INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2002; **39**:703–717 (DOI: 10.1002/fld.344)

An adaptive Cartesian grid generation method for 'Dirty' geometry

Z. J. Wang[1,*,†] and Kumar Srinivasan[2]

[1]*Department of Mechanical Engineering, Michigan State University, East Lansing, MI 48824, U.S.A.*
[2]*CIMS 481-41-01, DaimlerChrysler Corporation, Auburn Hills, MI 48326, U.S.A.*

SUMMARY

Traditional structured and unstructured grid generation methods need a 'water-tight' boundary surface grid to start. Therefore, these methods are named boundary to interior (B2I) approaches. Although these methods have achieved great success in fluid flow simulations, the grid generation process can still be very time consuming if 'non-water-tight' geometries are given. Significant user time can be taken to repair or clean a 'dirty' geometry with cracks, overlaps or invalid manifolds before grid generation can take place. In this paper, we advocate a different approach in grid generation, namely the interior to boundary (I2B) approach. With an I2B approach, the computational grid is first generated inside the computational domain. Then this grid is intelligently 'connected' to the boundary, and the boundary grid is a result of this 'connection'. A significant advantage of the I2B approach is that 'dirty' geometries can be handled without cleaning or repairing, dramatically reducing grid generation time. An I2B adaptive Cartesian grid generation method is developed in this paper to handle 'dirty' geometries without geometry repair. Comparing with a B2I approach, the grid generation time with the I2B approach for a complex automotive engine can be reduced by three orders of magnitude. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: grid generation; adaptive Cartesian; dirty geometry; unstructured

INTRODUCTION

Impressive progresses in computational fluid dynamics (CFD) have been made during the last two decades in many aspects including algorithms for grid generation and flow computation. As a result, CFD is increasingly used in the design process in many industries such as aerospace, automobile, and many others. Depending on the type of computational grids used, CFD solution algorithms can be classified as structured and unstructured grid methods. The structured grid method was popularized with the development of body-fitted-co-ordinate (BFC)

* Correspondence to: Z. J. Wang, Department of Mechanical Engineering, Michigan State University, East Lansing, MI 48824, U.S.A.
† E-mail: zjw@egr.msu.edu

Contract/grant sponsor: U.S. Navy; contract/grant number: N68335-98-C-0233

Received June 2001
Revised January 2002

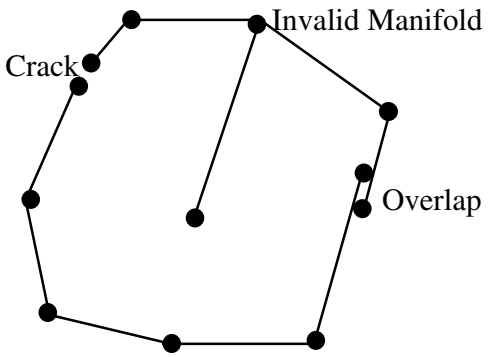


Figure 1. Problems areas of a dirty geometry in 2D.

grid generation approaches, which include the elliptic, hyperbolic, and algebraic grid generation techniques [1–5]. Although the structured grid method has been successful in handling complex geometries, it is usually very time consuming to generate BFC grids for complex geometries. The difficulty in generating structured grids and the desire to automatically compute flows over complex geometries spawned a surge of activities in the area of unstructured grids during the last one-and-half decades [6–18]. Unstructured grids provide considerable flexibility in tackling complex geometries and for adapting the computational grids according to flow features. Types of unstructured grids include classical triangular or tetrahedral grids, quadrilateral or hexahedral grids, prismatic grids, or mixed grids, and more recently adaptive Cartesian [19–23], Cartesian/prism [24, 25] and viscous Cartesian grids [26, 27].

With this impressive array of powerful grid generation approaches, the difficulty in grid generation has shifted from volume grid generation to surface grid generation with 'non-water-tight' geometry definitions, which contain topological defects such as gaps or overlaps. Many CAD packages use boundary representation (BREP) geometries to define solids [28]. With BREP, the boundary surface is composed of multiple boundary patches, which can be planar polygons, or non-uniform rational B-spline (NURBS) patches. The BREP geometry is said to be 'water-tight' if each boundary curve (of a patch) is shared and shared only by two patches. In the design process, design engineers use CAD packages to perform detailed geometry designs. The design of a typical system (for example an automobile engine) involves thousands of parts. The exact transfer of geometries from one CAD package to another, and from a CAD package to a grid generator is still an unresolved issue. Even if the CAD model is exactly transferred from a CAD package to a grid generator, the CAD model may still not be suitable for grid generation. To generate a computational grid with almost any current grid generators, a 'water-tight' geometry has to be defined first. If a geometry is 'water-tight', then the geometry is said to be topologically closed. In most cases, 'dirty geometries' are caused by cracks, overlaps or invalid manifolds in the geometry, which are put there by designers due to either mistakes or carelessness. In two dimensions (2D), these problems are illustrated in Figure 1. Note that in 2D, each end point of a curve must be and only be shared by two curves (lines) if the geometry is considered 'water-tight'. These geometric problems can be fixed easily for simple geometries. For reasonably complex geometry, geometry repairing can become very tedious and time consuming.

In the automobile industry, the stereolithography (STL) format is often used to transfer the designed geometries from one package to another for visualization, and grid generation. The use of STL has many advantages. First, the basic parts are defined using polygons (mostly triangles), which are the easiest to visualize on a computer screen. Second, many efficient computational geometry algorithms have been developed for triangles to perform operations such as intersections, projections, etc. Third, most CAD packages can output geometries in STL format, and lastly the transfer of STL files from one system to another is exact.

The use of STL files also has its disadvantages. First, STL-defined solids are usually NOT 'water-tight', which means that the geometry may have cracks and overlaps. Second, the use of planar facets to represent curved geometries inevitably result in the loss of accuracy. Third, the number of triangles used to define a solid is usually dictated by the solid surface curvature. Small triangles are often used in high-curvature regions, and large triangles are used in flat regions. The size of these triangles differs considerably. In order to perform a meaningful CFD simulation, the surface often needs to be re-meshed. For a 'water-tight' STL geometry, the remeshing is a relatively easy operation. The advancing front algorithm [6–8] can be used to remesh the surface. If the geometry is 'dirty', however, the remeshing operation can be prohibitively expensive. One such example is shown in Plate 1, which displays a 'dirty' automobile engine geometry. The geometry has 32 STL patches with cracks and overlaps. In addition, the geometry has topological problems in that some edges are shared by more than two triangles. Note that for a 'water-tight' geometry, each edge must be and only be shared by two triangles. The yellow lines in Plate 1 show the topological cracks, and the red lines are edges with topological problems. One can imagine the difficulty in repairing this 'dirty' geometry and remeshing the surface of the geometry. It is estimated that it can take a highly skillful grid generation engineer 2–3 months to repair this geometry for grid generation using the-state-of-the-art tools.

We cannot help asking the following question: Is it absolutely necessary to repair the geometry before grid generation can take place? Unfortunately, the answer is yes for nearly all the traditional grid generation approaches we have seen so far, be it the structured or unstructured grid approaches. All these grid generation approaches need a 'water-tight' geometry as the starting point.

In this paper, we advocate a different grid generation philosophy from the traditional grid generation approaches in that the interior volume grid is generated first before a surface grid is generated. Then the interior volume grid is 'intelligently' connected with the boundary geometry. We argue that a unique advantage of this new grid generation method is that 'dirty' geometries can be meshed without being repaired first. We will call the traditional grid generation approaches boundary to interior (B2I) approaches, and the new method a interior to boundary (I2B) approach following the popular internet-based naming convention of business to business (B2B) and business to consumer (B2C).

