Adaptive Parallel Meshes with Complex Geometry

Roy Williams

Concurrent Supercomputing Facilities, California Institute of Technology, Pasadena CA 91125

Abstract

We discuss the automatic creation and adaptive refinement of an unstructured mesh within a complex geometry such as the space surrounding an airplane. This may be formulated as two distinct parts; a non-parallel part requiring global knowledge which automatically creates a coarse compatible mesh, and a parallel local refinement algorithm, which refines the mesh until simulation can begin, then adaptively refines it according to the progress of the simulation.

Background-mesh methods, sequential and parallel, offer some promise if good numerical algorithms are available. Sequential advancing front methods combined with the parallel Rivara refinement algorithm are a good choice.

Introduction

The computer solution of a complex set of spatial partial differential equations such as those of fluid dynamics requires a definition of the domain in which the problem is to be solved, together with a set of boundary conditions to be applied at the boundaries of this domain. The interface between such a domain specification and a numerical simulation is usually accomplished with a computational mesh [1-4].

The purpose of the mesh is to provide a framework in the problem domain for the storage and manipulation of physical data. There are two distinct phases to creating a mesh, these being global/sequential and local/parallel respectively.

The global part of the mesh creation involves resolving the topology of the problem domain by splitting it into a small number of domains, each of simple topology and geometry, such as hexahedra or tetrahedra. This procedure requires global knowledge of the geometry, is logic intensive, and deals with a small amount of data, and is thus suitable for a sequential machine.

Having split the problem domain into a connected set of simple shapes, these may themselves be split into smaller shapes, thereby refining the mesh to a scale suitable to begin the numerical simulation, and incidentally providing a local adaptive refinement procedure. This refinement is local, requiring data only from a mesh entity and its immediate neighbors, and thus eminently parallelizable.

Mesh Requirements

A computational mesh is, for the purposes of this paper, a collection of closed polyhedral sets covering the problem domain, called *elements*; the intersections of these are *faces*, *edges* and *nodes*, of dimensionalities 2, 1 and 0, respectively. We may refer to these collectively as

mesh entities. The mesh is *proper* if the intersection of any pair of distinct mesh entities of the same dimensionality is a mesh entity of lower dimensionality or is empty.

In general there may be simulation data associated with each mesh entity, and also pointers to neighboring mesh entities. For example a face may have pointers to the edges surrounding it, or an element to the elements with which it shares faces.

The software which manages such a mesh must try to satisfy the perhaps conflicting requirements from four distinct sources:

- The Investigator,

would like a numerical code which is fast and accurate. The meshing software should be capable of automatically meshing a complex geometry, and adaptive so that computational resources may be concentrated where necessary within the domain. In addition, it should be reasonably easy to make changes to the solution algorithm, and the code should be portable to different parallel machines.

- The Machine

on which the code is to run will in general have multiple processors. The machine will also have a hierarchical memory structure, consisting of cache, main memory, the memory of other processors and external devices such as disks. The goal here is to minimize the communication rate and latency costs associated with data transfer between the processors and the memory units.

- The Numerical Algorithm

is easiest to formulate and code when the elements of the mesh are all the same type of polyhedron, for example tetrahedra or hexahedra. Also the algorithm should be able to feed back its requirements for element shape: in special parts of the mesh, such as a boundary layer, it may be desirable to have flat elements whose short dimension is normal to the boundary layer.

- The Geometry

of the domain may place requirements on the mesh simply from its specification, whether it is described as a set of surfaces or a set of volumes. The domain may have multiple holes of spherical or cylindrical topology, periodic or reflective boundaries, sharp edges and corners, and a multiplicity of boundary conditions.

In this paper we shall consider how these considerations affect the design of a three-dimensional, unstructured parallel mesh manager, so that a solver may be coded for a complex set of equations such as the Euler or Navier-Stokes equations.

We shall assume that a mesh is constructed by first automatically making a coarse mesh which is compatible with the topology of the problem domain. This meshing takes place on a sequential machine. The coarse mesh may then be loaded into the processors of a parallel machine, dynamically adapted and load-balanced among the processors of the machine.

In the following when discussing parallelism we shall assume that mesh entities communicate by message-passing, which is an implicit assumption that each mesh entity is managed by a separate processor. Of course for an actual implementation, a processor would own many mesh entities, and ‘message-passing’ to an entity in the same processor would reduce to a memory access.

