INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng. **44**, 1359–1376 (1999)

SURFACE TRIANGULATION OVER INTERSECTING GEOMETRIES

ALEXANDER A. SHOSTKO[1], RAINALD LÖHNER[2,*] AND WILLIAM C. SANDBERG[3]

[1]*Berkeley Research Associates Inc., Springfield, VA, U.S.A.*
[2]*CSI, George Mason University, Fairfax, VA, U.S.A.*
[3]*Naval Research Laboratory, Washington, DC, U.S.A.*

SUMMARY

A method for the rapid construction of meshes over intersecting triangulated shapes is described. The method is based on an algorithm that automatically generates a surface mesh from intersecting triangulated surfaces by means of Boolean intersection/union operations. After the intersection of individual components is obtained, the exposed surface parts are extracted. The algorithm is intended for rapid interactive construction of non-trivial surfaces in engineering design, manufacturing, visualization and molecular modelling applications. Techniques to make the method fast and general are described. The proposed algorithm is demonstrated on a number of examples, including intersections of multiple spheres, planes and general engineering shapes, as well as generation of surface and volume meshes around clusters of intersecting components followed by the computation of flow field parameters. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: unstructured grids; intersecting surfaces; mesh generation; advancing front

1. INTRODUCTION

The construction and adaptation of two- and three-dimensional surface and volume meshes has been widely studied in the engineering community and a number of algorithms have appeared in recent years to generate unstructured grids around complex geometrical shapes. Among the most popular algorithms we mention the Advancing Front,[1–3] Voronoi/Delaunay algorithms,[4,5] finite/modified quadtree/octree[6] and Paving and Plastering techniques.[7]

Before domain meshes can be generated, a surface representation for such domains needs to be constructed. Most grid generators start from analytical functions that describe the surface.

These analytical functions are obtained either by extracting the surface after performing a Boolean combination of solid primitives (e.g. boxes, spheres, cones, etc.), or they are specified in terms of independent surface patches (e.g. Coon's patches). In order to generate surface grids, these surface segments need to be mapped to two dimensions, using coordinate transformations. A considerable amount of shapes encountered in engineering cannot be represented in a

*Correspondence to: Rainald Löhner, GMU/CSI, MS 53C, Department of Civil Engineering, George Mason University, Fairfax, VA 22030-4444, U.S.A. E-mail: rlohner@science.gmu.edu

Contract/grant sponsor: Office of Naval Research

CCC 0029–5981/99/091359–18$17.50

Received 14 July 1997
Revised 30 June 1998

straightforward way by analytical functions. Examples of this kind are all data sets that originate from measurements (e.g. geological data, satellite data), from common virtual reality triangulations, large-deformation fluid-structure problems, and when hundreds of complex bodies are considered for a simulation. In these cases, the shapes can be more suitably viewed on a component-based basis, rather than as a single complete configuration. In order to model geometries represented by arbitrary overlapping components, a robust algorithm for automatic grid construction over intersecting shapes is mandatory. While considerable efforts have been devoted to the problem of joining smooth parametric surfaces,[8] far less study has been reported on performing Boolean operations on discretely specified shapes. Unstructured grid generation over intersecting geometries was pioneered by Lo.[9] An algorithm for Cartesian mesh generation over component-based geometry was reported recently by Aftosmis *et al.*[10] In the field of computer graphics, the issues of combined surface and volume representation, surface reconstruction from a set of points, and re-tiling of polygonal surfaces were studied in References 11–14.

In the current work a new general and robust method for mesh generation over intersecting geometries is described. It is based on the algorithm that constructs surface meshes from given intersecting triangulated surfaces by identifying intersecting surface parts, re-triangulating them along the intersection lines, followed by a removal of internal enclosed triangles. As a final step, the surface quality is improved by identifying and removing badly shaped elements. Efficient data structures eliminate most of the space search overhead, resulting in an algorithm that for most cases scales as the number of intersecting surface parts, not the whole input mesh size.

The remainder of the paper is organized as follows: the details of the algorithm are described in Section 2, followed by applications and a discussion of performance in Sections 3 and 4. Some conclusions are drawn in Section 5, together with an outlook for further research.

2. DESCRIPTION OF THE ALGORITHM

The input to the algorithm consists of two triangulations defined by a set of coordinates and a connectivity matrix to define the triangles.

2.1. Intersection algorithm options

Originally, the intersections were computed using the following algorithm (similar to the method discussed in Reference 9):

(a) Find intersections between input surfaces; this generates intersection points and edges.
(b) Remove the intersecting triangles, identifying the resulting disjoint surface segments.
(c) Select the surface segments that are needed.
(d) Identify the edges of the selected surfaces; this set of edges constitutes the original front.
(e) Remesh gaps/voids with new faces (triangles) using advancing front technique.
(f) Determine and retain internal or external parts of the surface if so required.

These steps may be seen in the examples shown in Figure 4, where the original intersecting surfaces, generated intersection lines and points, intersecting triangles and the final joined surfaces are illustrated for a plane–sphere intersection. While a number of useful shapes were generated by the application of this algorithm, robustness problems were encountered for some complex geometric cases. The generation of the surface mesh directly in 3-D requires a consistent

direction, which is contained in the shape of intersecting surfaces. The algorithm worked well to compute intersections typically used for engineering applications, such as planes, spheres and cylinders. The algorithm was then applied to generate meshes around large protein molecules (from the Brookheaven protein data bank).[15] The surfaces thus created exhibited kinks and sharp geometry changes (see Figure 10). In some cases the advancing front was losing a consistent direction and was unable to fill in the gaps. The direction of the front edges often was pointing inside or outside the domain, where no points or wrong points can be selected. Abrupt changes of the front direction, resulting from the intersection of surfaces with sharp topology variations, were common. In order to maintain the direction consistency, a number of heuristics were introduced. While it significantly improved the method performance, it did not result in a general algorithm.

Therefore, an alternative method was developed. The major steps of the modified algorithm are:

1. Find intersections between input surfaces;
2. Generate intersection points and lines;
3. Re-triangulate each intersecting triangle with new elements using the advancing front technique;
4. Identify resulting surface segments and select those that are needed;
5. Improve surface mesh quality.

Note that the advancing front algorithm is applied within each intersecting element, which makes it much simpler and robust. The algorithm does not delete any elements and therefore is independent of the surface topology and the granularity of the mesh discretization. The disadvantages of this option relate mainly to additional development efforts. This algorithm requires special efforts to handle singularities. The iterative application of the algorithm to construct surfaces from many intersecting objects (see Figure 10) can lead to badly shaped elements, demanding some kind of mesh improvement effort. However, the advantages of this option far outweigh its possible disadvantages and this alternative technique is described in detail below. The steps of the algorithm are demonstrated in Figure 5 for the intersection of a sphere and a plane, where the original intersecting surfaces, generated intersection lines and points, intersecting triangles and the final joined surfaces are illustrated. The intersections of a number of engineering shapes are illustrated in Figures 6–9. The steps of this algorithm are described below.

2.2. Surface intersection

The intersection between surfaces is determined by considering each triangular element in turn, analysing its neighbourhood for triangles from the other surface, and calculating the intersection points.