Another I2B approach was developed by Schneiders [14] for generating hexahedral grids. Although the approach is similar to the I2B approach developed by the current authors, there are dramatic differences. First, Schneiders' approach requires 'water-tight' geometry. The intersection operations are designed for 'water-tight' geometries only. Second, the volume grid used in Reference [14] is a uniform Cartesian grid, which is not capable of supporting local grid refinement and coarsening. Finally, the current I2B approach is developed for a very general class of geometry entities (to be defined later) with demonstration performed with STL geometries.

The paper is, thus, organized as follows. In the next section, we present the new philosophy behind the I2B grid generation method. In order to handle 'dirty' geometries, a new definition for geometric entities is presented. Based on this definition, a general I2B grid generation approach is then given. After that, a particular I2B grid generation approach using adaptive Cartesian grid is presented. Implementation issues such as data structures, algorithm efficiency are addressed. Next, several demonstration cases with complex 'dirty' geometries are presented. Finally, conclusions are included in the last section to conclude the paper.

B2I versus I2B grid generation approaches

The first step in a CFD simulation involving non-trivial geometries is to import a geometry, define a closed computational domain, and generate an adequate computational grid. For external flow problems, a user must also define a truncated far-field boundary such that the computational domain is finite and closed. Because the shape of the far-field boundary usually does not affect the solution, any elementary shapes such as a sphere or a cube can be used. For internal flow problems, the user may need to define an inlet or an exit boundary to close the computational domain. Generally, the grid generation process can be broken into the following steps:

- acquire the geometry;
- define a water-tight (closed) computational domain and repair the geometry if necessary;
- define topology (for the structured grid approach);
- generate the computational grid on the boundaries;
- generate the computational grid (i.e. the volume grid) in the interior of the computational domain.

As can be seen that the conventional grid generation methods MUST start from a boundary grid, and then the interior grid is generated based on the grid on the boundary. The process is illustrated in Figure 2. Note that all structured grid generation approaches also fall in this category. If a 'dirty' geometry is imported, the geometry must be cleaned or repaired so that a 'water-tight' geometry can be defined. Geometry repair can be an extremely time-consuming business if the geometry is complex. It involves very tedious manual labour of an experienced grid generation engineer.

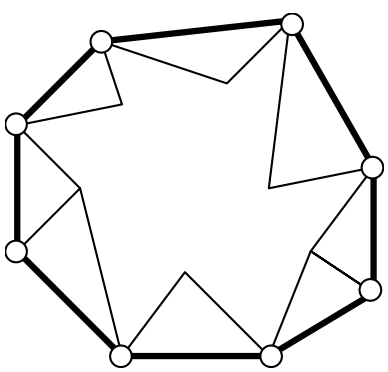


Figure 2. The schematic of a traditional B2I grid generation approach.

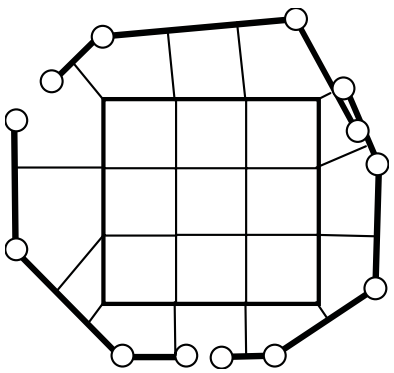


Figure 3. Schematic of the new I2B grid generation approach.

It seems that the only possibility of eliminating geometry repairing is to somehow reverse the grid generation process. Instead of generating the boundary grid first, the interior volume grid must be generated first, and then the interior grid is 'connected' with the boundary. In this case, we do not need a 'water-tight' geometry to start the grid generation process, and the approach has the potential of completely eliminating geometry repair from grid generation. A schematic of this grid generation approach is shown in Figure 3. In Figure 3, the 'interior' Cartesian grid is generated first. Then the Cartesian grid is 'connected' to the boundary (which has two cracks and one overlap) through projections, i.e. to connect the volume grid to the boundary in the minimum distance direction. This is only one means of 'connecting' the volume grid to the boundary. Other techniques are definitely possible. In order to present the new I2B grid approach, we need to precisely define what a geometric entity is. Generally speaking, a geometry is usually represented by a group of surface patches, or solids. A surface patch can be defined with a variety of formats, such as a triangulated surface, a Coons patch, a NURBS surface, or a computational grid with a square topology. There are also many different ways of defining a solid. One way is the constructive solid geometry. In order to support any geometry in an arbitrary manner in the present I2B grid generation method, the definition of a geometric entity must be generalized. Besides solids and surface patches in any format, discrete points and curves can also serve as a geometric entity. In order to support 'dirty' geometries, the geometric entities defined here do not have 'inside' or 'outside'. Instead, a geometric entity is defined to be any entity supporting the following two operations:

(1) Given a simple solid (e.g. a cube or a tetrahedron), the entity is capable of returning a status 'intersected' or 'non-intersected' based upon whether the entity intersects the given solid. Let the geometric entity be represented by G (which is defined as a set of points), the given solid by S. The intersection operation is $I(G,S)$ is then defined by

$$I(G,S) = \begin{cases} 1 & \text{if } G \cap S \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

(2) In the Euclidean space, given an arbitrary point (q) in space, the projection (p) from the given point to the entity is well defined. The line segment from the given point to

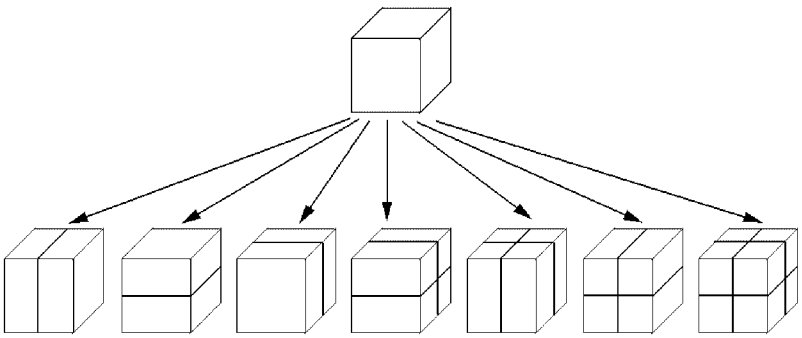


Figure 4. Arbitrary cell subdivision supported by 2^n tree.

the projection is the shortest distance from the given point to the entity, i.e.

$$p(G,q) := |pq| \leqslant |rq|_{r \in G}$$

Note that this definition of geometric entity is very general, and any geometry defined with a solid or a surface patch can be seen to be a valid geometric entity, since these operations can be easily implemented. Note that any discrete points, lines, curves, and planes are also valid geometric entities.

Before we present the general I2B grid generation approach for a given set of geometric entities, two meshing parameters, $d_{\min}$ and $d_{\max}$ are given. They represent the minimum and maximum sizes of grid cells to be generated. The only requirement that the set of geometric entities must satisfy is that the computational domain formed with the entities is 'physically' closed if gaps or holes smaller than $d_{\min}$ are ignored. This is to say that if a gap or a hole exists in the geometry (which should not have been there), the size of the gap or the hole must be smaller than $d_{\min}$. Note that this enclosure condition is much weaker than the condition of 'water-tightness' required by B2I approaches. Obviously, if all the gaps between the line segments in Figure 3 are smaller than $d_{\min}$, the set of line segments actually defines a valid computational domain. There is one other possible complication the user must clarify. For the computational domain defined with the line segments as shown in Figure 3, the user needs to decide whether the 'inside' or the 'outside' should be the computational domain. For the grid shown in Figure 4, the 'inside' represents the computational domain.