Structured and Unstructured Meshes

Meshes may be logically structured, partially structured or unstructured. A logically structured mesh is a regular array of mesh entities, with each type of mesh entity having the same connectivity to corresponding mesh entities. The mesh may in addition be crystalline, such that the mesh entities are laid out like a crystal in a geometrically regular way.

A partially structured or macro-unstructured mesh is one consisting of many small structured meshes connected in an unstructured way. We may think of each of these structured meshes as a “super-element”, so that a partially structured mesh is equivalent to an unstructured mesh of super-elements.

If these super-elements are large, then we may take advantage of vector machines which work especially well with structured data, as well as reducing latency costs associated with message-passing, because inter-element messages are longer. The partial structuring also means that less memory need be taken with connective and geometric data in exchange for more simulation data.

On the other hand, large structured super-elements mean less definition of the boundaries of the problem domain, greater distortion of the element shape, and a more difficult job for the software making the initial mesh. Thus there is an optimum size of these super-elements. From now on we shall implicitly assume that an unstructured mesh is actually a mesh of partially structured super-elements.

Multiblock

The *multiblock* method [5-7] uses a partially structured mesh, but the assumption is that there is a small number of very large, structured, usually hexahedral elements. The elements have been carefully placed in the problem domain by a human operator with a graphics workstation, and the internal curvilinear meshing of each block fitted to its requirements. Such meshes may effectively use flexible and trusted finite-difference algorithms, which are eminently parallelizable.

Unfortunately the creation of a multiblock mesh requires expensive human resources, whereas meshes with a larger number of smaller elements are more easily created automatically. Furthermore, it is difficult to locally adapt the multiblock mesh to the emerging solution of the physical problem. Quadtree methods [8] have been used for this, but at the expense of complicating the algorithm structure.

Henceforth we shall consider methods for automatic creation and adaptation of the mesh; we shall assume that there is a preparatory sequential program which creates a coarse mesh that is compatible with the problem geometry, followed by a parallel program which alternates between simulating the physical problem and adapting the mesh.

Before describing methods of creating this coarse mesh in a complex geometry, we must define what is to be produced and the geometry from which it comes.

Geometry Specification

Analogously to the different dimensionalities of mesh entity which constitute the mesh, the geometry of the domain to be meshed consists of a *volume*, some curved *surfaces* bounding the volume, some *curves* bounding the surfaces and some *vertices* at the ends of the curves. We may refer to these collectively as *model entities*.

As well as simulation data, mesh entities should also keep information about their status with respect to boundaries; so that nodes may be classified as belonging to a volume, surface, curve or vertex. Similarly edges may belong to a curve, surface or volume and faces may belong to a surface or a volume.

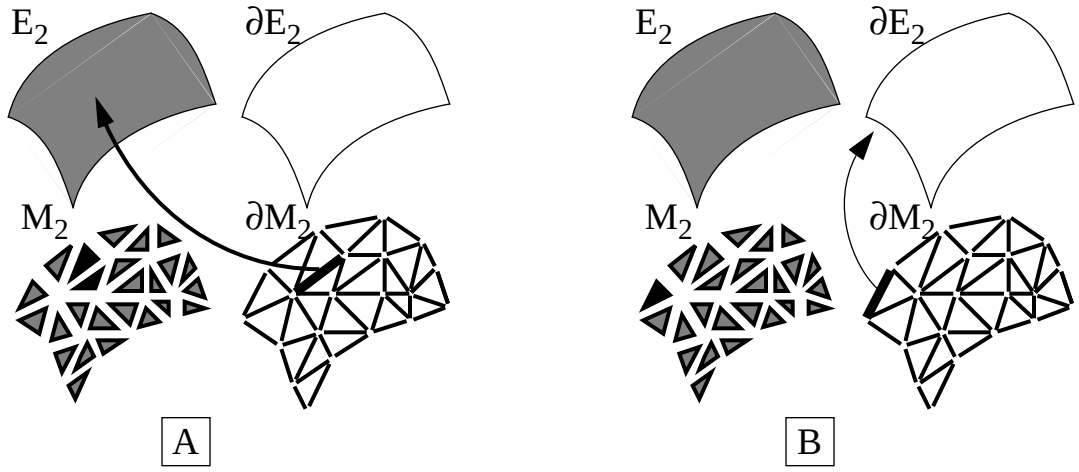

Figure 1: In a compatible mesh, for each edge, either A: the edge is part of the surface and has two neighboring faces, or B: the edge is part of the boundary of the surface and has one neighboring face.