2.2.1. Algorithm

(i) Filter out all elements of the surfaces that are too far apart and cannot intersect. This filtering is done using a Cartesian bounding box technique.
(ii) For the remaining points and triangles: build the data structures required for computational efficiency (spatial trees, lookup tables, etc.). Only the points of the second surface are stored in a tree.
(iii) For every triangular face **iface** of the first surface:

(a) Find all faces of the second surface close to this face; this is done by first finding all points close to face **iface**, and then finding all faces adjacent to these close points.
(b) For each close face **jface** found:

Determine whether faces **iface, jface** intersect; if they do, generate and save the intersection points and intersection edge. The intersection is determined by testing all edges of one face against the plane segment enclosed by another face.

2.2.2. Computing intersection edges and intersection points. The intersection is computed by testing whether an edge of the triangle intersects the plane of the other triangle. In general, in order for two triangles to intersect, the following conditions must be met:

1. The edge must cross the plane of the triangle.
2. The intersection point must lie within a plane segment enclosed by the triangle.

Originally, the intersection was constructed by computing directly the pierce point of the edge and triangle. This scheme required special efforts to take care of zeroes in divisions and the introduction of tolerances to avoid comparing the scalar and vector products with zeroes.[9] Although the scheme worked satisfactorily for a number of cases, it was not found sufficiently accurate and robust for general intersections. In an alternative method, the test whether edge and triangle can intersect was implemented using signed volumes of tetrahedra (similar to the framework reported in Reference 10 by Aftosmis *et al.*) and the intersection point was generated by solving the parametric equation of the edge–plane intersection.

The following algorithm was used, given the edge and the face to determine the intersection:

(a) Check for singularity cases and generate intersection point. The intersection points were generated according to the singularity cases (line touching the plane or line in plane of the triangle). This condition holds if (see Figure 1)

$$V_{\mathrm{deab}} = 0 \cdot 0 \tag{1}$$

where V_{deab} is a volume of a tetrahedron formed by base DEA and point B.
(b) If the above condition is true, determine whether points are on the opposite side of the face:

$$V_{\mathrm{abcd}} \cdot V_{\mathrm{abce}} > 0 \cdot 0 \tag{2}$$

where V_{abcd} is a volume of tetrahedron formed by face ABC and point D, and V_{abce} is a volume of a tetrahedron formed by base ABC and point E,

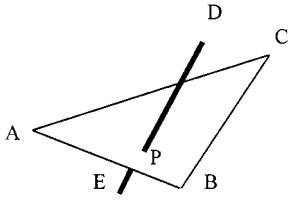

Figure 1. Generation of intersection points

(c) If the previous test is passed, determine whether the intersection point is within the boundaries of the triangle ABC. For this to be true, all three tetrahedra formed by line ED and edges of the triangle ABC must have the same sign, i.e.

$$V_{\text{eabd}} > 0{\cdot}0 \wedge V_{\text{ebcd}} > 0{\cdot}0 \wedge V_{\text{ecad}} > 0{\cdot}0 \vee V_{\text{eabd}} < 0{\cdot}0 \wedge V_{\text{ebcd}} < 0{\cdot}0 \wedge V_{\text{ecad}} < 0{\cdot}0 \quad (3)$$

(d) If all above tests are passed, compute the intersection point P (see Figure 1) by solving the parametric equation for plane–line intersection. In order to avoid computing the same intersection again for adjacent faces and edges, the generated intersection points are stored in a hash table.

It was found that this way to compute triangle intersections is void of the problems encountered by the direct computing scheme, with no additional computational overhead. The intersection points and edges are illustrated in Figures 5–9 for a variety of intersecting shapes.

2.3. Advancing front mesh generation algorithm

After computing the intersections, relevant line segments are identified and stored for each intersected triangle. These segments divide each face into several regions, which are either inside or outside of the body. In order to produce a consistent mesh, each face needs to be re-triangulated. This new triangulation has to be constrained along the intersection segments to enable removal of those triangles which are inside. A variety of approaches exist for constructing a triangulation in 2D or 3D. In the current work we selected the advancing front method due to its generality and universality. A brief description of the advancing front algorithm is provided below for completeness. For a detailed description of the advancing front technique we refer the reader to References 1 and 3.

In the advancing front family of algorithms the points and elements are introduced into yet empty space until the complete computational domain is filled with elements. The 'front' denotes the boundary between the region of space that has been filled with elements and that which is empty. The advancing front algorithm is general for complex geometries, and offers easy incorporation of grading and stretching. The disadvantages of the advancing front relate mainly to its efficiency. However, considerable gains in performance have been reported by using hierarchical search, parallelization, vectorization of intersection calculations, as well as other techniques.[1–3] The advancing front algorithm requires an initial front to start the generation of elements in the domain. It is illustrated for a simple example in Figures 2(a) and 2(b), where the front, close edges and a new introduced point are displayed.

The following algorithm was applied for each face to re-triangulate the faces. While there are segments to fill:

1. Identify the next closed loop and construct a front direction for each edge;
2. Select the next available active edge to be deleted from the front;
3. For the edge to be deleted:

   (a) Find all points and front-edges close to selected edge;
   (b) Select the best point **pnew** for the introduction of a new element. The point is selected based on Voronoi criteria;[4, 5, 9]
   (c) Determine whether the element formed with the selected point **pnew** crosses any relevant close faces. If it does, take another point.

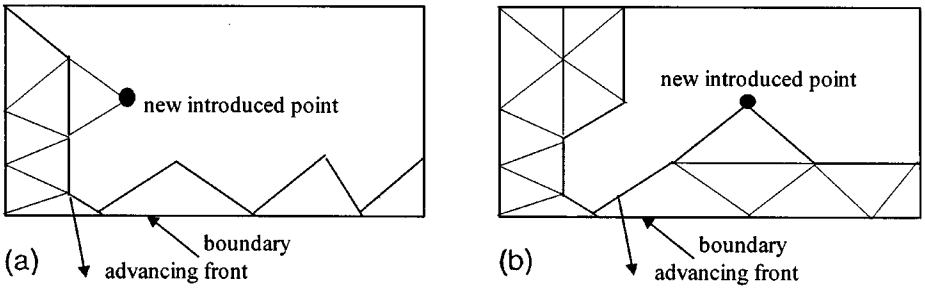

Figure 2. (a) Advancing front on simple example; (b) progressed advancing front

4. Add new element, point, edges to respective lists;
5. Find the generation parameters for a new edge;
6. Delete known edges and inactive points from the front.

In the current work an advancing front algorithm for direct surface triangulation in 3-D was used. However a 2-D version of the advancing front is sufficient to accomplish the task. The initial advancing front edges identified after the intersection step are illustrated in Figure 4(e).

2.4. Determination of internal/external surfaces

Several techniques were reported in the literature for the determination of internal and exposed (external) parts of intersecting components, including ray-casting or painting algorithms.[10] In the current work, the following algorithm was used for inside/outside determination of two intersecting components:

1. Mark all triangles as untouched, build lookup tables of points surrounding points and faces surrounding faces;
2. Compute the min/max of all surfaces and initialize the selection list; the first triangle which contains min/max dimensions is selected;
3. While there are triangles in the selection list:

   (a) Obtain the next untouched triangle **iface** from the selection list and mark it as touched;
   (b) For each side of the triangle **iface**:

      (i) If there exists a side-adjacent neighbor **jface** that is not marked yet: append it to the selection list;
      (ii) If there exist several neighbors on this side:

         (1) Select the triangle **jface** visible from **iface**
         (2) Append it to the selection list if it is not marked before;
         (3) Mark the selected triangle **jface**;
         (4) If no triangles are left in the selection list: break from the loop.

The selection of a visible neighbour is demonstrated in Figure 3.

