44

Flow Simulation using a Hybrid of Cartesian and Prismatic Grids

by
Paulus Lahur*1

ABSTRACT

This paper discusses the current development of a hybrid grid method suitable for compressible and viscous flow simulation. The method is part of computational software platform, UPACS (Unified Platform for Aerospace Computational Simulation), being developed in National Aerospace Laboratory (NAL). Its need arises due to the level of difficulty and the time required in generating grid of reasonable quality using currently available tools. A hybrid of prismatic and Cartesian grids is chosen as an extension of previous research in Cartesian grid for inviscid flow simulation. The main objective is to develop a robust, fast, and automatic grid generation tool, together with its flow solver. Grid generation module of each cell type (prismatic, Cartesian, cut cell) can be combined in such a way to suit various flow computation needs. Major considerations, problems encountered during the development and their possible solutions are discussed.

1. Introduction

Grid generation is a crucial part in computational fluid dynamics. It determines not only the accuracy of the solution, but also the overall efficiency and the cost of flow simulation process. As the simulation objective shifts toward more and more realistic body geometry, grid generation becomes a bottleneck. This is especially true for the case of structured grid, which is widely used by many commercial grid generation tools today. It is well known that grid generation process using this method can take weeks of manual labor.

It is clear that trying to build a 3D structured grid around an essentially unstructured, complicated body surface is the root of the problem. The success of unstructured grid is due to the fact that it takes exactly the opposite approach. Using elementary cell shape (tetrahedron in 3D), unstructured grid method has been applied successfully to complicated surface geometries. At present, this task can be carried out with a high degree of automation and under a very short time.[1,2]

Although less mature than unstructured grid method, Cartesian grid method is also capable to achieve similar performance, due to its "non-body-fitted" characteristic. The grid is constructed following the Cartesian coordinate system instead of the body surface, which enables very simple grid generation in most part of the computational domain. The cubic cell shape also contributes to more accurate flow solution when compared to tetrahedral cell. One of the main problems is treatment of cells intersecting the solid surface. Several approaches are available: (1) assume there is no cell, (2) cut the Cartesian cells,[3,4] or (3) deform the Cartesian cells to fit body surface.[5] The first method is usually employed in simulation of very complicated objects involving mostly separated flow, such as flow around a large number of buildings. For flows around smooth surface, the second and third methods become necessary.

Concerning the nature of the flow, the present demand in CFD has shifted toward simulation of viscous flows with high Reynolds number. The presence of very thin boundary layer requires the use of prismatic grid, in order for the simulation to be accurate and efficient. Robust prismatic grid generation of reasonable quality in the vicinity of a complicated body surface, however, is a difficult task, let alone filling the whole computational domain. This is where the concept of hybrid grid becomes indispensable, because it allows one to worry about generating prismatic grid only in the region very close to body surface. Prismatic grid can be used in hybrid with tetrahedral-based unstructured grid[1,2] or with Cartesian grid.[3,5]

At present, a development of 3D grid generator and flow solver for a hybrid of Cartesian and prismatic grids for viscous flow simulation is being carried out in National Aerospace Laboratory (NAL). The method is based on unstructured approach, applied to cells of arbitrary shape. The Cartesian grid method is an extension of the author's previous research.[4] The key development concepts are presented here.

*1 National Aerospace Laboratory (NAL)
Email: lahur@nal.go.jp

2. Grid Generation

It is assumed that the surface grid is already available, and given as a collection of interconnected triangles or rectangles or both, with the connection information between them.

Each type of grid is generated in separate modules, such that by matching these modules in certain order, it is possible to produce grids for different simulation purpose, as shown in Fig. 1. For example, flow simulation around a suburb comprising a large number of buildings can be handled most efficiently using Cartesian grid. Smooth body surface, on the other hand, requires proper attention regarding solid boundary interface, thus cut cells are generated to fill the room between Cartesian grid and body surface. When viscous flow is simulated, especially at high Reynolds number, prismatic grid is generated first around body surface, to provide proper boundary layer resolution. Each grid generation method will be discussed in the following.