With the above definition and clarifications, we are ready to present our general I2B grid generation approach.

(1) determine the size of the computational domain, and fill the computational domain with a particular type of simple solids or cells (cubes or tetrahedra) whose sizes are between $d_{\min}$ and $d_{\max}$;
(2) determine the cells which intersect the set of geometric entities;
(3) recursively refine the cells intersected by the geometric entities until all the cells are smaller than a given threshold d_{int} (which is usually chosen to be $2d_{\min}$). This requirement guarantees that a minimum grid resolution near all geometric entities is achieved. Note that all cells must be bigger than $d_{\min}$;

(4) select one cell in the computational domain, and use a neighbour-painting algorithm to identify all cells in the computational domain;
(5) remove all the cells intersected by the geometric entities, and all unpainted cells;
(6) remove cells too close to the geometry;
(7) connect the exposed Cartesian faces to the geometric entities through projections.

A schematic showing the major steps of the method is showing in Plate 2.

Next, we present an I2B grid generation approach based on the adaptive Cartesian grid method. The viscous Cartesian grid method developed by Wang *et al*. [26, 27] is further extended to be a truly I2B method capable of handling complex dirty geometries.

I2B ADAPTIVE CARTESIAN GRID METHOD

Supported geometric entities

As mentioned earlier, any entities supporting the operations defined in the last section can be considered as geometric entities. In this paper, however, we have limited ourselves to points, lines, line segments and triangulated surfaces. This choice is really dictated by the type of geometric inputs we usually get, and does not imply that other choices are not possible. In particular, in many cases, especially in the automotive industry, the input geometry is in the STL format because of its portability and popularity. The fact that the surface is defined by triangles (or polygons in general) makes the geometry easy to display, transport, and manipulate. This format can be exported by nearly all the major CAD packages, and is independent of the CAD systems where the surface is created. Each STL file can be viewed as a separate 'part', and a system of parts can be produced by concatenating all the part files. This feature is particularly useful in the early design stage, in which many of the parts undergo constant design modifications.

Surface grids in other formats such as PLOT3D, FAST and PATRAN formats can be easily converted into triangulated surfaces, and serve as geometric entities. For example, given a PLOT3D structured surface grid, each quadrilateral of the surface grid can be divided into two or four triangles, thus producing a triangulated surface.

If one is interested in dealing with NURBS or IGES patches, one approach is to generate a 'structured grid' for the patch using the local co-ordinates in the parameter space. This structured grid is then subdivided into a triangulated surface. This triangulated surface can be viewed as a 'finite resolution' representation of the true NURBS or IGES patch.

Interior grid generation

Once the geometric entities are given, the next step is to define the computational domain. Based on whether the problem is an internal or external flow problem, the size of the computational domain can be determined. For an external flow problem, there is usually a characteristic body length L (the chord length of an airfoil, or the length of an aircraft). The far-field boundary should usually be at least 10 times the body length away from the geometry so that the location of the far-field boundary does not influence the solution quality significantly. For an internal flow problem, there are usually an inlet and exit. If the inlet and exit are not defined, the user must first define the inlet and exit geometry based on experiences and

common sense. For example, inlet and exit surfaces can be usually assumed to be planar. With properly defined inlets and exits, the physical domain of interest should be physically closed excluding gaps or holes smaller than $d_{\min}$. The computational domain must be big enough to contain the physical domain of interest.

Once the size of the computational domain is determined, we are ready to generate the interior grid. 'Interior' here means 'inside' the computational domain, and not necessarily inside a geometry (e.g. for external flow problems). To generate a computational grid inside a computational domain, the easiest approach may be to use a uniform Cartesian grid. However, in this paper, 2^n tree adaptive Cartesian grids are used following the viscous Cartesian grid approach [26, 27]. The adaptive Cartesian grids can be clustered or de-clustered in a truly arbitrary fashion. In the 2^n tree data structure, one tree node can have 2, 4, or 8 children as shown in Plate 3. This tree can be used to record the recursive subdivisions of a single Cartesian cell in an arbitrary fashion as shown in Figure 4. Note that a Cartesian cell using the 2^n tree data structure can be subdivided in one, two or all three co-ordinate directions. The popular Octree data structure can be viewed as a special case of the 2^n tree.

To start the adaptive Cartesian grid generation process, a single-root Cartesian cell covering the entire computational domain is produced first. This cell is called a root cell because it occupies the top level—the root—of the 2^n tree. This root cell is recursively subdivided so that all the Cartesian cells—the leafs in the 2^n tree—satisfy the minimum grid resolution, i.e.

$$\Delta x \leqslant d_{\max}, \quad \Delta y \leqslant d_{\max}, \quad \Delta z \leqslant d_{\max}$$

Then for each geometric entity, all the Cartesian cells intersecting the entity are again recursively refined so that all the Cartesian cells intersecting the geometric entity satisfy another resolution requirement, i.e.

$$\Delta x_i \leqslant d_{\text{int}}, \quad \Delta y_i \leqslant d_{\text{int}}, \quad \Delta z_i \leqslant d_{\text{int}}$$

Note that since all geometry entities support the 'intersection' operation, it is easy to identify which Cartesian cells are intersected by the geometric entities. In the case of a triangulated surface, each triangle is used to 'intersect' the Cartesian cells. To determine whether a triangle intersects a Cartesian cell, the bounding box of the triangle is computed first to obtain ($x_{\min}$, $y_{\min}$, $z_{\min}$) and ($x_{\max}$, $y_{\max}$, $z_{\max}$). A necessary condition for the triangle to intersect a Cartesian cell is that the bounding box must overlap the Cartesian cell. If they do, a triangle–cube intersection algorithm is used to further test whether they intersect each other.

The next step is a significant departure from the approaches presented in References [26, 27]. In order to support 'non-water-tight' geometries, the 'inside' or 'outside' of geometries are deliberately not defined. Because each geometry entity may have a different orientation, it is not possible to use the normal of an entity as an indication as to which is inside the computational domain. Instead, the user needs to select one single Cartesian cell inside the computational domain, as shown in Plate 3(c). Note that we have identified cells which intersect the geometric entities, as shown in Plate 3(b). These intersected cells also serve to divide the 'interior cells' and the 'exterior cells' (cells outside the computational domain). Then by using a neighbour-painting algorithm, all the Cartesian cells inside the computational domain can be determined. This neighbour-painting algorithm is again recursive in nature. The actual painting starts from the cell which the user selects. This cell is first painted as 'interior' (i.e. a flow cell). Then its neighbours are examined. If a neighbour is

not a 'intersected' cell (a cell which is intersected by a geometric entity), then this neighbour is also painted as 'interior'. This process is carried out for all the painted cells until none of the neighbours of the 'interior' cell are unpainted, as shown in Plate 3(d). At this stage, the painted cells are divided from unpainted cells by the 'intersected' cells. All the unpainted cells are then considered 'exterior' cells (i.e. cells considered outside the computational domain). After that, all the exterior cells and intersected cells are removed from the computational domain. Cartesian cells too close to the geometry (e.g. within $d_{\min}/2$ distance from the geometry) are also removed so that the gap between the Cartesian grid and the geometry is sufficiently large, as shown in Plate 3(e). A gap of reasonable size allows high-quality grid cells to be generated between the Cartesian grid and the geometry.

