Generation of Multi-Million Element Meshes for Solid Model-Based Geometries: The Dicer Algorithm

Darryl J. Melander
Brigham Young University
Provo, UT 84602

Timothy J. Tautges
Sandia National Laboratories
PO Box 5800, Albuquerque, NM, 87112

Steven E. Benzley
Brigham Young University
Provo, UT 84602

ABSTRACT

The Dicer algorithm generates a fine mesh by refining each element in a coarse all-hexahedral mesh generated by any existing all-hexahedral mesh generation algorithm. The fine mesh is geometry-conforming. Using existing all-hexahedral meshing algorithms to define the initial coarse mesh simplifies the overall meshing process and allows dicing to take advantage of improvements in other meshing algorithms immediately.

The Dicer algorithm will be used to generate large meshes in support of the ASCI program. We also plan to use dicing as the basis for parallel mesh generation. Dicing strikes a careful balance between the interactive mesh generation and multi-million element mesh generation processes for complex 3D geometries, providing an efficient means for producing meshes of varying refinement once the coarse mesh is obtained.

INTRODUCTION

When considering the value of a finite element mesh generation algorithm, the primary criteria have always been algorithm robustness, the quality of the resulting mesh, the amount of required user interaction, memory usage, and algorithm speed. With the increasing need for meshes consisting of a large number of elements (one million or more), these criteria become more vital. A meshing algorithm or technique is needed that is able to generate large, high quality meshes for solid model-based geometries that is efficient in both memory and speed while minimizing user intervention.

The accuracy of a finite element analysis depends on the resolution of the finite element mesh that is used. The resolution of the mesh is limited by the time it takes for the mesh to be generated and by the available computer resources. The limiting factor is most often computer resources, especially the amount of available memory. Recent advances in computer technology have raised the ceiling on computing resources, making large meshes feasible and desirable. The teraflop computer, for example, contains 9000 parallel processors and 600 gigabytes of memory, giving it the capacity to run much larger problems than have been possible in the past [1]. In addition to the greater capacity of computing resources, the need for accurate modeling is also driving us towards finer finite element meshes. In order for complex systems such as the weapons systems of the ASCI program to be accurately modeled for reliability, performance, and safety, much higher resolution is needed in the finite element models for these systems. As the potential of computer systems approaches the requirements of vital analyses, algorithms must be developed to meet the demands of very fine meshes, capable of generating meshes as large as 100 million elements.

One approach that has been pursued in the CFD community is to use block-structured meshing. In this approach, the user defines a very coarse, all-hexahedral block structure; this structure is then refined to arrive at the final mesh. Examples of codes which use this technique are TRUEGRID [2] and ICEM-CFD [3]. The advantage of this technique is that once the initial block structure is defined, the generation of the final grid can be completely automated. Refinement can also be targeted at specific areas of the model with a careful choice of block structure. However, the primary disadvantage of this technique is that the block structure be defined manually. This task is one that is difficult to learn, and can be impractical for complex solid models. Since the primary difficulty in deriving all-hexahedral meshes for complex geometries is finding all-hexahedral

The portion of this work performed at Sandia National Laboratories was supported by the United States Department of Energy under Contract Number DE-AC04-94AL85000.

DISCLAIMER

connectivity [4], block-structured meshing techniques solve only part of the problem.

The Dicer algorithm attempts to resolve these issues by refining coarse all-hexahedral meshes generated by other algorithms. Each element of an existing coarse mesh is treated as a volume that will receive a fine mesh. Because each coarse element is hexahedral, assumptions can be made that allow it to be meshed quickly and consistently. By sharing meshing responsibilities with existing algorithms, speed and memory efficiency can be achieved without detrimental effects on mesh quality. Also, in contrast to the definition of blocks for block structured meshing applications, the definition of coarse hex elements can be somewhat automated, and can take advantage of new developments in all-hex mesh generation as they come along.

This paper is arranged as follows. Section 2 discusses overall strategies for generating large, all-hexahedral meshes. Section 3 describes the Dicing algorithm in detail. Section 4 discusses some overall issues in using the Dicing technique. Section 5 gives examples of where the dicing algorithm has been applied. Finally, Section 6 discusses future directions and Section 7 summarizes this paper.