The difficulty of mesh-making is the creation of a topologically *compatible* mesh [9, 10], defined as follows:

- Let E_d be a model entity of dimension d with ∂E_d its boundary model entity of dimension d-1. Let M_d be the set of mesh entities classified on E_d and ∂M_d the set of boundary entities of M_d. Then the mesh is *compatible* with E_d if for all $x \in \partial M_d$ either:
  $x \in E_d$ and exactly 2 mesh entities in M_d share x, or
  $x \in \partial E_d$ and exactly 1 mesh entity in M_d shares x.

This is illustrated in Figure 1 for the case $d = 2$, so that E_d is a surface and ∂E_d its surrounding curves, M_d is a set of faces and ∂M_d a set of edges.

We discuss below two methods of specifying domain geometry, where the geometry is considered either to be a collection of surface patches, or as a combination of volumes.

Atlas of Charts

An atlas of charts is a collection of differentiable mappings from sub-manifolds Ω_i of two-dimensional space $\Re^2$ to three-dimensional space $\Re^3$. This is analogous to making a football by patching together a number of patches of leather, each of which is brought from a flat state into three dimensions.

In addition to this set of surfaces, we would like a decision function which decides if a given point in $\Re^3$ is inside or outside the simulation domain. It may be possible to decide this automatically by finding the number of intersections between the set of surfaces and a line extending to infinity, though such a procedure is difficult to make robust.

In CFD applications, we would like our problem domain to be a volume, rather than just a set of surface patches in space. To define a surface which is the boundary of a volume, we

must also require in the above definition that for each point on $\partial\Omega_i$ there is another point on some $\partial\Omega_j$ which maps to the same point in $\Re^3$, or in other words, the edges of each patch must be joined to the edges of a patch (which may be the same one) so that our football does not have holes. A major difficulty with this method of geometry specification is that it is difficult to decide whether a geometry specification is even valid; even if the surface definition is valid, finite precision arithmetic may render it apparently invalid, causing difficulty for an automatic mesh maker.

Combinatorial Solid Geometry (CSG)

The CSG approach to geometry definition begins with a set of primitive volumes and allows these to be combined by the Boolean operations intersection and union to create further volumes. Each primitive volume is associated with a function which maps $\Re^3$ to $\Re$, and the volume is defined to be the set of points which the function maps to a positive quantity.

The atlas-of-charts definition of a domain is an explicit statement of the set of points at the surface of the domain, but it is difficult to decide whether a given point is inside or outside the domain. On the other hand, such a decision is easily made for the CSG definition, but we have no example of a point which actually lies in the domain or on its surface. Thus to be useful the CSG description should be supplemented with a set of seed points, such that no pair of these may be connected without crossing a surface. Without these, the mesh maker may never be sure that it has found all the surfaces; finding a domain where the function is greater than zero is much easier given a seed point in the domain.

Mesh Creation

There seem to be three classes of methods for converting a geometry specification into a compatible mesh, these being based on the Delaunay triangulation, advancing front methods, and background-mesh methods.

Delaunay triangulation uses a set of points chosen from the boundaries and interior of the problem domain, makes a high-quality mesh, then checks to make sure that this mesh is compatible. Advancing-front methods increment the dimension of the mesh; we make a 3D mesh from the 2D surface mesh by ‘growing’ elements from it. The background-mesh methods start with a space-filling regular mesh, then distort and enrich it to accommodate the boundaries.

Delaunay Triangulation

The Delaunay triangulation [1, 11-16] in its simplest form takes a set of nodes in $\Re^3$ and produces an almost unique mesh of tetrahedra filling the convex hull of the nodes. An algorithm for this is presented in a later section. Two properties of the triangulation are:

- the circumsphere of the four nodes of any element does not properly contain any other node,
- if each node is associated with a number, and *f(x)* is the linear finite-element interpolation function derived from these numbers and the triangulation, then the Delaunay triangulation minimizes

$$I = \int (\nabla f)^2$$

This latter property [17] implies a certain optimality in solving elliptic systems with the mesh: if we solve Laplaces' equation with linear finite elements, we are minimizing I with respect to the field-values at the nodes, and the use of the Delaunay rather than another triangulation additionally minimizes I with respect to the triangulation.

The general approach to mesh making is to choose a set of nodes, each classified according to model entity, make the Delaunay triangulation and then classify the edges, faces and elements. If there are elements of poor shape, we may incrementally add nodes at well-chosen places [18] to produce more but better-shaped elements. Jameson et al.[11] use the vertices derived from separate structured meshes about the various components of the geometry.