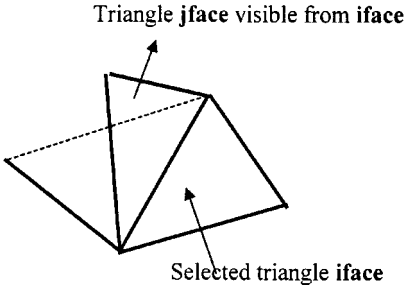

Figure 3. Automatic removal of occluded triangles

2.5. *Surface improvement*

The surface generated by the intersection algorithm was used to construct volume meshes using a general purpose 3D grid generator.[3] It was found that the repetitive application of the intersection algorithm (e.g. for many-body problems) leads to the appearance of highly distorted elements. Some of the aspect ratios encountered exceeded 1:100 000. Apart from a decrease in computational efficiency, the appearance of highly distorted elements introduces significant difficulties in the subsequent volume mesh generation process. This makes a surface improvement a mandatory step after the intersection computation. The surface improvement is performed in several passes after the intersection stage, using the following algorithm:

1. Mark all highly distorted faces;
2. For all faces marked:
   (a) Select the smallest side of the face;
   (b) See if collapsing this side leads to a reversal of normals of any of the neighboring faces; if it does, skip this face and continue to the next face;
   (c) See if collapsing this side leads to quality improvement in the neighboring faces; if there is no improvement, continue to the next face;
   (d) Collapse this face and update surface topology.

2.6. *Data structures for efficiency*

There are a number of operations in the algorithms above that potentially lead to $O(N^2)$ or $O(N^{1.5})$ complexity. Among those operations are:[1,16]

(a) Finding the element close to a given element to compute intersection;
(b) Finding whether the intersection point has been already computed (to avoid duplication of efforts);
(c) Finding the next best edge to be taken from the front;
(d) Finding the closest given points to a new point;
(e) Finding the faces adjacent to a given point;

The data structures used to minimize search overhead include:

1. Bounding boxes to filter elements too far off;
2. Spatial trees to store points of the surfaces;

3. Lookup tables of elements surrounding each point and element surrounding each element;
4. Priority queues for advancing front;
5. Sorting routines to find the best point from the list of close points;
6. Hash tables to avoid re-computing the same intersection points for neighbouring triangles.

Generic libraries were used for most data structures mentioned. Spatial trees were implemented as N-ry[17] trees operating on octant class, which contains min/max of the space region assigned to it and the list of points. The spatial encoding of the surface information is paramount for algorithm robustness and efficiency. Both adaptive binary trees and octrees were tried in the current work with a varying number of points stored per octant. While octrees in most cases lead to a better performance, an adaptive binary tree was found more accurate for complex geometric shapes. However, an optimal selection is application specific. The application of spatial trees drastically reduced the computational time required. However, for some cases (compact geometries with uniform fine grid) efficiency was observed to decrease. The data structures mentioned reduce algorithm complexity to $O(N \log N)$[16] and allowed to use the method at interactive rates for large cases on a single processor SGI Indigo workstation (see Section 4 for performance results).

3. APPLICATIONS

The described algorithm was used for a number of problems. Due to lack of space, only a small sampling of these is included here.

3.1. Intersection of a sphere and a plane

The method is demonstrated in Figures 4 and 5 for the repeated intersection of planes and spheres, where input surfaces, computed intersection, marked intersecting elements, resulting internal/external surface segments and final joined surfaces are shown for both intersection algorithm alternatives discussed above.

3.2. Engineering shapes

The intersection of two cylinders are shown in Figure 6. Each input mesh consists of 8200 triangles and the resulting joined mesh has $> 14\,000$ elements. The input meshes are shown in Figure 6(a), the computed intersection line is shown in Figure 6(b) and (c), identified external/internal surface segments are displayed in Figure 6(d). The final joined surface is illustrated in Figures 5(e) and (f). The intersection of the caudal fin and the tail of robotuna is shown in Figure 7. The intersection of two planes and three spheres is demonstrated in Figure 8(a)–(c). The intersection of a channel and a sphere is displayed in Figure 9. The code is now being routinely used by engineers for complex cases to compute the intersection line for subsequent surface definition in general purpose CAD modules.

3.3. Molecules

Current interest in predicting accurately physical phenomena at the scale of individual molecules has prompted to use molecular clusters from Brookheaven Protein Data Bank as

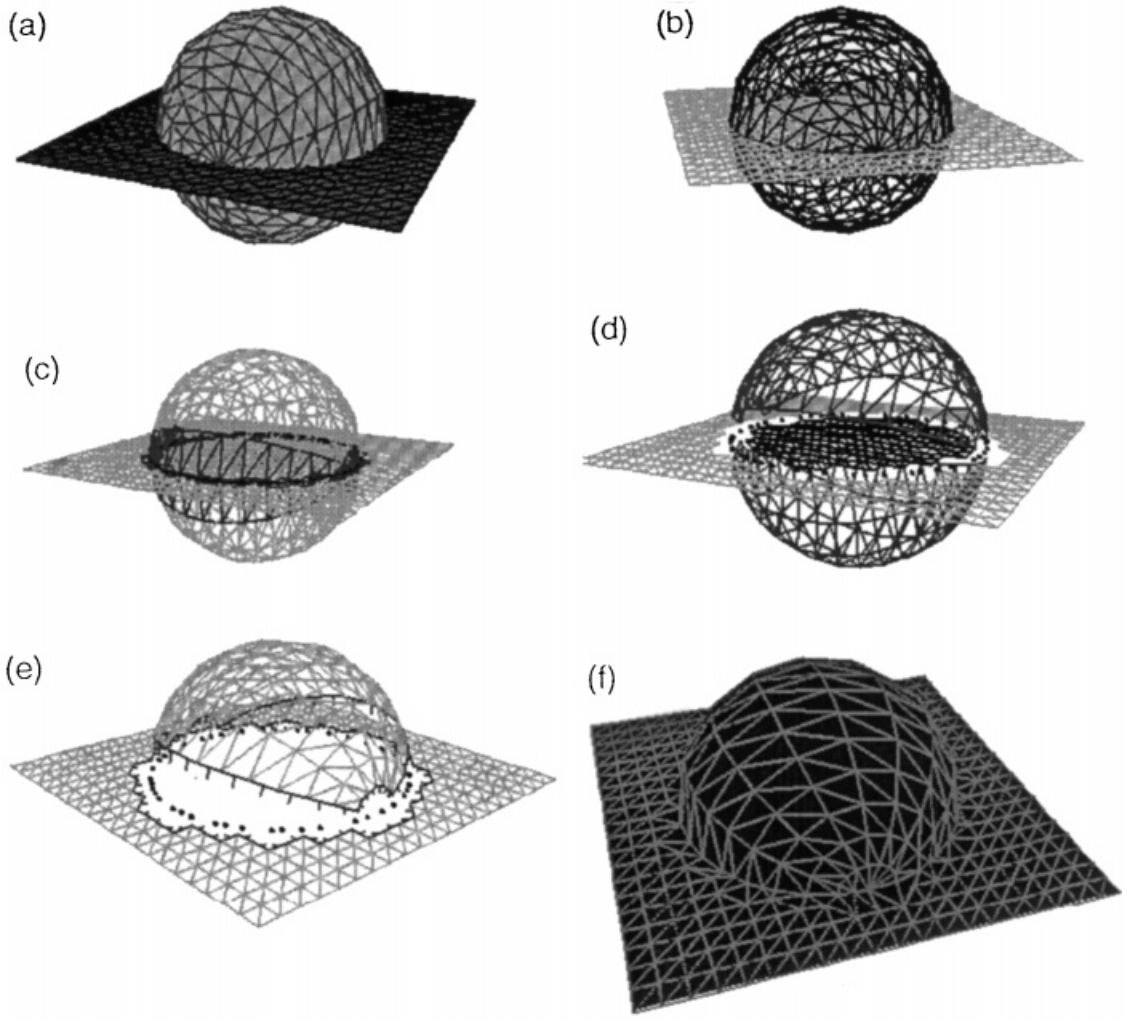

Figure 4. Intersection example. Global advancing front: (a) surface to intersect; (b) computed intersection line; (c) intersecting elements marked; (d) surface resulted from intersection; (e) selected surfaces to join; (f) joined surfaces