STRATEGIES FOR GENERATING LARGE MESHES

Analysts seek meshing algorithms which produce high quality elements quickly while operating within the memory constraints of the computer. As one evaluates a mesh generation algorithm, five issues that must be considered are:

a) robustness (able to properly mesh a variety of different problems with complications)

b) element quality,

c) automation,

d) memory consumption, and

e) algorithm speed.

These challenges of mesh generation are usually amplified as the number of elements in a mesh increases or the available resources for generating a mesh decrease. Time-consuming calculations intended to ensure quality and promote robustness generally increase at a rate proportional to the number of elements in the resulting mesh. The amount of memory storage required also increases proportionally in most cases, including data structures attached to each element which are used by the algorithm but not vital to the final mesh (e.g. interior mesh faces and edges).

There are several possible approaches to generating large finite element meshes. At one end of the spectrum is the "all in one" approach, where the entire mesh is generated in a single execution of a meshing code. The opposite extreme, referred to as "all in many", generates a piece of the mesh at a time, most often for distinct regions or materials in the mesh, then assembles the entire mesh from the pieces in a separate step. While the all-in-one approach is very memory intensive, it does retain global information about the model. The all-in-many approach is not nearly as memory intensive, but also does not retain global information about the model. The dicing algorithm strikes a compromise between the two approaches by treating the global problem for the coarse mesh, while refining each coarse element individually.

THE DICER ALGORITHM

The Dicer algorithm is part of a two-step mesh generation process we envision to deal with the difficulty of generating large meshes for complex geometries. First, a relatively coarse all-hexahedral mesh is generated by the user, in a manner identical to mesh generation methods currently used. Then, this "coarse" mesh is refined to a user-defined level of refinement, using a simple Trans-Finite Interpolation (TFI) algorithm on each of the coarse hexahedra. Each of these steps is described in greater detail below.

1. A Coarse All-Hexahedral Mesh is Generated

As the first step to generating a fine finite element mesh, a coarse mesh is generated using one or more existing all-hexahedral mesh generation algorithms. The analyst should use algorithms that result in the best possible quality at a coarse level of refinement for the geometry in question. Because this coarse mesh will be used as the basis for the fine mesh, the coarse mesh should have element sizes and shapes that will be appropriate for refinement. On the other hand, if CUBIT [5] is used for the coarse mesh, the coarse elements must be fine enough to resolve all geometric features of the model, due to the requirement that all geometric features be meshed with at least one element. Note that the coarse mesh could also be assembled from meshes generated in other runs of CUBIT or even meshes generated by other codes (assuming the mesh from these other codes can be associated to solid model geometry before refinement).

2. Determine Dicer Loops & Sheets

In order to ensure a conformal mesh, three principles are enforced:

a) Coarse element faces that share an edge must also share the fine mesh along that edge;

b) Coarse hexahedral elements that share a face must also share the fine mesh on that face;

c) Coarse edges opposite each other on a given coarse face must receive the same degree of refinement in the fine mesh.

Principle a) and principle b) are necessary to ensure a conformal fine mesh; these constraints are enforced by storing only one set of nodes for each coarse edge or face. Principle c) above ensures that the coarse hexes and faces will be meshable with a simple TFI algorithm. These principles together result in a certain amount of pre-processing to ensure that they are followed.

To find the collection of coarse edges which must have the same refinement interval, a series of "dicer sheets" is constructed, using the following algorithm. First, all edges are unmarked. An unmarked coarse edge is added to a new dicer sheet, and to a list of untraversed edges for that sheet (this list is initially empty except for that first edge). Then, while the untraversed list is not empty, a new untraversed edge is taken off

the list; for each face connected to this edge, the opposite edge from the edge being traversed is found and, if not already marked, is added to the untraversed edge list. Lastly, the edge being traversed is added to the dicer sheet edge list. When the untraversed list is empty, the entire dicer sheet has been traversed, and the edges on that sheet must have the same refinement. When all edges in the coarse mesh have been marked, all the dicer sheets have been found. For example, Figure 1 shows edges that belong to the same dicer sheet.