Although the Delaunay triangulation suffers from poor efficiency in parallel, as discussed below, the sequential implementation takes time of order $N^{4/3}$ (or $N\log N$ with additional software and memory overhead). However if we use the Delaunay triangulation only for the creation of an initial mesh with a sequential machine, this is not a problem.

The Delaunay triangulation is just a mesh, not necessarily a compatible mesh, as shown in Figure 2, where the model curve surrounding the shaded area with closed-loop topology is

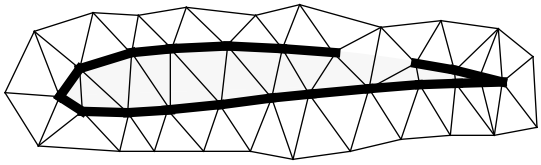

Figure 2: A non-compatible mesh produced by the Delaunay triangulation.

different from the open-loop topology of the set of edges classified to lie on the curve.

One way to avoid this non-compatibility is to either override the Delaunay property [14], or put a sufficient density of nodes on the surfaces and curves of the model, and make sure that no volume node is too close to the surfaces and curves [11]. We can try a given set of nodes, and if the mesh is not compatible enrich the set of nodes until the mesh is so.

Advancing Front Methods

Advancing front methods [1,19-21] 'grow' the mesh around the surfaces of the problem domain. We start with a geometry specification based on the atlas of charts, and triangulate all the surface patches in $\Re^2$ before mapping them to $\Re^3$. to provide the seed surfaces for the method. When these surfaces are mapped, we have a set of nodes, edges and faces with their proper classification to the model entities, but no elements. These model entities constitute the surface of a polyhedron, and advancing front methods incrementally place elements at the inside surface of this polyhedron, thereby making a smaller polyhedron with fewer faces. The computational effort comes from deciding where to put the next element, and from checking that the new element lies entirely within the polyhedron.

It may not be possible to add an element and thereby reduce the number of faces of the polyhedron; it may be necessary to add several elements which in aggregate reduce the number of faces. Also the polyhedron may have regions where two surfaces are much closer together than the desired mesh spacing which are thus difficult to fill with elements of reasonable shape.

An advancing-front mesh maker may rely on sophisticated heuristics to decide on where to put the next element [19,20], or we may use a more deterministic method [21], derived from the Deluanay triangulation, as follows.

We search through all the faces; for each face we search through the nodes looking for the minimum circumradius of the sphere passing through the node and the vertices of the triangular face. As long as this tetrahedron has no intersections with any boundary faces, and lies inside the problem domain, it is added to the mesh. Eventually the domain is filled with a compatible mesh, and we may improve the mesh by adding volume nodes, so long as they do not make the mesh incompatible.

Background Mesh Methods

These methods [9, 22-24] are based on the idea of surrounding the problem domain with an enclosing crystalline or octree mesh, and using this structure as the mesh, except near the boundaries.

We should note that the mesh far from the domain boundaries is either crystalline or octree, and this is presumably the majority of the mesh; we might hope to find an algorithm that can use this regularity effectively for algorithmic efficiency, yet also work with the unstructured part of the mesh near the boundaries where the mesh has been distorted to fit. In the crystalline case this efficiency could come from vector processing hardware, and in the octree case that there is a natural multigrid structure available for an algorithm that can use it.

To fit the mesh to the boundaries, we may move nodes from the regular mesh to the domain vertices, curves and surfaces, presumably moving sufficiently close nodes to the point on the surface closest to the node.

Alternatively or in addition we may add extra nodes to the mesh where

- a model surface intersects a mesh edge,
- a model curve intersects a mesh face,
- a model vertex.

A mesh created by addition of surface nodes is shown in Figure 3.