Connecting the interior grid to the boundary

At this stage, we have a region or regions of adaptive Cartesian grids, whose boundary faces form the so-called Cartesian front. For example, the Cartesian front generated with a missile geometry is shown in Figure 5. The Cartesian front is composed of Cartesian faces, with stair-step-kind of sharp corners. The Cartesian front must somehow be 'connected' to the

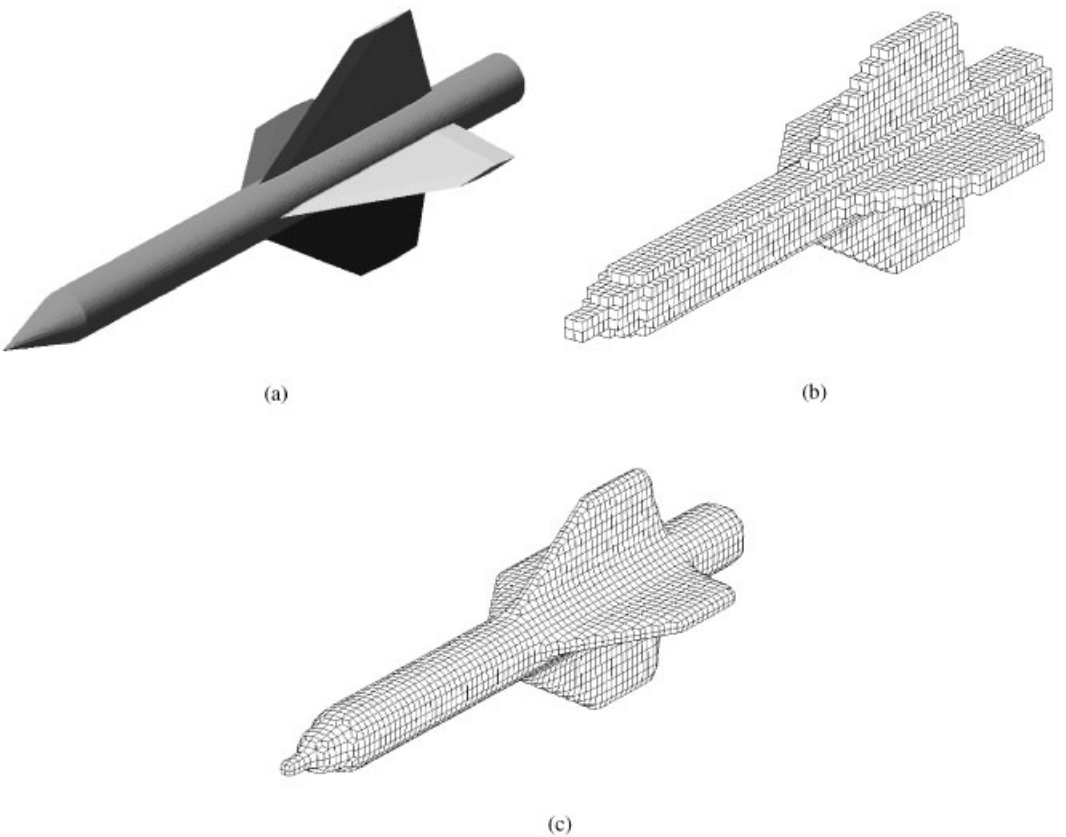


Figure 5. (a) A missile geometry, (b) the stair-step Cartesian front and (c) the smoothed Cartesian front.

geometric entities to produce a useful computational grid. There are possibly many different ways in which one can connect the Cartesian front to the boundary. The most obvious and robust way is probably through projecting the Cartesian nodes from the Cartesian front to the geometric entities. Before one does the projection, it is appropriate to smooth the Cartesian front so that the 'steps' in the Cartesian grid is smoothed out. This can be accomplished through the use of a Laplacian smoother, i.e.

$$\boldsymbol{r}_i^{\text{new}} = \boldsymbol{r}_i^{\text{old}}(1-w) + w\,\frac{1}{N_{\text{nb}}}\sum \boldsymbol{r}_{\text{c}}$$

where $\boldsymbol{r}_i$ is the position vector of a node on the Cartesian front, and N_{nb} is the number of Cartesian faces sharing node i, and $\boldsymbol{r}_{\text{c}}$ are the face center position vectors of the faces sharing node i, and w is a relaxation factor in [0 1]. The Laplacian smoothing algorithm can be applied several times to obtain a reasonably smooth front. Shown in Figure 5(c) is the smoothed Cartesian front after the Laplacian smoother is applied 4 times with $w = 0.5$. Note that the front is much smoother than the stair-step Cartesian front shown in Figure 5(b).

The smoothed front is then projected to the geometry according to the minimum distance rule. It can be proved mathematically that the projection lines cannot intersect each other away from the geometric entity. Note that per definition, the geometric entities must support the projection operation, which comes handy now. Because the Cartesian front is composed of boundary faces of a 'solid region', the front is closed and 'water-tight'. After the front is projected to the boundary geometric entities, a 'water-tight' surface grid is generated on the boundary. By connecting each point on the Cartesian front and the corresponding projected point on the boundary, we obtain a valid computational grid as shown in Plate 3(f). After the projection, a single layer of prism cells (quadrilateral cells in two dimensions) with arbitrary polygon footprints is generated in three dimensions. This single layer can be sub-divided into multiple layers with proper grid clustering near the geometry to resolve a viscous boundary layer if necessary.

The efficiency of the projection operation in three dimensions is critical to the success of the method. In a typical application, we usually have a triangulated surface with N triangles, and a Cartesian front with M nodes. A brute-force exhaustive search-based projection algorithm would take $O(MN)$ operations, which is too expensive even for medium-sized applications. Instead, an ADT tree [29] is used to record the bounding boxes of the triangles. Given a node to project, only triangles close to the node are identified from the tree-based search operation, and are projected to. This new algorithm reduces the number of operations from $O(MN)$ to about $O(M \log N)$. The speed-up for a medium-sized problem ($N = 100\,000$, $M = 100\,000$) can be more than several orders of magnitude.

A projection based on the minimum distance rule usually misses geometrically important concave features, such as the corner points in Figure 3. In order to preserve these features, they must be detected or specified first. One criterion is to detect all sharp edges based on the angle between the two faces sharing an edge. Then a feature-preservation technique is used to reconnect the front nodes to those features. This technique is schematically shown in Figure 6.

In many applications, it may be too expensive or unnecessary to preserve all the features in a geometric entity. Then the projection algorithm can serve as an automatic feature suppression operator. This capability will be demonstrated in a test case later.

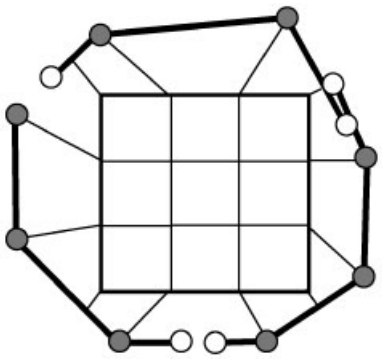


Figure 6. Schematic of feature preservation.

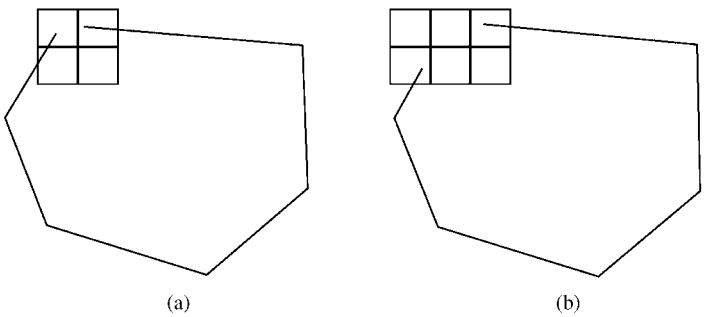


Figure 7. An invisible gap and a fully resolved gap.