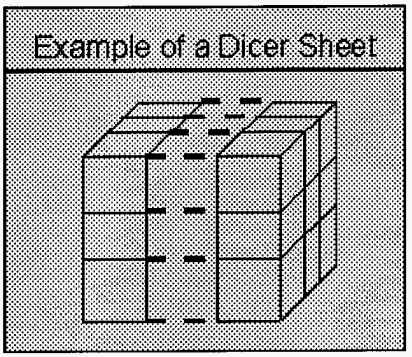

Figure 1. Edges with a dashed line must have the same refinement interval.

3. Refinement Intervals Are Specified

After sheet traversal has determined which edges correspond to each dicer sheet and must have the same refinement interval, the user must specify the refinement intervals. This can be done for each dicer sheet individually, for each geometric entity (e.g. volume), or for the entire model. Assigning a refinement interval to a volume, for example, would set the refinement interval for every sheet that has at least one coarse edge within the volume. Allowing these methods of interval specification gives the user fine control over the degree of refinement without requiring tedious work to assign the same interval to a region of constant refinement.

4. Coarse Elements Are Meshed With a Simple TFI Algorithm

After the refinement interval has been specified for all sheets, each coarse element may be meshed as follows. The coarse edges are meshed first, using equal interval refinement. The coarse faces are meshed by gathering the fine nodes, arranging them on the boundary of a two-dimensional array, and filling the interior using a simple surface TFI algorithm [6]. The coarse hexes are meshed in a similar manner: the fine mesh on all six faces is gathered and arranged on the boundary of a three-dimensional array, and the interior is filled using a volume TFI algorithm.

The method used to mesh each coarse edge and face is deterministic, and this can be used to optimize the refinement process. First, since the refinement intervals have been set for dicer sheets as a whole, there is no need to check that they are equal for each pair of opposite edges on a face, and opposite faces on a hex. Also, the structured topology of the fine mesh on the coarse mesh faces can be retained; this makes the construction of the boundary of the three-dimensional node array simpler, requiring simply a comparison of coarse nodes at a corner and orientation of the coarse nodes for each face. Note that the orientation of a fine mesh on a coarse face will have to be reversed when meshing one of the bounding hexes.

By constructing the bounding node arrays as outlined above, we are able to use the standard map mesh functions inside CUBIT for computing the fine mesh.

Geometric boundaries are the only regions that need special consideration. Because element faces and edges are only approximations to the underlying geometry, refining these entities requires that any fine nodes generated in the refinement be moved to the geometry boundary. Nodes along a coarse mesh edge "owned" by a bounding geometric curve should be moved to that curve before the attached coarse face is refined, and nodes on a coarse mesh face owned by a geometric surface should be moved to that surface before the associated hexahedral element is refined. This will encourage a higher quality mesh on the model as a whole. Figure 2 shows coarse elements and the associated

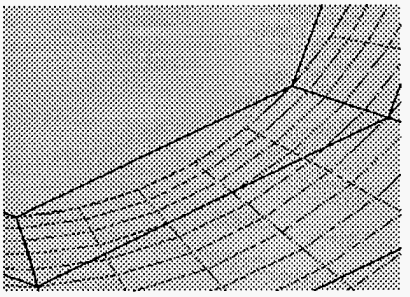

Figure 2. Elements refined along a curved surface follow the original curve, not the coarse element edge.

fine elements along a curved surface that has been refined using the Dicer algorithm.

DISCUSSION

In the generation of large finite element meshes, the Dicer algorithm offers several benefits over simply generating a fine mesh in one step with an existing algorithm. The advantages relate to robustness, element quality, automation, memory usage, and algorithm speed.

Dicing holds a robustness advantage over using advanced 3D unstructured meshing algorithms, in particular whisker weaving [7], for generating the fine mesh. This is due to the fact that whisker weaving performs poorly when generating relatively fine mesh.

Widespread use of the TFI algorithm can minimize the overall number of irregular nodes in the mesh, since TFI produces a structured mesh. However, using TFI to mesh complex geometries can result in poorly shaped elements. By using a combination of other algorithms and TFI, the user can strike a balance between element shape quality and number of irregular nodes.