samples of overlapping components. The models of 3-D molecular surface are useful for computing the electrostatic field around the molecules in a solvent,[18–21] visualization, calculation of the accessible area and excluded volume of polymers, drug design, study of protein–substrate docking, protein-folding phenomena, polymer design and molecular dynamics simulations. There are several definitions of molecular surfaces. For detailed information on the subject, the reader is advised to consult the appropriate References 19–27. The generated molecular surface around a molecular cluster (2000 molecules: 100 854 elements and 50 410 points) from the Human Hemoglobin protein is shown in Figure 10.

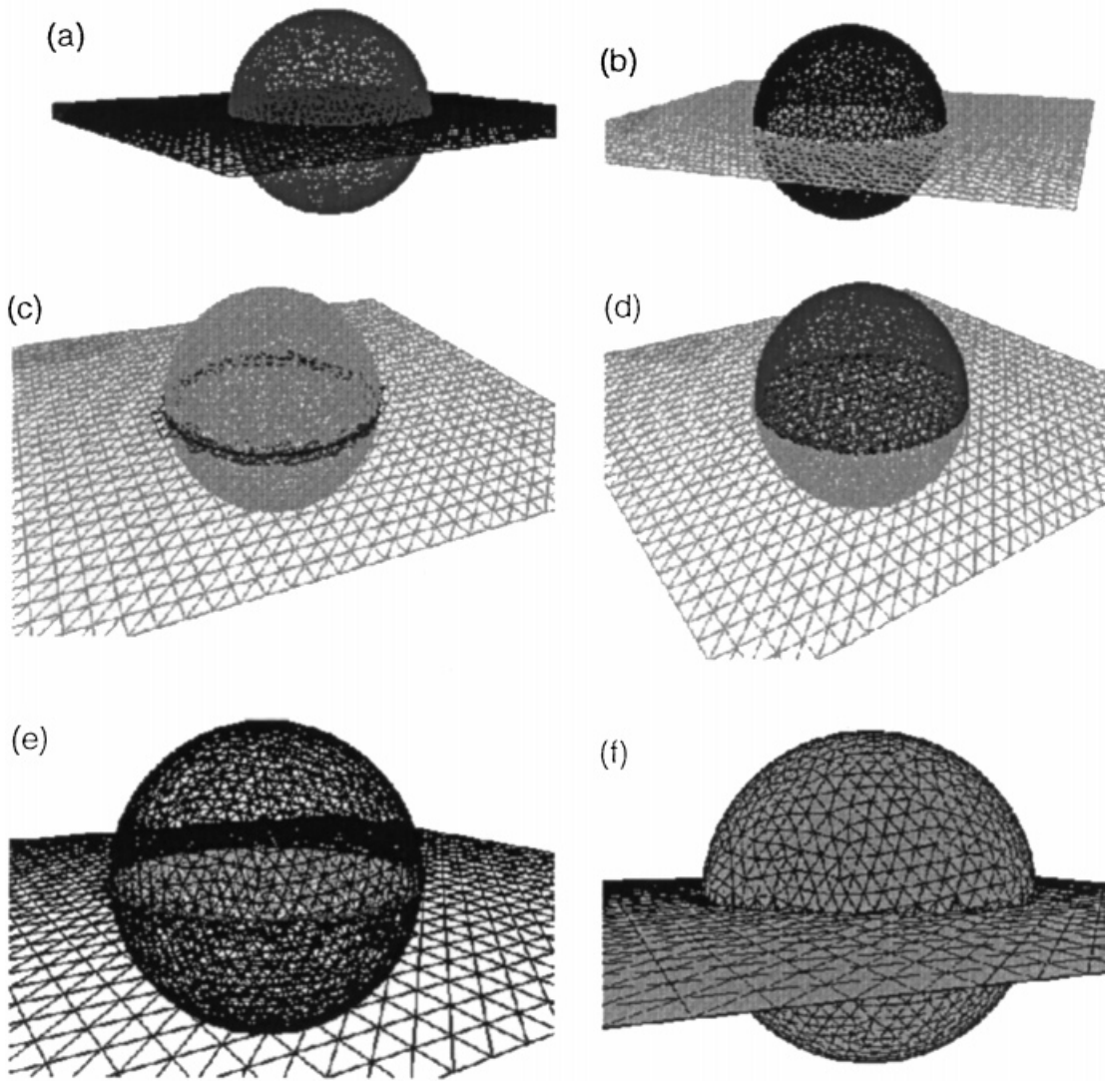

Figure 5. Intersection example. Local advancing front: (a) surfaces to intersect; (b) computed intersection line; (c) intersecting elements marked; (d) surfaces resulted from the intersection; (e) internal/external identified; (f) final joined surfaces

3.4. Flow computations past intersecting components

The surface mesh generated around intersecting components was used to construct the volume mesh and to compute external flow field parameters. This application is demonstrated in the example of 20 intersecting spheres representing molecular cluster in Figures 11–13. The generation of the volume mesh and flow field computation was done using the general-purpose code.