Handling of cracks, overlaps and invalid manifolds

Recall that B2I approaches cannot handle cracks and overlaps because a 'water-tight' surface grid is a necessary starting point. In the current I2B approach, cracks smaller than the minimum grid resolution $d_{\min}$ are not 'visible' to the Cartesian grid generator, which is illustrated in Figure 7(a). However, if the geometry has an opening bigger than the minimum grid resolution, this opening is considered 'physical' by the Cartesian grid generator, and is fully resolved as shown in Figure 7(b). Fortunately, most of the cracks appearing in CAD STL files are due to slight mismatches between different patches. Therefore, the current I2B viscous Cartesian grid approach can handle these cracks without any problems because the minimum grid resolution is usually much larger than the size of the cracks. However, if large cracks exist in a geometry model, which are not physical, they have to be closed manually by the user. Fortunately, these large cracks are rare, and can be easily spotted through visual inspection of the generated Cartesian grid.

It turns out that overlaps are no problems for the I2B approach because overlaps do not alter the topology (inside and outside) of the Cartesian grid. Through the use of projections, the patch closest to the Cartesian front is always used. It is guaranteed that the projections would not intersect each other.

In the case of an invalid manifold caused by an extra piece of geometry 'outside' the computational domain, it is actually not a problem at all for the grid generator. However, if the piece is inside the computational domain, the grid generator will treat it as a 'thin' wall. As a result, volume grids on both sides of the extra piece of geometry will be generated. If the extra piece should not have been there, the user must manually remove this extra piece. Fortunately, invalid manifolds 'inside' the computational domain are usually rare, and can be easily spotted and removed.

It is easy to see that the I2B viscous Cartesian approach is completely 'topology' based, and cannot fail for arbitrarily complex geometries. This property has been confirmed with many cases involving complex 'dirty' geometries.

One difficulty with the current I2B approach is the control of grid quality near the geometry. A small number of cells may be skewed and have small volumes. The way we used to fix these problems is to merge the skewed cells with their neighbours to improve the grid quality. Since the flow solver can handle arbitrary polyhedra, this approach has been shown to be effective.

DEMONSTRATION CASES

Demonstration of automatic feature suppression

The I2B viscous Cartesian grid method can be fully automated. In most cases, the user needs only to input one parameter, i.e. the geometry surface grid resolution d_{int}. The minimum and maximum grid resolution can be determined based on the characteristic length scale of the input geometry. Another unique advantage of the method is automatic feature suppression. In many simulations, it is not necessary to resolve very fine geometric features, or it is too costly to resolve all the features. In those cases, we would like to suppress the fine geometric features. The viscous Cartesian grid can automatically suppress all features smaller than the surface grid resolution, i.e. smaller features than the grid resolution are smoothed out automatically. To demonstrate this capability, we again use an auto part as an example. Figure 8(a) shows the geometry of the part, and Figure 8(b)–8(d) shows the surface meshes with varying surface grid resolutions. It is obvious that the geometry is better resolved with finer grid resolutions.

Demonstration with more complex geometries

Several more complex geometries are shown here with generated surface grids. Shown in Plate 4 is the engine grid generated for the 'dirty' geometry shown in Plate 1. This grid has about 125 000 cells, and was generated in about 40 min on a Pentium III 450 MHz PC running the Linux operating system. With a traditional grid generator, it took 2–3 man-months to repair the geometry and generate a computational mesh for this geometry. A speed-up factor of more than three orders of magnitude was achieved in this case.

Plate 5 displays a more complex engine geometry, and the generated computational grid. Plate 6 shows the viscous Cartesian grid for a passenger cabin with six dummies. Both grids were generated within 4 h. Note that all the geometries are not 'water-tight', and the I2B viscous Cartesian grid method had no difficulty in handling them without any 'cleaning' or 'repairing'.

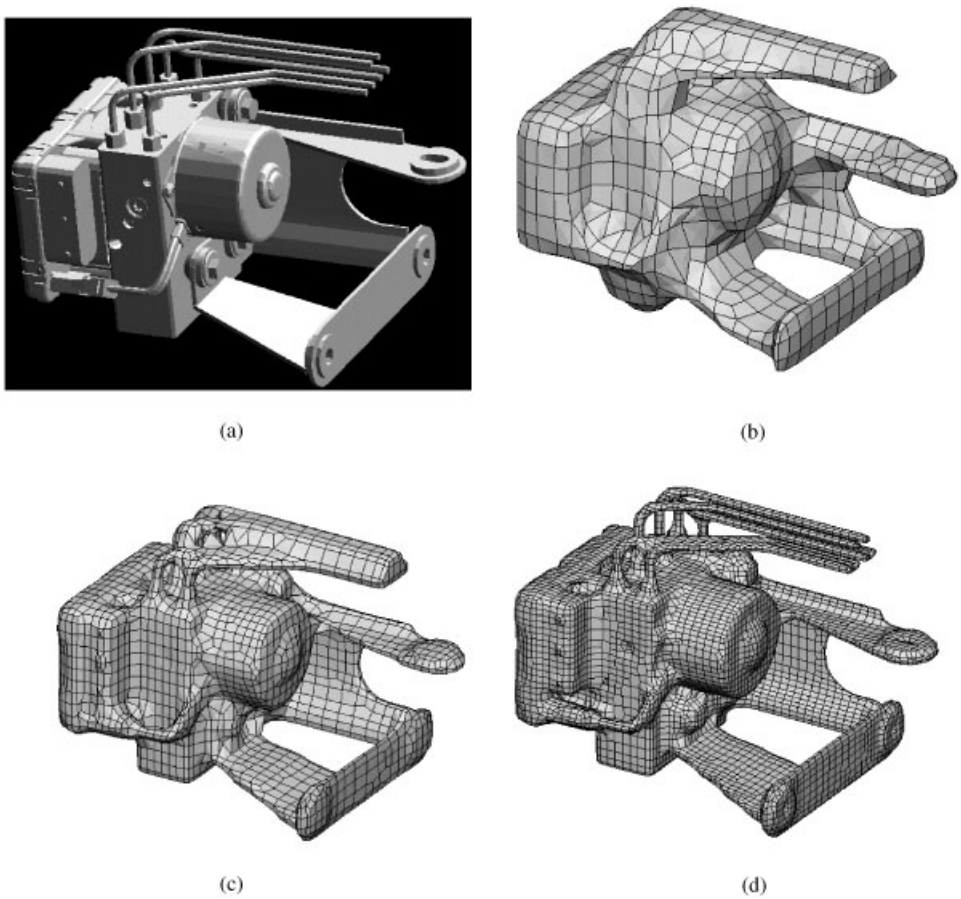


Figure 8. STL geometry of automotive part and surface grids with various resolutions.

Demonstration of feature recovery

In all the previous examples presented so far, no feature preservation is necessary. In aerospace applications, however, geometric feature preservation may be critical. For example, geometric features in an aircraft must be captured if one is to accurately predict the lift and drag of the aircraft. In this demonstration, a fighter aircraft is used as an example to demonstrate feature preservation. The input format of the aircraft is PLOT3D surface patches. The patches, however, do not form a 'water-tight' geometry surface. There are mismatches, holes and overlaps along the patch boundaries. As a matter of fact, there is one large hole in the geometry, which must be filled before grid generation can take place. The other cracks and holes are very small, and do not need any special treatment. In addition, geometrically important sharp edges are detected automatically, or specified by the user, as shown in Plate 7(a). Furthermore, several surface sources are used to cluster grid cells near sharp edges or high-curvature regions due to the flexibility offered by the adaptive Cartesian grid. A viscous Cartesian grid was then generated successfully with critical feature preservation. The computational grid is

shown in Plate 7(b). Note that the critical features were captured correctly. A sample flow computation was carried out, and the computed pressure distribution at Mach = 0.3 is shown in Plate 7(c), demonstrating the integrity of the computational grid. The total time spent in generating this grid is about a day, which includes both user time and CPU time to produce the grid.

