 383–422, 1992.

[5] J.Bonet and J.Peraire. An Alternate Digital Tree for Geometric Searching and Interaction Problems. *Intl. Journal Num. Methods in Eng.*, 1995.

[6] J.Peraire, J.Peiro, and K.Morgan. The FELISA system. An unstructured mesh based system for aerodynamic analysis. Technical report, Computational Aerospace Science Laboratory/M.I.T, 1983.

[7] M.Saxena and R.Perruchio. Parallel FEM Algorithms Based on Recursive Spatial Decompositions. *Computers and Structures*, pages 817–831, 1992.

[8] M.S.Shephard, J.E.Flaherty, H.L. de Cougny, C.Ozturan, C.L.Bottasso, and M.W.Beal. Parallel Automated Adaptive Procedures for Unstructured Meshes. *Parallel Computing in Computational Fluid Dynamics, AGARD-FDP-VKI*, 1995.

[9] N.P.Weatherhill. The Delaunay Triangulation - From The Early Work in Princeton. *Frontiers of Computational Fluid Dynamics*, pages 83–100, 1994.

[10] N.T.Frink, P.Parikh, and S.Pirzadeh. A Fast Upwind Solver for the Euler Equations on 3D Unstructured Meshes. *AIAA*, 1991.

[11] N.Weatherhill and O.Hassan. Efficient three-dimensional grid generation using the Delaunay triangulation. *1st Euro. Comp. Fluid Dyn. Conf., Brussels*, 1992.

[12] P.Chew. Mesh generation, curved surfaces and guaranteed quality triangles. Technical report, IMA Workshop on Modeling, Mesh Generation and Adaptive Num. Meth. for Part. Diff. Eqs, Univ. Minnesota, Minneapolis, 1993.

[13] P.J.Green and R.Sibson. Computing the Dirichlet Tessellation in the Plane. *The Computer Journal*, pages 168–173, 1977.

[14] R.Löhner, J.Camberos, and M.Merriam. Parallel Unstructured Grid Generation. *Comp. Meth. Appl. Mech. Engg.*, pages 343–357, 1992.

[15] S.Rebay. Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm. *Journal of Computational Physics*, pages 125–138, 1993.

[16] A. Vidwans, Y.Kallinderis, and V. Venkatakrishnan. Parallel Dynamic Load-Balancing Algorithm for Three-Dimensional Adaptive Unstructured Grids. *AIAA*, pages 497–505, 1994.