As stated earlier, the dicing algorithm is analogous to block structured meshing, differing only in its use of 3D meshing

algorithms to define the block structure. This difference is important for several reasons. First, the generation of blocks can in some cases be automated, or at least can be simplified; this dramatically reduces the user interaction required to define the block structure. Secondly, the number of blocks is not determined by the tenacity of the user, but rather by available memory space and required geometric refinement. Finally, using a coarse mesh-based approach allows the user to take advantage of new meshing algorithms that are developed in parallel to this effort, for example the whisker weaving algorithm [7]. The only requirement on generating the coarse mesh is that it yield a mesh that is associated with the geometry, so that refined mesh can be moved to that geometry.

As discussed in the preceding section, storing the mesh in array-based structures allows us to retain array information useful in constructing the bounding node arrays for meshing the next higher order coarse element. Using this storage scheme reduces execution time, but also can lead to reduced memory usage. For example, if the bounding fine mesh is known for a given coarse hex, the interior mesh entities may possibly be inferred without needing to actually generate and store them. This could lead to dramatically reduced storage requirements of very large meshes. Note that this assumes that a deterministic algorithm is used to generate the fine mesh.

If the entire fine mesh were generated at once, and assuming the geometry was non-trivial, these elements would need to be generated with algorithms more complex than TFI. Since these algorithms are typically more computationally expensive than TFI, the mesh generation speed would be decreased. Using dicing, we generate the majority of elements using the most efficient algorithm available to us. Also, the coarse mesh effectively divides a solid model into regions that may be processed independently, after the interface mesh has been computed. Structuring the dicing algorithm to refine the coarse edges, faces and hexes in that order promotes the use of parallel processing to speed that task. We have plans to pursue parallel mesh generation using the dicing algorithm.

EXAMPLES

The dicing algorithm has been implemented inside the CUBIT mesh generation toolkit [5]. Both surface and volume dicing are available; however, due to database issues, this code does not yet have the capability to save the fine mesh to a disk file.

An example of surface dicing is shown in Figure 3. Here, the geometry is first meshed coarsely using the paving algorithm, then diced with a refinement interval of 10 everywhere.

Figure 4 shows a diced mesh where a non-constant refinement has been used. Note how the dicer loop refined to 5 intervals propagates around the surface.

In Figure 5, a simple volume has been meshed with the multi-sweep algorithm[5], then diced with a constant refinement interval of 5.

FUTURE DIRECTIONS

Future plans for the dicing algorithm include integrating it with higher level mesh storage techniques and enabling the dicing of non-hex elements.

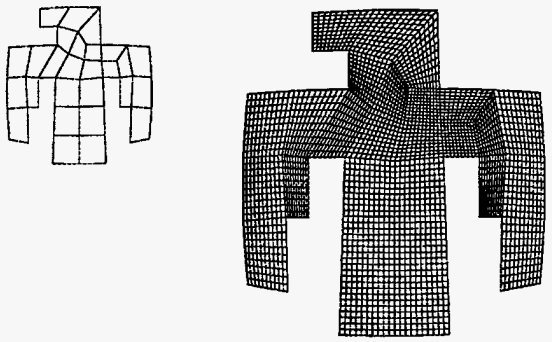

Figure 3. Thunderbird with coarse mesh (left) and constant dicing refinement of 10 (right).

Figure 4. Surfaces meshed with non-constant refinement (left - coarse mesh; right - partially refined mesh).

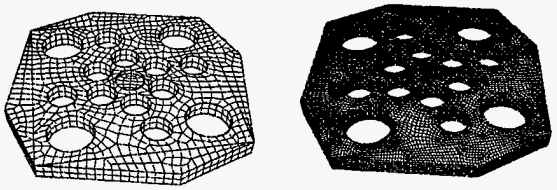

Figure 5. Volume dicing on a simple geometry; coarse mesh (left), fine mesh (right); only the surface mesh is shown for clarity.