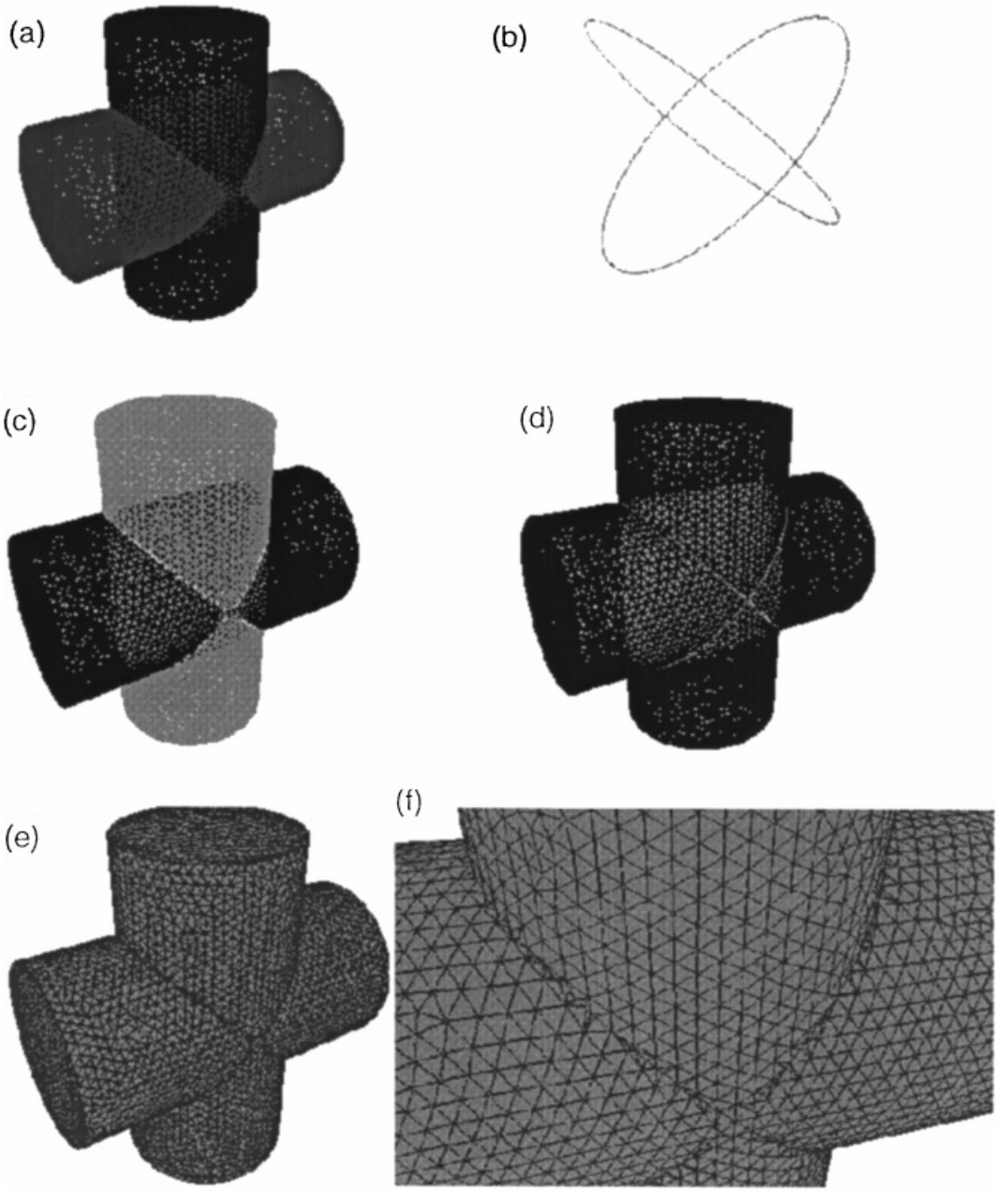

Figure 6. Intersection of two cylinders: (a) input surfaces for intersection; (b) computed intersection line; (c) intersection line; (d) external/internal identified; (e) final surface; (f) final surface, enlarged view

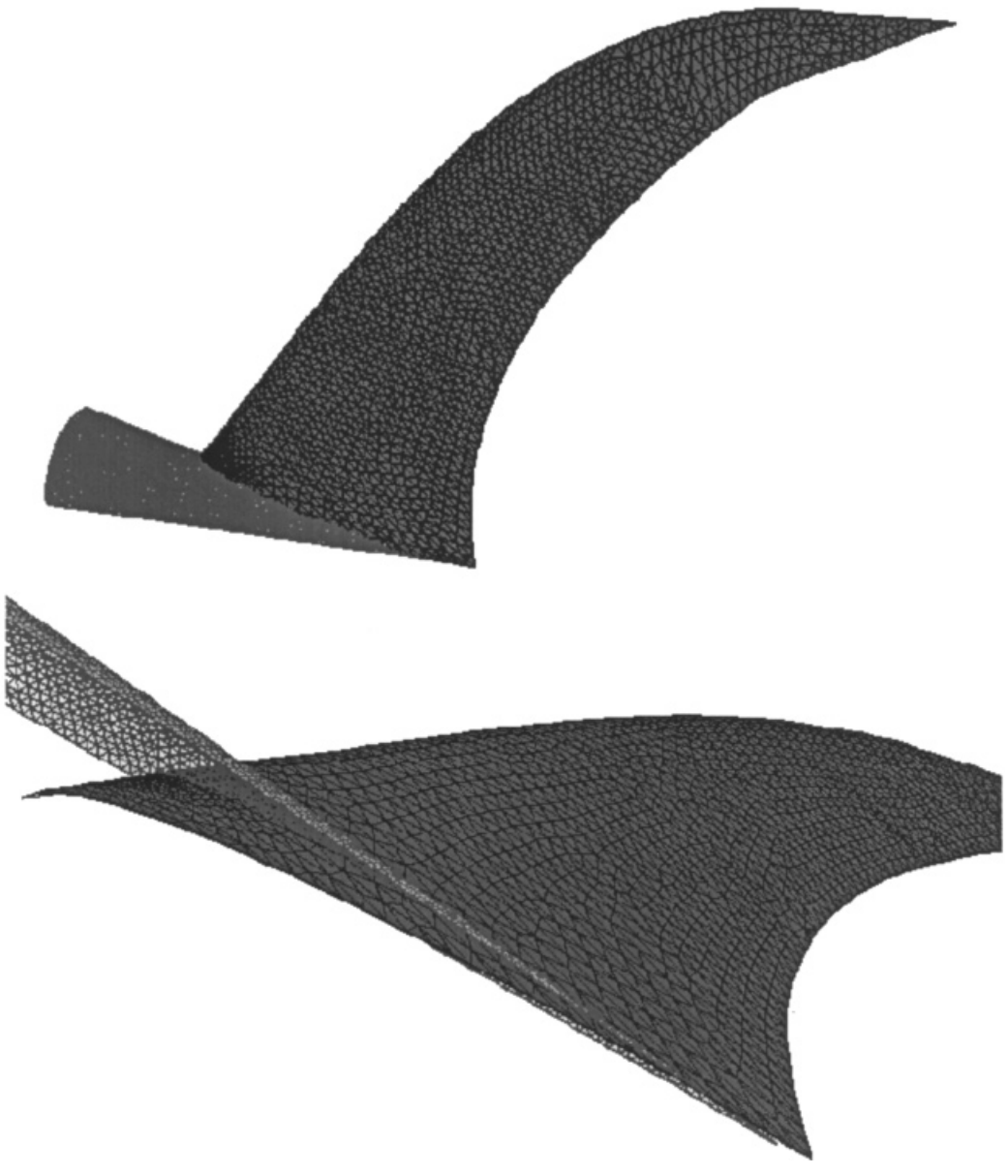

Figure 7. Intersection of the caudal fin and the tail of robotuna

The volume mesh consists of 236 427 elements and 46 918 points. Figures 11 and 12 show the outside view of the volume mesh and cut through the volume mesh. Figure 13 demonstrates the results for steady flow at $Re = 1{\cdot}0$, based on the average diameter of each sphere.