CONCLUSIONS

In order to handle 'dirty' geometries with cracks and overlaps, a new grid generation approach, namely interior to boundary (I2B) approach, is advocated in this study. To support 'non-water-tight' geometries, a more general definition of a geometric entity is also given. Any geometry supporting the operations of intersection and projection can be used in grid generation. Therefore, the requirement of 'water-tightness' is completely avoided. Based on the generalized definition of geometric entities, a general I2B grid generation approach is then presented. The viscous Cartesian grid approach is further extended to be an I2B approach to handle arbitrary geometries. Many very complex geometries are used to demonstrate the ability of new grid generation approach. It is confirmed that arbitrary 'dirty' geometries can be handled automatically.

ACKNOWLEDGEMENTS

The research was supported by the U.S. Navy under Contract N68335-98-C-0233 with Darren Grove being the Technical Monitor. The first author would like to thank Ashok Singhal of CFD Research Corporation for giving him the permission to use CFDRC software including CFD-Viscart.

REFERENCES

1. Thompson JF, Warsi ZUA, Mastin CW. Boundary-fitted coordinate systems for numerical solution of partial differential equations—a review. *Journal of Computational Physics* 1982; **47**:1–108.
2. Eriksson LE. Generation of boundary conforming grids around wing-body configurations using transfinite interpolation. *AIAA Journal* 1982; **20**:1313–1320.
3. Eiseman P. Grid generation for fluid mechanics computations. *Annual Review of Fluid Mechanics* 1985; **17**:487–522.
4. Chan WM, Steger JL. Enhancements of a three-dimensional hyperbolic grid generation scheme. *Applied Mathematics and Computation* 1992; **51**:181–205.
5. Shih TIP, Bailey RT, Nguyen HL, Roelk RJ. Algebraic grid generation for complex geometries. *International Journal for Numerical Methods in Fluids* 1991; **13**:1–31.
6. Lo SH. A new mesh generation scheme for arbitrary planar domains. *International Journal for Numerical Methods in Engineering* 1985; **21**:1403–1426.
7. Peraire J, Vahdati M, Morgan K, Zienkiewicz OC. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics* 1987; **72**:449–466.
8. Lohner R, Parikh P. Generation of three-dimensional unstructured grids by the advancing front method. *International Journal for Numerical Methods in Fluids* 1988; **8**:1135–1149.
9. Watson DF. Computing the *n*-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal* 1981; **24**:167–172.
10. Jameson A, Baker TJ, Weatherill NP. Calculation of inviscid transonic flow over a complete aircraft. *AIAA Paper 86-0103*, 1986.
11. Barth TJ. Steiner triangulation for isotropic and stretched elements. *AIAA Paper No. 95-0213*, 1995.
12. Venkatakrishnan V. A perspective on unstructured grid flow solvers. *AIAA Paper 95-0667*, January 1995.
13. Anderson WK. A grid generation and flow solution method for the Euler equations on unstructured grids. *Journal of Computational Physics* 1994; **110**:23–38.

14. Schneiders R. Automatic generation of hexahedral finite element meshes. *Proceedings of 4th International Meshing Roundtable'95*, Albuquerque, NM, October 1995; 130–114.
15. Nakahashi K. Adaptive prismatic grid method for external viscous flow computations. *Proceedings of 11th AIAA Computational Fluid Dynamics Conference*, Orlando, FL, July 1993; 195–203.
16. Coirier WJ, Jorgenson PCE. A mixed volume grid approach for the Euler and Navier–Stokes equations. *AIAA Paper 96-0762*, January 1996.
17. Kallinderis Y, Khawaja A, McMorris H. Hybrid prismatic/tetrahedral grid generation for complex geometries. *AIAA Journal* 1996; **34**:291–298.
18. Luo H, Sharov D, Baum JD, Lohner R. On the computation of compressible turbulent flows on unstructured grids. *AIAA Paper 2000-0927*, January 2000.
19. Berger MJ, LeVeque RJ. An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries. *AIAA-89-1930-CP*, June 1989.
20. DeZeeuw D, Powell K. An adaptively refined Cartesian mesh solver for the Euler equations. *AIAA-91-1542-CP*, 1991.
21. Coirier WJ, Powell KG. Solution-adaptive Cartesian cell approach for viscous and inviscid flows. *AIAA Journal* 1996; **34**:938–945.
22. Bayyuk SA, Powell KG, van Leer B. A simulation technique for 2D unsteady inviscid flows around arbitrarily moving and deforming bodies of arbitrarily geometry. *AIAA Paper 93-3391-CP*, 1993.
23. Aftosmis MJ, Berger MJ, Melton JE. Robust and efficient Cartesian mesh generation for component-based geometry. *AIAA Paper No. 97-0196*, 1997.
24. Karman SL. SPLITFLOW: a 3D unstructured Cartesian/prismatic grid CFD code for complete geometries. *AIAA-95-0343*, 1995.
25. Wang ZJ. A quadtree-based adaptive Cartesian/quad grid flow solver for Navier–Stokes equations. *Computers & Fluids* 1998; **27**:529–549.
26. Wang ZJ, Chen RF, Hariharan N, Przekwas AJ, Grove D. A 2^N tree based automated viscous Cartesian grid methodology for feature capturing. *Proceedings of 14th AIAA Computational Fluid Dynamics Conference*, Norfolk, VA, 1999; 447–457.
27. Wang ZJ, Chen RF. Anisotropic Cartesian grid method for viscous turbulent flow. *AIAA Paper 2000-0395*, January 2000.
28. Thompson JF, Warsi ZUA, Martin CW. *Numerical Grid Generation*: *Foundations and Applications*. North-Holland: New York, NY, 1985.
29. Bonet JA, Peraire J. An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems. *International Journal for Numerical Methods in Engineering* 1991; **31**:1–17.

Plate 1. A ‘Dirty’ non-water-tight automobile engine geometry.

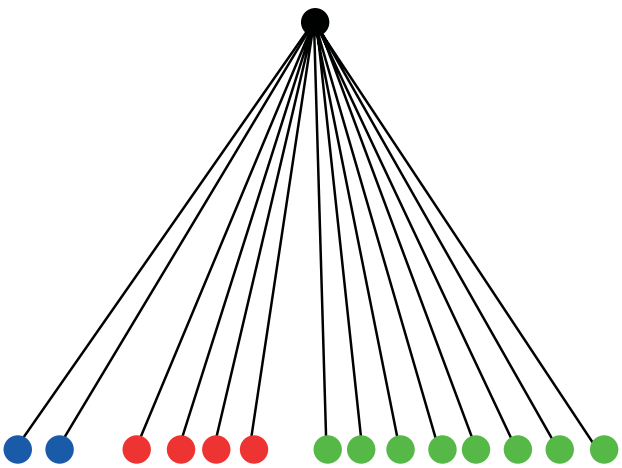


Plate 2. 2^n Tree data structure.