The primary purpose of developing the dicing algorithm within CUBIT is to enable the construction of very large meshes, i.e. meshes of 10-100 million elements. Therefore, it will be of paramount importance to reduce the storage requirements of the fine mesh in CUBIT. We envision several methods for doing this. First, only critical data for this mesh will be kept, rather than the full data usually kept for mesh in CUBIT. Second, we plan to explore higher level methods for storing mesh data, for example storing the data as a bounding mesh plus an algorithm. Finally, we also expect to require some out of core storage of mesh data,

done in such a way as to minimize repeated reads and writes from/to the disk. All these methods will be encapsulated in a C++ class framework so that details of the storage technique are hidden from the code using the mesh.

Currently, most of the hex meshing algorithms in CUBIT produce all hexahedral meshes. However, there are several algorithms under development which produce not only hexes but also limited numbers of tetrahedra and knife elements [8]. Since knife elements are bounded completely by quadrilaterals, the propagation of dicing into and out of these elements should be possible. Dicing knife elements will produce a mixture of fine hex elements and finer knife elements. Dicing meshes containing tetrahedra will not be as straightforward. Since tetrahedra will be used to close small voids in the mesh not handled by an all-hex meshing algorithm, they should appear in small pockets rather than interspersed through the mesh. Therefore, the approach envisioned now would be to simply re-tetrahedralize these voids, starting with the fine mesh boundary instead of the coarse mesh boundary.

SUMMARY

The Dicer algorithm is a finite element mesh refinement algorithm that allows meshes consisting of large numbers of elements to be generated quickly while minimizing memory storage requirements. The Dicer algorithm generates a fine mesh by refining each element in a coarse mesh generated by any existing all-hexahedral mesh generation algorithm. Each coarse element is refined by treating it as a semi-independent hexahedral element. Mesh conformity is achieved by propagating refinement intervals across opposite edges and faces in the coarse all-hex mesh. Fine elements are generated quickly because the simple geometry of a hexahedral element allows it to be mapped with only minimal calculations. The hexahedral geometry of coarse elements also allows the fine mesh to be stored efficiently in arrays.

The Dicer algorithm will be used to generate large meshes in support of the ASCI program. We also plan to use dicing as the basis for parallel mesh generation. Dicing strikes a careful balance between the interactive mesh generation and multi-million element mesh generation processes for complex 3D geometries, providing an efficient means for producing meshes of varying refinement once the coarse mesh is obtained. Using multiple levels of refinement in the same problem may enable the use of multi-level solution techniques on unstructured hexahedral meshes as well.

REFERENCES

1. ""Intel Captures $46m Teraflop Supercomputer Development", press release, Intel Corp, Sept. 7, 1995. See http://www.ssd.intel.com/success/tflop1.html.

2. "TrueGrid: Hex Meshing for Structures and Fluids"; see http://www.truegrid.com.

3. "ICEM CFD Hexa"; see http://icemcfd.com/hexa.html.

4. Timothy J. Tautges, Scott A. Mitchell, "Progress Report on the Whisker Weaving All-Hexahedral Meshing Algorithm", in Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Field Simulations, p. 659-670, Mississippi State University, MS, April 1-5, 1996.

5. Ted D. Blacker et al., "CUBIT Mesh Generation Environment, Volume 1: User's Manual", SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, May 1994.

6. W. A. Cook, W. R. Oakes, "Mapping Methods for Generating Three-Dimensional Meshes", *Computers in Mechanical Engineering*, Aug. 1982, pp 67-72.

7. Timothy J. Tautges, Ted D. Blacker, Scott Mitchell, "The Whisker Weaving Algorithm: a Connectivity-Based Method for Constructing All-Hexahedral Finite Element Meshes", *Int. j. numer. methods eng.*, 39, 3327-3349 (1996).

8. Timothy J. Tautges, Scott Mitchell, "Whisker Weaving: Invalid Connectivity Resolution and Primal Construction Algorithm", Proceedings of the 4th International Meshing Roundtable, Sandia National Laboratories report SAND95-2130, Oct. 16-17, 1995, Albuquerque, NM.

Report Number (14) SAND--97-2125C
CONF-9706106

Publ. Date (11) 199709
Sponsor Code (18) DOE/DP, XF
UC Category (19) UC-700, DOE/ER

DOE