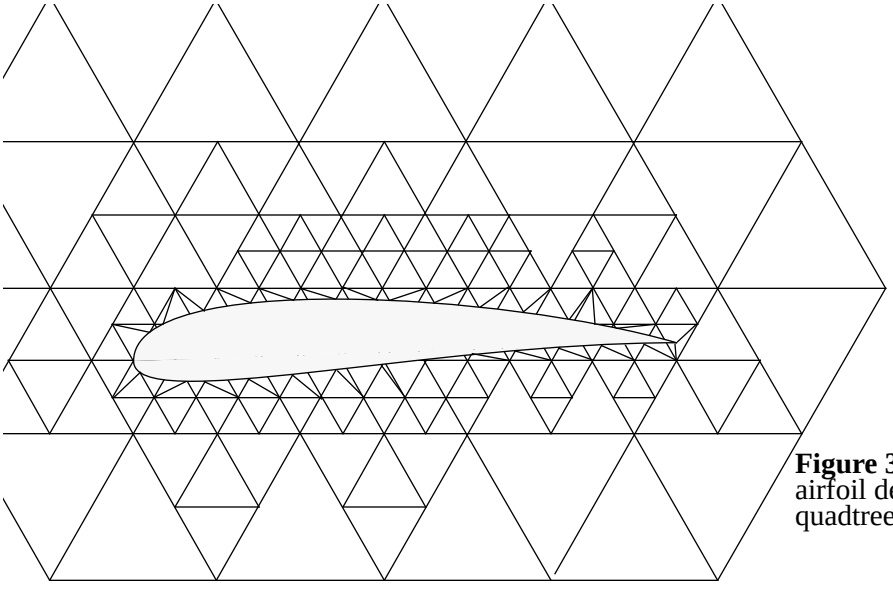

Figure 3: A mesh surrounding an airfoil derived from a triangular quadtree.

The problem with this approach to mesh creation is that in many applications, such as CFD, the quality of the solution depends critically on the correct treatment of boundaries, which is where the mesh of Figure 3 has particularly badly shaped elements. While there are curative methods to improve the mesh, the essential problem is that the mesh is distorted to fit the boundaries, rather than first laying out the boundaries and creating the mesh to accommodate them.

Once the coarse mesh is made by a sequential machine, it may be loaded into a single processor of the parallel machine, then split into pieces and distributed among the processors. We now alternately run the simulation and adapt the mesh, based either on automatic or manual decisions. Thus we need to be able to do two things: adapting, discussed below, and load-balancing [25].

Adapting the Mesh

There are two distinct reasons for which the mesh may be adapted; either to obtain better resolution of a particular solution of the physical equations, the *steady* case; or to better resolve transient phenomena such as shocks in a time-varying solution, the *unsteady* case. Consider the density of simulation data, which is the quantity of data in a unit geometrical volume. The adaptation may be loosely characterized by the extent to which a particular adaptation strategy may change this data density.

For unsteady adaptation, both increase and decrease in the density of simulation data are necessary, since there will be previously adapted regions of the problem domain which no longer contain the transient phenomena. The data density will not vary by more than an order of magnitude or so; and the adaptation should cause minimal numerical error to appear in the solution because such error is propagated and retained during the rest of the simulation.

Steady adaptation has opposite requirements. There may be many orders of magnitude difference in data density both from one area of the mesh to another; no mesh coarsening is necessary; and we may allow the adaptation may introduce some numerical diffusion, because the simulation will presumably converge to the correct steady solution even if error is introduced on the way.

Another desirable feature for an adaptation method would be the ability to do *directional refinement* [15, 19], as illustrated in Figure 4. Many physical systems, such as the Navier-

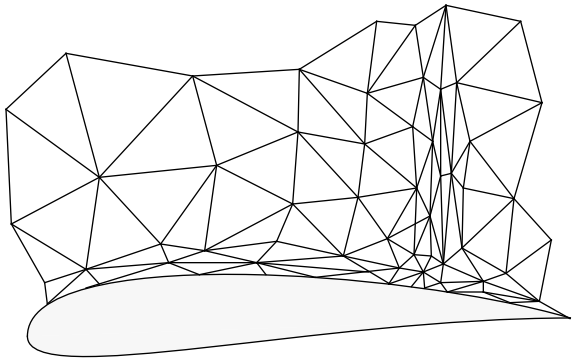

Figure 4: A mesh which is directionally refined at a boundary layer and also at a shock.

Stokes equations, exhibit boundary layers, where simulation data changes rapidly across the layer but slowly along the layer; so we would like adequate resolution in one direction without wasting resources in the other directions.

There are essentially three methods for adapting the mesh, either to change the geometric structure (mesh movement), to change the data and algorithm executed by a mesh entity (*p*-refinement), or to change the topological structure of the mesh (*h*-refinement).

Mesh movement

Mesh movement is an easy way to produce moderate changes in data density, and is suitable for unsteady adaptation. But when we try to make large changes, the elements tend to get highly distorted. The software overhead is low because decisions are made by solving an elliptic system, and the software for this may be included in the simulation code, rather than by software-intensive logic and communication.

p-refinement

p-refinement may be thought of as increasing the amount of data associated with an element without changing the mesh. A global adaptation may be achieved by, for example, increasing the order of the finite-elements employed, or by increasing the size of each structured mesh associated with each element.