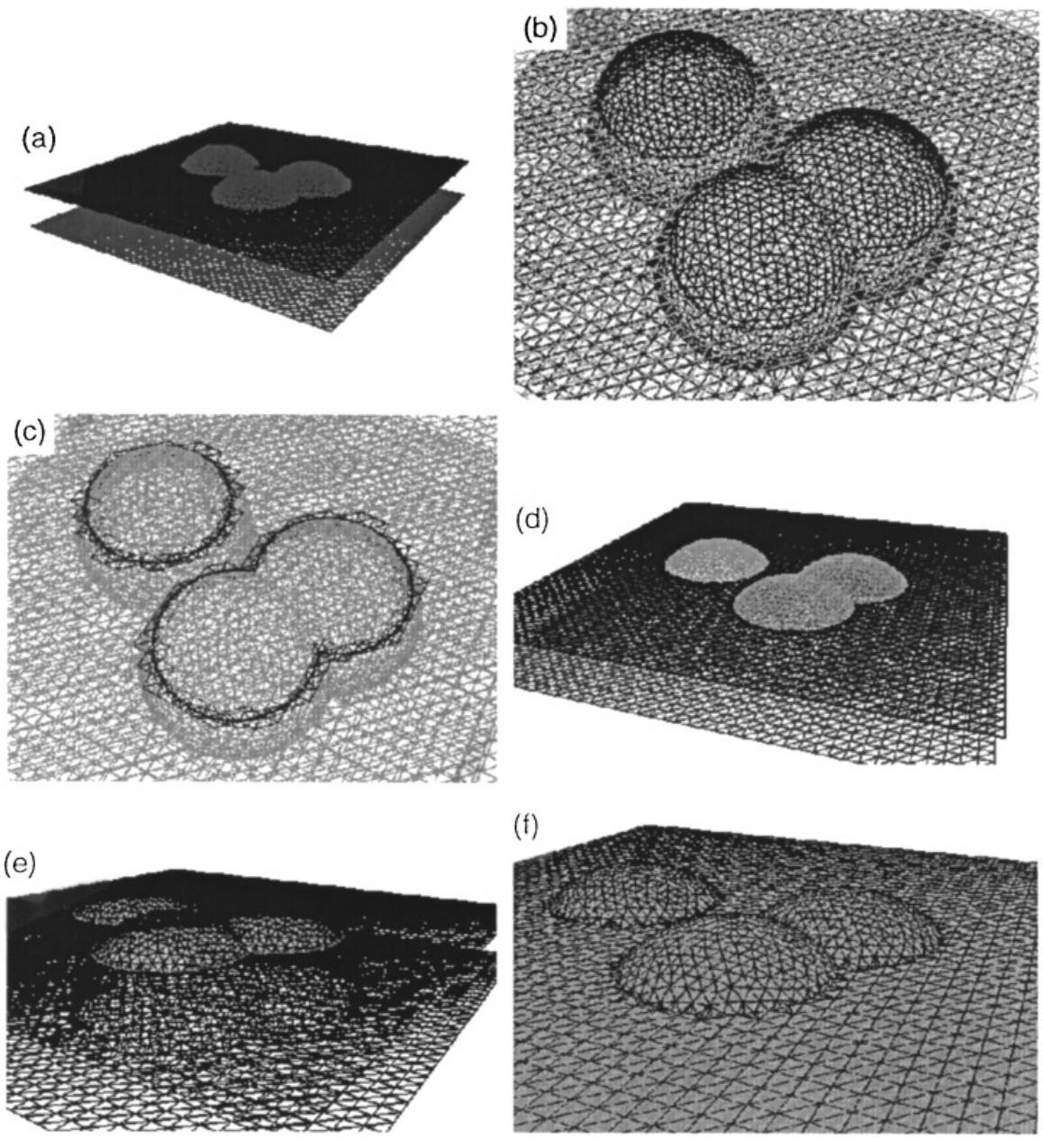

Figure 8. Intersection of spheres and planes: (a) surfaces to intersect; (b) computed intersection line; (c) marked intersecting elements; (d) surfaces resulted from the intersection; (e) selected surfaces to retain; (f) final surface

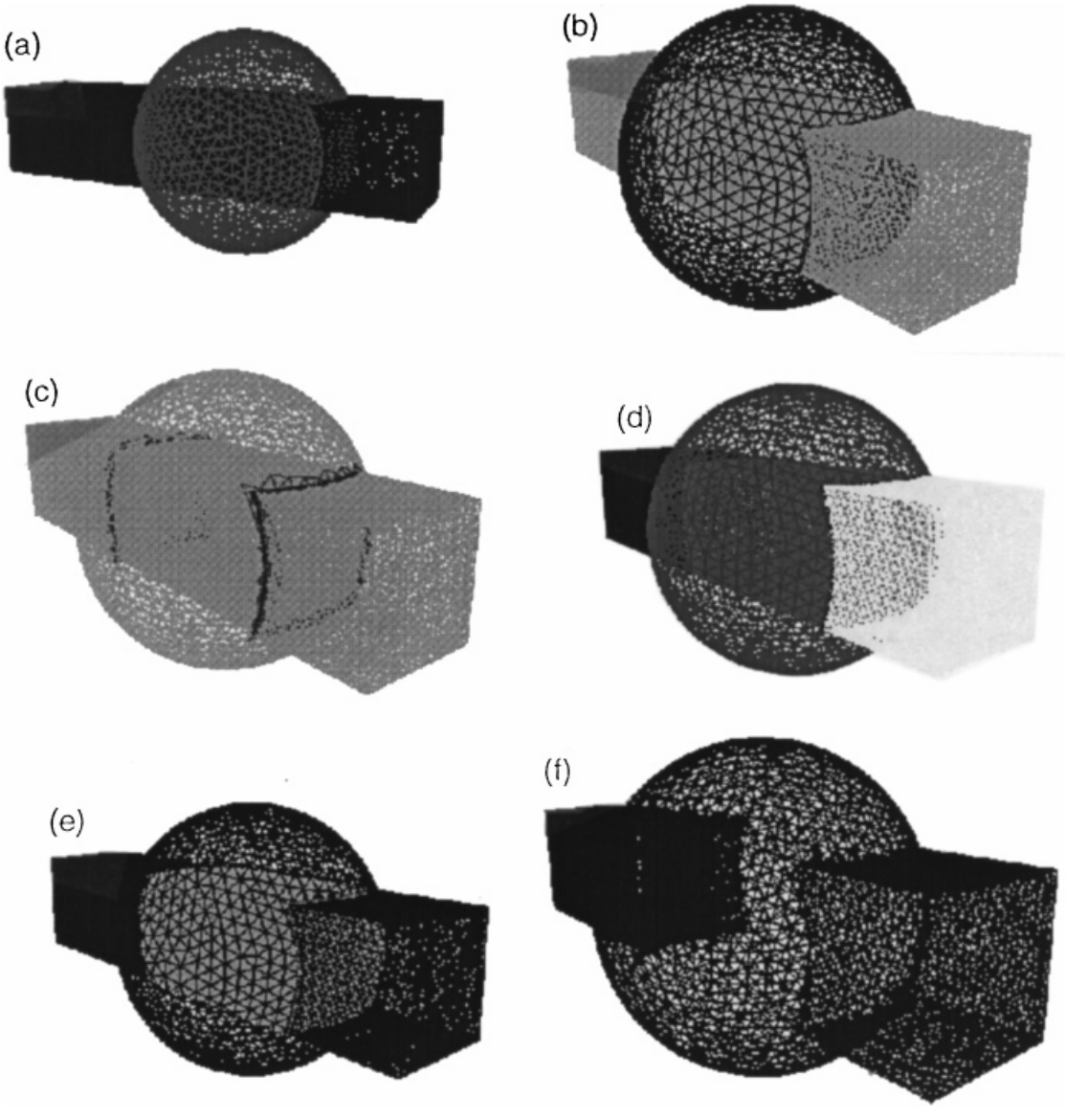

Figure 9. Intersection of the channel and sphere: (a) surfaces to intersect; (b) computed intersection line; (c) intersecting elements; (d) surfaces resulted after intersection; (e) external/internal identified; (f) final joined surfaces

4. PERFORMANCE

The algorithm was implemented in C/C++ programming language using SGI C++ compiler and benchmarked on a Silicon Graphics Indigo workstation with MIPS R8000 175 MHz IP26 processor. The results from several applications are summarized in Table I.