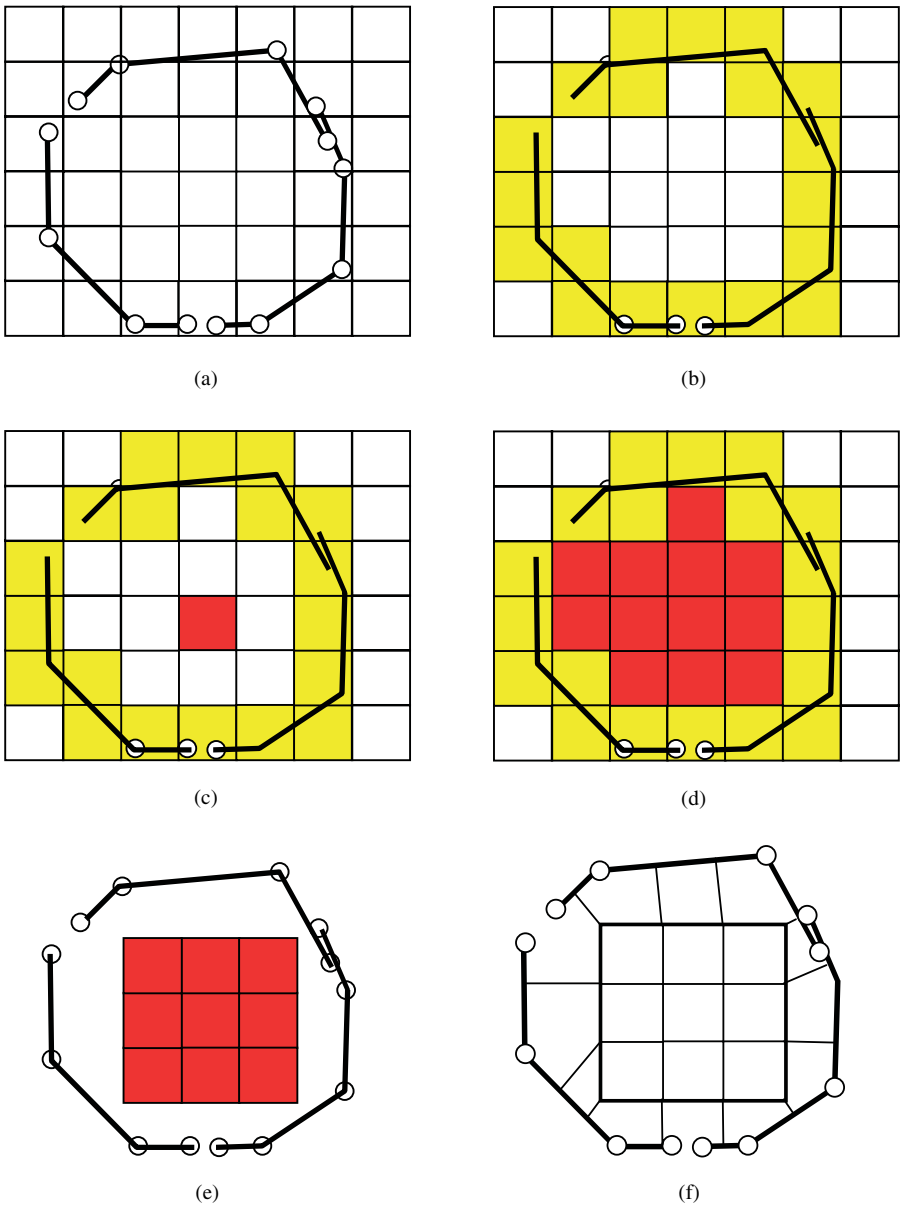


Plate 3. Schematic of I2B grid generation approach.

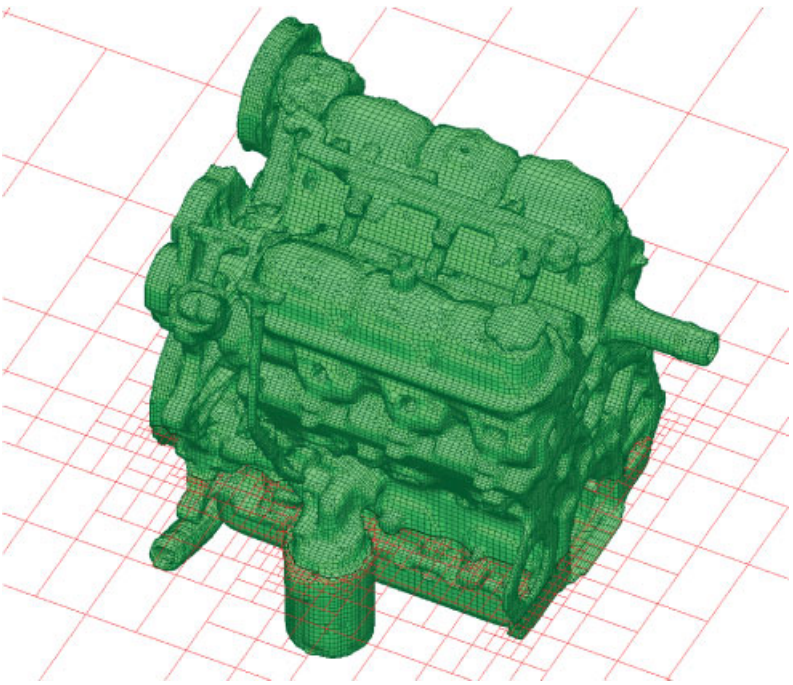
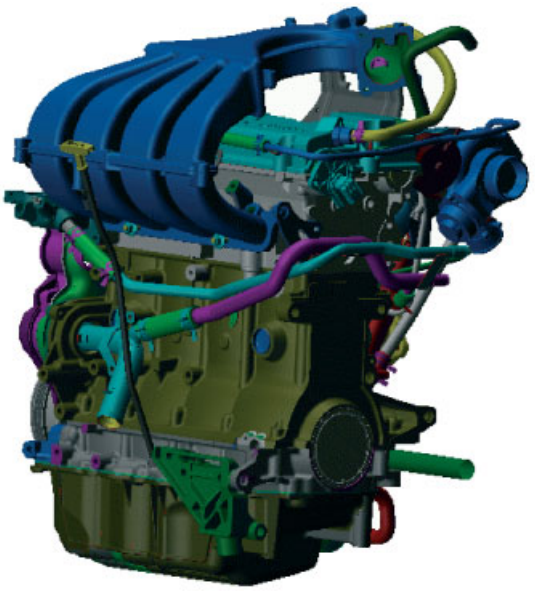
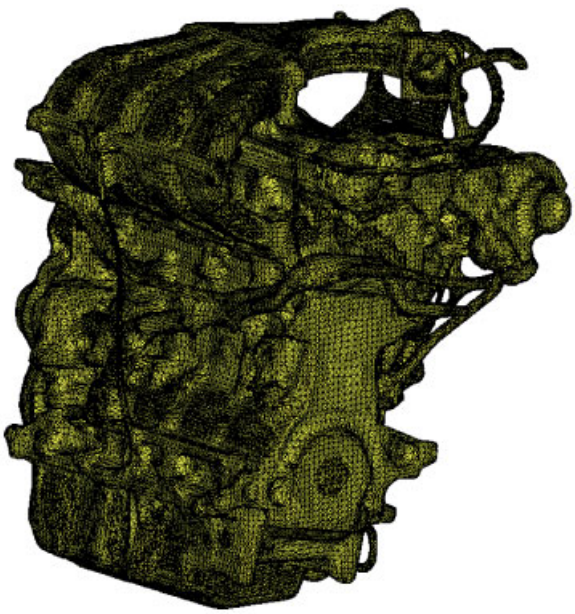


Plate 4. Adaptive Cartesian grid for the car engine shown in Plate 1.