Local adaptation by *p*-refinement is rather more difficult; if we change some finite elements from linear to quadratic for example, there must also be several types of transition elements whose faces and edges are partly linear and partly quadratic. This method also reduces the flexibility of the simulation code, since the numerical algorithm must be stated and evaluated not just for a single type of element, but for a whole class of elements, plus all possible transition elements between the members of the class.

We shall henceforth concentrate on topological methods for enriching the mesh, since these offer the ability to arbitrarily adapt the data density of the mesh. Furthermore, we shall only consider tetrahedral elements, since these simplest polyhedra reduce software complexity to a minimum.

h-refinement

One way to enrich the mesh is of course to make a completely new mesh which is adapted in the correct way, using the old mesh for interpolation of simulation data and as a framework for defining the space of the problem domain.

Unfortunately the new mesh must be made in parallel, since the mesh and simulation are distributed when the adaptation occurs. The mesh-creation methods discussed in the previous section all offer little scope for parallelism, and sending the old mesh then receiving the new to and from a sequential machine would be most inefficient.

In the following sections, we shall consider some methods for parallel local topological adaptation of a tetrahedral mesh.

Local Topological Adaptation

Delaunay Triangulation

Bowyer’s algorithm for Delaunay triangulation [12] adds new nodes sequentially to an existing mesh, and this method has been used successfully for adaptation by a number of workers, albeit with sequential machines. It seems difficult, however, to efficiently add nodes

in parallel [26] . Figure 5 shows the operation of Bowyer's algorithm; when a new node is to

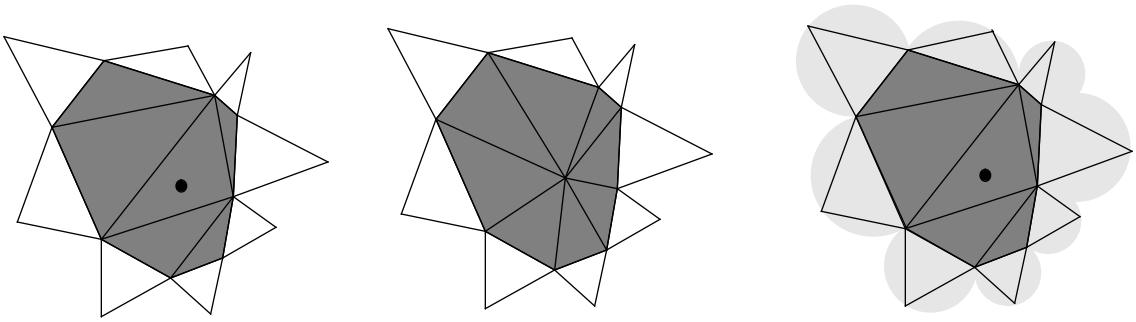

Figure 5: Bowyer's algorithm for Delaunay mesh refinement. Left: The original mesh with a new node about to be added and the influence domain of the new node. Middle: the structure in the influence domain replaced by radial elements from the new node, Right: the region where another node may not be added in parallel

be added, we first find the element in which the node lies, then all surrounding elements whose circumsphere contains the new node, which is the influence domain. All structure is removed from the influence domain and replaced by a set of elements each of which has the new node as a vertex.

If the influence domains of two nodes overlap, then these nodes may not be added in parallel; indeed the influence domains must be further separated by a guard element, as shown in Figure 5.

A possible parallel algorithm could work as follows [26]. First we define a tree structure for communication within the mesh: a parent element communicates to its four neighboring elements, and each of these four children may either pass the message to its neighbors, then wait for acknowledgment, or acknowledge to its parent. In this way we can map out the influence domain of a new node: an element passes on the message about the position of the new node or simply acknowledges the message depending on whether or not its circumsphere contains the new node.

But we must make sure the influence domain is not overlapping the influence domain of some other node being added in parallel. Again the messages pass through the tree and back, and if there is a conflict, there is an arbitration between the new nodes, and the loser is put into temporary memory so it may be added to the mesh later.

Elements in the influence domain may now generate new elements to fill the influence domain, and delete themselves, completing the addition of some of the new nodes. The whole cycle is repeated, using the nodes that were placed in temporary memory, if any, until all the new nodes have been integrated into the mesh.