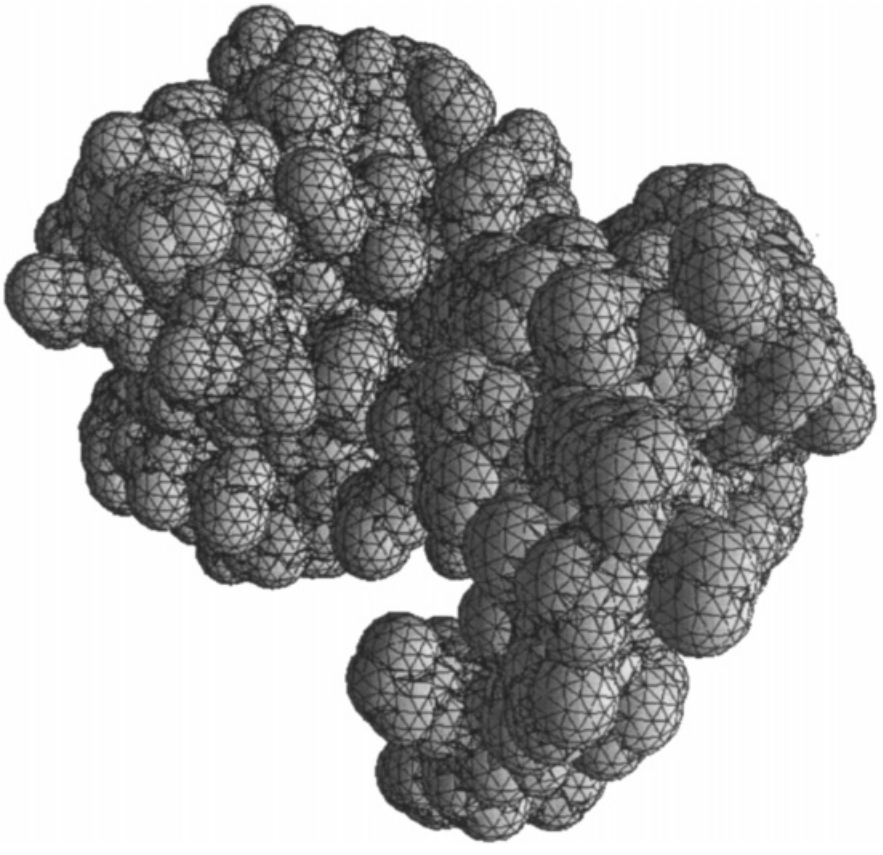

Figure 10. Automatic component-based grid generation, 2000 intersecting spheres (clusters from human hemoglobin)

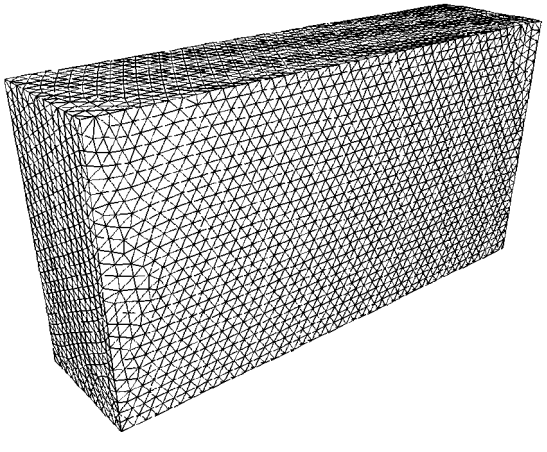

Figure 11. Volume mesh around molecular cluster, outside view

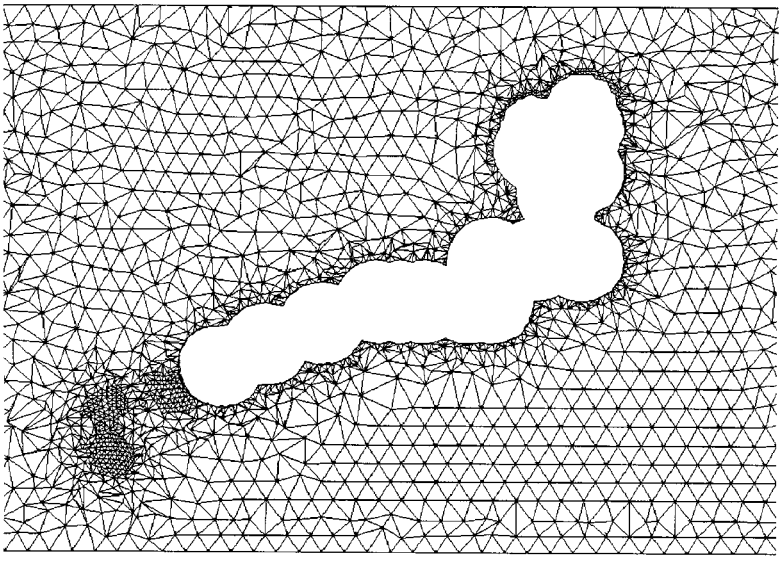

Figure 12. Cut through the volume mesh around molecular cluster

Typically, 80–90 per cent of time is spent computing the intersection and the remaining time is spent on re-triangulation. The expense for identifying external/internal parts and for the surface improvement is negligible. The results show that the algorithm is fast and computing time depends mostly on the size and shape of the intersection patterns, and is not very sensitive to the cumulative input surfaces size. The effective spatial enumeration and search techniques are essential to the algorithm performance. The performance figures are the prime focus at the present time and likely to change in the nearest future. We believe that significant improvements in speed can still be realized.

5. CONCLUSION AND OUTLOOK

A method for the generation of unstructured grids over component-based geometries has been described. The method is fast, simple to use and is based on an algorithm that automatically generates surface meshes from given intersecting triangulated surfaces by using Boolean intersection/union operations. A variety of complex shapes can be built by computing the intersection of predefined surfaces. The algorithm is intended for rapid, interactive construction of non-trivial surfaces for engineering design, manufacturing, visualization and molecular modelling applications. Techniques to make the method fast and general are described. The algorithm is demonstrated on a number of engineering examples including generation of surface and volume meshes around clusters of intersecting components followed by the computation of flow field parameters. The developed algorithm is currently being used by engineers and has proven to be a very useful complement to existing CAD and finite element mesh generation modules. The technique described in the current work is based on triangular elements, but can be extended to general elements by splitting them into triangles before application of the method. Future work will focus