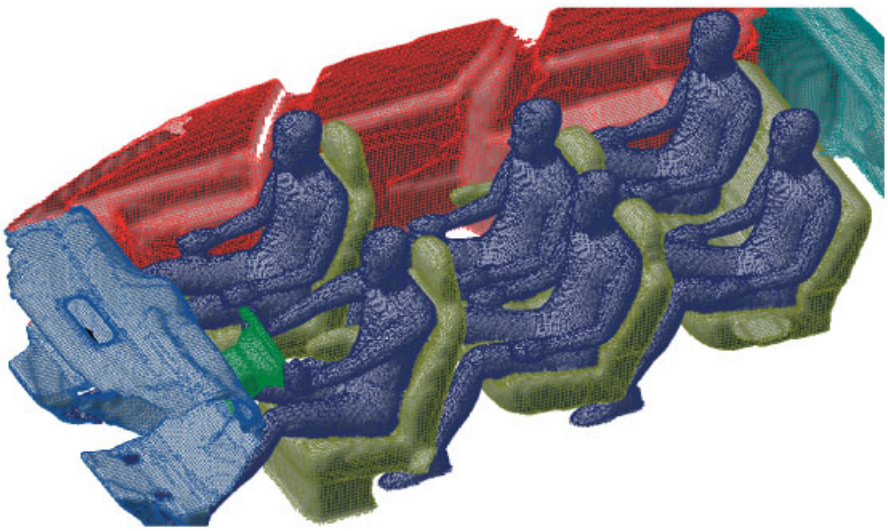


(a)

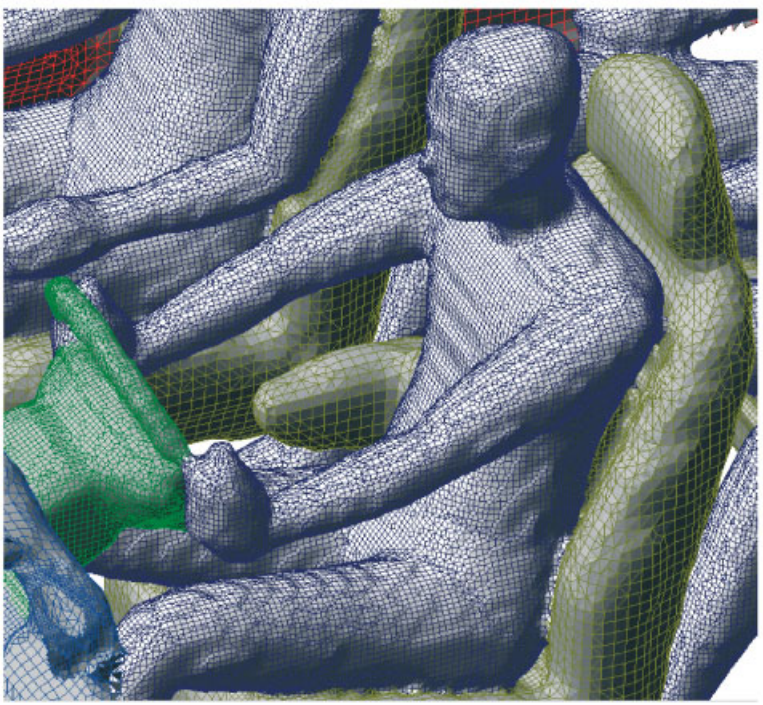


(b)

Plate 5. A more complex automobile engine and its surface grid.

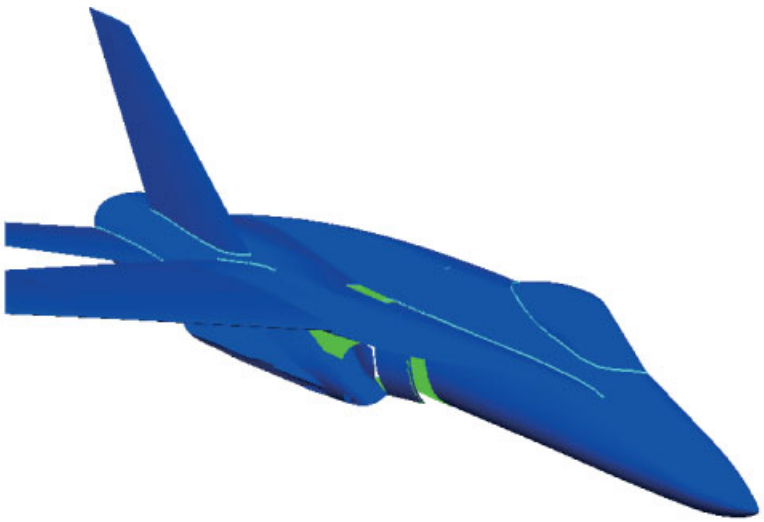


(a)

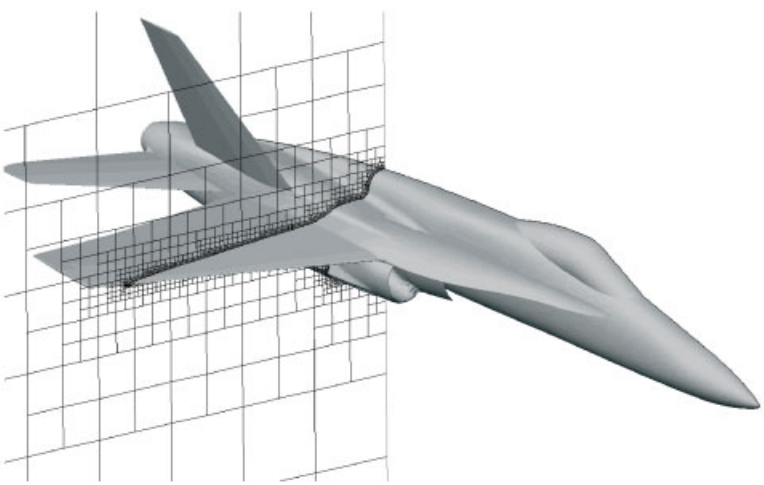


(b)

Plate 6. Passenger cabin mesh.

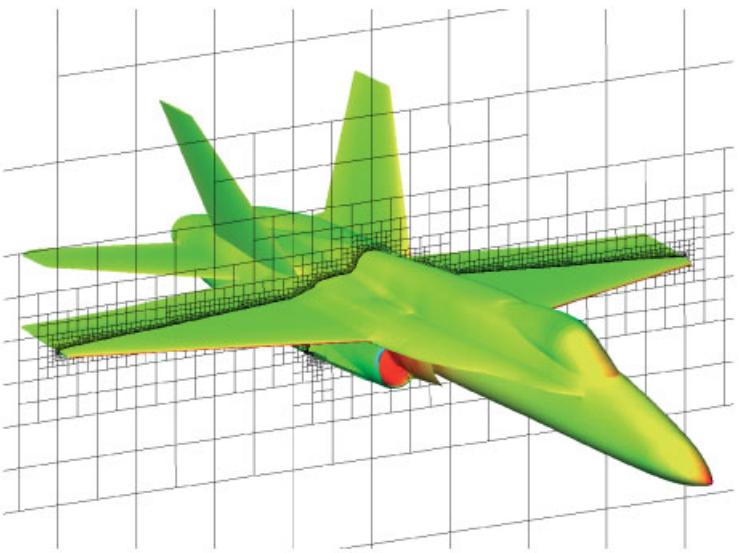


(a)



(b)

Plate 7. Geometry, computational grid and computed pressure distribution with Mach 0.3.



(c)

Plate 7. (*Continued*).