An implementation of this algorithm on an Ncube parallel machine yields speedups of 1.1 to three on 16 processors; this is a speedup compared to the parallel code running on a single processor. However the best sequential algorithm uses not message-passing but recursion for the tree-like search operations, and need not do conflict checking. Comparing with this optimal sequential version, parallel Delaunay triangulation achieves speedups of 0.1 to 0.3 on 16 processors. The problem is that the influence domain of a particular node may extend arbitrarily far and thus overlap the influence domains of arbitrarily many other new nodes.

Rivara Refinement

We would like to have a local refinement strategy which keeps the changes to the mesh local, or at least propagates such changes in well-defined loosely synchronous stages; such a refinement method for tetrahedral meshes was suggested by Rivara [27], and is illustrated for triangles in Figure 6.

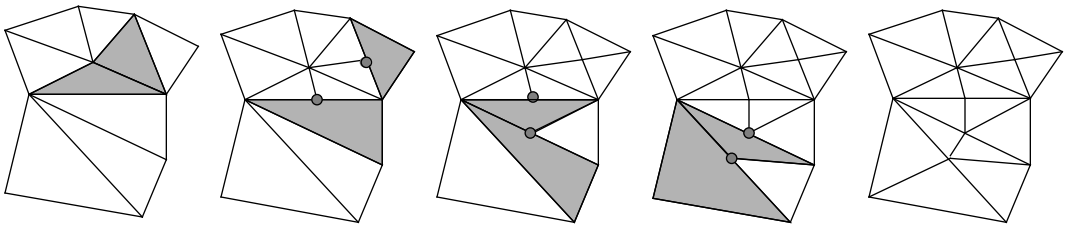

Figure 6: Stages in the Rivara local refinement method.

A set of elements are nominated for refinement, shown shaded at the top left of the Figure. For each of these elements, a new node is created at the midpoint of the longest edge of the element, so long as a new node has not already been created on that edge by another element. This new node is *nonconforming*, shown by blobs in the Figure, and other elements (those that share the refined edge) are now marked for refinement at the next stage (top right panel). The process continues: refinement of the marked elements along the longest edge, and marking of elements which share those refined edges. The algorithm eventually finishes, leaving a mesh of well-shaped elements, assuming the starting mesh has well-shaped elements.

Conclusions

Mesh creation has two distinct functions: global and local. The global, or sequential, part is the conversion of a geometry specification to a compatible unstructured mesh. This mesh must contain enough elements to resolve the topology of the problem domain: that is to split the domain into a set of connected subdomains each with the simple topologies: sphere, disk, line or point.

The local, or parallel, part of the mesh creation/adaptation process consists of refining some of these simple mesh entities to enhance the resolution of the physics we are trying to simulate. This refinement must retain the compatibility of the mesh to the geometry, and maintain or improve the geometrical quality of the mesh entities.

The initial mesh may be created with expensive human time, with the aid of sophisticated software and hardware. The resulting high-quality mesh may be used by efficient and parallel numerical algorithms, but is difficult to adapt locally. We have discussed three methods for automatic creation of the initial mesh:

- Delaunay triangulation creates a high-quality mesh, but it is difficult to ensure compatibility and element quality.
- Background-mesh methods create a mesh which is excellent far from the boundaries, but element quality may be poor at the boundaries. A regular background mesh offers scope for vectorization, and an octree implementation offers an easy route to powerful multigrid algorithms.

- Advancing-front methods require sophisticated heuristics, but can create a compatible mesh with good element quality at the boundaries.

In CFD applications, the correct treatment of boundaries is crucial for accurate flow simulation. Either we need a numerical algorithm which is not sensitive to element quality, or use a mesh creation method which ensures element quality at the boundaries. The latter is possible with a good selection of nodes for input to the Delaunay method, or by using an advancing-front method.

Local parallel refinement of a tetrahedral mesh may be achieved by the Rivara refinement algorithm, which does not improve the quality of the existing mesh and thus relies on the original mesh being of good quality.

The Delaunay refinement might be used for coarse-grained parallel machines by simply adding nodes sequentially, though Amdahls' Law severely limits the extension of this approach to larger number of processors.

If an algorithm can deal with the poor element quality characteristic of background-mesh methods [28, 29], it is possible to locally refine an octree version; in addition we might reap the advantages of multigrid methods.