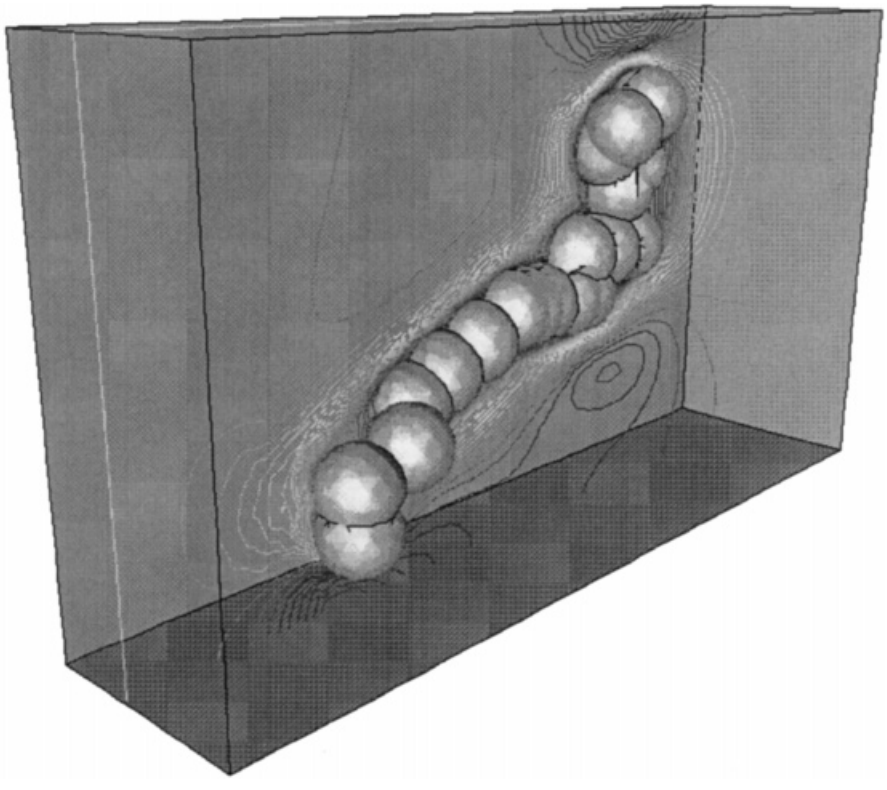

Figure 13. Uniform flow around molecular cluster, velocity contours, 3-D view 236 427 elements, 46 918 points, 13 834 boundary points

Table I. Performance statistics

Number of triangles in first surface	Number of polygons in second surface	Intersection time (s)	Total time (s)	Comments
100 854	2536	28·93	41·2	Clusters from human hemoglobin
100 854	336	2·98	7·1	Clusters from human hemoglobin
80 000	3034	8·37	10·2	Plane and sphere intersection
3034	3200	0·4	0·7	Planes and spheres
6270	3500	4·5	6·0	Planes and spheres
8200	8200	31·2	34·00	Two cylinders

on extending the present capabilities and improving the efficiency of the intersecting mesh generation module. A multithreaded version of the algorithm is currently being developed to take advantage of multiprocessing architectures. Extension of the current work will include computation of flow fields in and around porous media.

ACKNOWLEDGEMENTS

It is a pleasure to acknowledge the help of Dr. Ravi Ramamurti from the Laboratory for Computational Physics and Fluid Dynamics, Naval Research Laboratory. This work was funded by the Office of Naval Research through the Naval Research Laboratory.

REFERENCES

1. R. Löhner and P. Parikh, 'Three-dimensional grid generation by the advancing front method', *Int. J. Numer. Meth. Fluids*, **8,** 1135–1149 (1988).
2. A. Shostko and R. Löhner, 'Three-dimensional parallel unstructured grid generation', *Int. J. Numer. Meth. Engng.*, **38,** 905–925 (1995).
3. R. Löhner, 'Extending the range of applicability and automation of the advancing front grid generation technique', *34th Aerospace Sciences Meeting and Exhibit*, Reno, NV, January 1996.
4. D. F. Watson, 'Computing Dirichlet tesselation', *Comput. J.*, **24**(2), 162–167 (1981).
5. N. P. Weatherhill, 'Delaunay triangulation in computational fluid dynamics', *Comput. Math. Appl.*, **24**(5/6), 129–150 (1992).
6. M. S. Shepard and M. K. Georges, 'Automatic three-dimensional mesh generation by the finite octree technique', *Int. J. Numer. Meth. Engng.*, **20,** 1965–1990 (1991).
7. T. D. Blacker and M. B. Stephenson, 'Paving: a new approach to automated quadrilateral mesh generation', *Int. J. Numer. Meth. Engng.*, **32,** 811–847 (1991).
8. G. Farin, 'Curves and surfaces for computer-aided geometric design, A practical guide', Academic Press, New York, 1988.
9. S. H. Lo, 'Automatic mesh generation over intersecting surfaces', *Int. J. Numer. Meth. Engng.*, **38,** 943–954 (1995).
10. M. J. Aftosmis, N. J. Berger and J. E. Melton, 'Robust and efficient Cartesian mesh generation for component-based geometry', *35th AIAA Aerospace Sciences Meeting and Exhibit*, 6–9 January 1997.
11. W. J. Schroeder, J. A. Zarge and W. E. Lorensen, 'Decimation of triangle meshes', *Comput. Graphics*, **26**(2), 65–70 (1992).
12. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, 'Surface reconstruction from unorganized points', *Comput. Graphics*, **26**(2), 1992.
13. J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics, Principles and Practice*, Addison-Wesley, Reading, MA, 1990.
14. G. Turk, 'Re-tiling polygonal surfaces', Siggraph 1992 Proc., *Computer Graphics*, **26,** 55–64 (1992).
15. A. Murzin, S. Brenner, T. Hubbard and C. Chothia, 'SCOP: a structural classification of proteins database for the investigation of sequences and structures', *J. Mol. Biol.*, **247,** 536–540 (1995).
16. R. Löhner, 'Some useful data structure for the generation of unstructured grids', *Commun. Appl. Numer. Meth.*, **4,** 123–135 (1988).
17. J. Esakov and T. Weiss, 'Data Structures, an Advanced Approach using C', Prentice Hall Software Series, Englewood Cliffs, N.J., 1989.
18. A. I. Fedoseyev, S. V. Purtov, I. I. Petrenko, P. I. Lazarev and V. S. Sivozhelev, 'Mathematical modeling of 3D protein molecule potential in nonlinear media', '*Physique en Herbe 92*', Congress, Marseille, 6–10 July 1992.
19. F. M. Richards, 'Areas, volumes, packing and protein structure', *Ann. Rev. Biophys. Bioengng.*, **6,** 151–176.
20. A. Shrake and J. A. Rupey, 'Environment and exposure to solvent of protein atoms', *J. Mol. Biol.* (1973).
21. T. J. Richmond, 'Solvent accessible surface area and excluded volume in proteins', *J. Mol. Biol.* (1984).
22. A. Varshney and F. Brooks, 'Fast analytical computation of Richard's Smooth molecular surface', *Visualization Proc.*, 1993.
23. R. J. Zauharr, 'SMART: a solvent-accessible triangulated surface generator for molecular graphics and boundary element applications', *J. Comput.-Aided Mol. Des.*, **9,** 149–159 (1995).
24. M. L. Connolly, 'Solvent-accessible surfaces of proteins and nuclear acids', *Science*, **221,** 1983.
25. F. M. Richards, 'Areas, volumes, packing and protein structures', *Ann. Rev. Biophys. Bioengng.*, **6,** 151–176 (1977).
26. L. Stryer, *Biochemistry*, W. H. Freeman and Company, San Francisco, 1995.
27. B. K. Lee and F. M. Richards, 'The interpretation of protein structures: estimation of static accessibility', *J. Mol. Biol.*, **55,** 379–400 (1971).
28. A. Shostko and R. Löhner, 'Parallel adaptive finite element flow solver', *Proc. of AIAA 12th Computational Fluid Dynamics Meeting*, San Diego, June 1995.
29. J. Blomenthal and K. Ferguson, 'Polygonization of non-manifold implicit surfaces', *Siggraph 1995 Conf. Proc.*, 1995.
30. W. Sandberg, A. Shostko, U. Obeysckare and C. Williams, 'Accessible surface and excluded volume of atomic clusters in shear flows', *J. Mol. Phys.*, submitted.
31. V. Pratt, 'Direct least-squares fitting of algebraic surfaces', *Comput. Graphics, SIGGRAPH'87 Proceedings*, **21**(4), 145–152 (1987).