References

1. N. P. Weatherill, *Mesh Generation in CFD*, in *Computational Fluid Dynamics*, von Karman Institute for Fluid Dynamics, Lecture Series 1989-04, March 1989.
2. M. S. Shephard, K. R. Grice, J. A. Lo and W. J. Schroder, *Trends in Automatic Three Dimensional Mesh Generation*, Comp. Struct., **30** (1988) 421.
3. T. J. Baker, *Developments and Trends in Three Dimensional Automatic Mesh Generation,* Appl. Num. Math., **5** (1989) 205.
4. R. Lohner, *Finite Elements in CFD: What Lies Ahead*, Int. J. Num. Meth. Engng., **24** (1987) 1741.
5. P. D. Thomas and J. F. Middlecoff, *Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations,* AIAA Journal **18** (1980) 652.
6. J. Hauser, S. Sengupta, P. R. Eiseman and J. F. Thompson, (Eds.), *Numerical Grid Generation in CFD*, Pineridge Press, Swansea, U.K., 1988.
7. J. Hauser, H. G. Paap, D. Eppel and S. Sengupta, *Boundary Conformed Coordinate Systems for Two-dimensional Fluid Flow Problems, I and II*, Int. J. Num. Meth. Fluids, **6** (1986) 507.
8. J. F. Dannenhoffer, III, *Adaptive Grid Computations for Complex Flows: A Supercomputing Challenge*, vol. II, p. 206 in Proc. 4th Int. Conf. on Supercomputing, eds L. P and S. I. Kartashev, Int. Supercomputing Inst., 1989.
9. W. J. Schroder and M. S. Shephard, *Geometry-Based Fully Automatic Mesh Generation and the Delaunay Triangulation*, Int. J. Num. Meth. Engng., **26** (1988) 2503.
10. P. M. Finnegan, A. Kela and J. E. Davis, *Geometry as a Basis for Finite Element Automation*, Engng. Comput., **5** (1989) 147.

11. A. Jameson, T. J. Baker and N. P. Weatherill, *Calculation of Inviscid Transonic Flow over a Complete Aircraft*, AIAA paper 86-0103.

12. A. Bowyer, *Computing Dirichlet Tesselations*, Comp. J., **24** (1981) 162.

13. F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

14. T. J. Baker, *Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation*, Engng. Comput., **5** (1989) 161.

15. D. J. Mavriplis, *Adaptive Mesh Generation for Viscous Flows Using Delaunay Triangulation*, J. Comput. Phys., **90** (1990) 271.

16. T. J. Baker, *Three Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets*, AIAA paper 87-1124-CP.

17. S. Rippa, *Minimal Roughness Property of the Delaunay Triangulation*, PhD thesis, Tel-Aviv University, 1990.

18. W. H. Frey, *Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes*, Int. J. Num. Meth. Engng. **24** (1987) 2183.

19. R. Lohner and P. Parikh, *Generation of Three Dimensional Meshes by the Advancing Front Method*, Int. J. Num. Meth. Fluids, **8** (1988) 1135.

20. J. Peraire, M. Vahdati, K. Morgan and O. C. Zienkiewicz, *Adaptive Remeshing for Compressible Flow Computations*, J. Comput. Phys, **72** (1987) 449.

21. M. M. Merriam, *A Fast Robust Algorithm for Delaunay Triangulation*, Internal Report, CFD branch, NASA Ames, 1990.

22. W. C. Thacker, A. Gonzales and G. E. Putland, *A Method for Automating the Construction of Irregular Computational Grids for Storm Surge Forecasts*, J. Comp. Phys., **37** (1980) 371.

23. W. J. Schroder and M. S. Shephard, *An O(N) Algorithm to Automatically Generate Geometric Triangulations Satisfying the Delaunay Circumsphere Criteria*, Engng. Comput., **5** (1989) 177.

24. E. K. Buratynski, *A Fully Automatic Three-Dimensional Mesh Generator for Complex Geometries*, Int. J. Num. Meth. Engng., **30** (1990) 931.

25. R. D. Williams, *Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations*, Concurrency, to be published.27M.-C. Rivara, *Design and Data Structure of Fully Adaptive, Multigrid, Finite-Element Software*, ACM Trans. Math. Soft., **10** (1984) 242.

26. R. D. Williams and E. W. Felten, *Distributed Processing of an Irregular Tetrahedral Mesh*, Caltech Report C3P793, 1989.

27. M.-C. Rivara, *Design and Data-Structure of Fully Adaptive, Multigrid, Finite-Element Software*, ACM Trans. Math. Soft., **10** (1984) 242.

28. M. Berger and R. J. LeVeque, *An Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries*, AIAA paper 89-1930.

29. R. J. Leveque, *High Resolution Finite Volume Methods on Arbitrary Grids via Wave Propagation*, J. Comput. Phys., **78** (1988) 36.