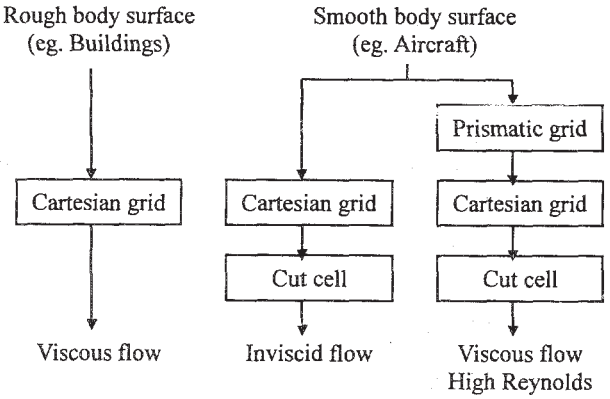


Fig. 1 Modes of grid generation

2.1. Prismatic Grid Generation

Prismatic grid generation around complicated body surface is still a challenge, and is a research subject by itself. The strategy of this study is to start from the simplest method, tackling the simplest class of geometry, and then proceed to the more complicated ones, in the following order:

(1) Mildly curved surface,
(2) Highly convex surface,
(3) Highly concave surface,
(4) Highly convex and concave surface,
(5) Surface with gap.

These types of geometry are illustrated using the rear section of a generic aircraft shown in Fig. 2.

In simulations where boundary layer effect is important, usually body surface is dominated by mildly curved geometry. Difficult regions are actually very limited, so it is tempting to generate prismatic grid using the simplest method first, and then apply some adjustment for the more complicated cases. Thus, a simple advancing layer method is chosen as the basic approach. The method makes a new surface outside the given surface. An

advancing vector at each node is constructed by taking a proper average of the normal vectors of the manifold, that is, the faces surrounding the node, as shown in Fig. 3.

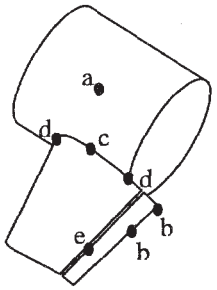


Fig. 2 Surface types: (a) mildly curved, (b) highly convex, (c) highly concave, (d) highly concave and convex, and (e) gap.

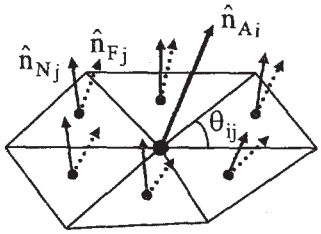


Fig. 3 A manifold.

Here $\hat{n}_{Nj}$ and $\hat{n}_{Fj}$ are defined as the normal vector and the vector of face j, respectively. $\hat{n}_{Ai}$ is the advancing vector at node i, whereas θ_{ij} is the included angle at node i and face j. Note the difference between face normal vector and face vector. The first is a geometric property and an unchanged quantity, whereas the second is an auxiliary variable, used for smoothing.

When a highly convex surface such as wing trailing edge is present, it is important to ensure smooth transition among neighboring advancing vectors to enhance grid quality in this region. Thus, an iterative smoothing involving the direction of the vectors is carried out, using equations (1) and (2). At the beginning of iteration, the face vector is taken to be equal to the face normal vector. The number of iteration normally used here is about 10 times.

$$\hat{n}_{A_i} = \frac{\sum \theta_{ij} \hat{n}_{F_j}}{\sum \theta_{ij}} \qquad (1)$$

$$\hat{n}_{F_j} = w\,\hat{n}_{N_j} + (1-w)\frac{\sum \theta_{ij} n_{A_i}}{\sum \theta_{ij}} \qquad (2)$$

where w is the weighting factor to control how far the face vector is allowed to deviate from its corresponding normal vector. Fig. 4a and 4b illustrates the effect of smoothing.

The magnitude of an advancing vector is determined in such a way that the next surface is more round that the preceding one, that is, small advancing distance for convex surface, and large for concave one.

As an example, a prismatic grid is generated around a dicretized surface of ONERA M6 wing with span of 1.2m. The surface grid consists of 14,968 triangles and 7,557 nodes. The surfaces of 10, 20, 30, and 40 layers of prismatic grid are shown in Fig. 7. It is evident that the surface becomes more rounded as more layers are generated. The thickness of the first layer is set at 0.1mm, and it increases by a factor of 1.2 for the subsequent layers. The resulting thickness of 30 layers is thus about 12cm. The total number of cells is 449,040. Close-up views of the grid in the leading and trailing edge regions at the wing's root and tip are shown in Fig. 8.

This point marks the present development status of our grid generator. Below is a brief discussion on key aspects concerning the coming development.

(1) Multiple advancing vectors at highly convex surface, for example wing's trailing edge, to increase grid resolution (see Fig. 4c). The concept of "split manifold" will be introduced, to assist construction of the advancing vectors.
(2) Algorithm to avoid grid crossing in the region of highly concave surface and gap.
(3) The use of Cartesian grid to substitute for prismatic grid in the regions where the geometry is especially difficult to handle.

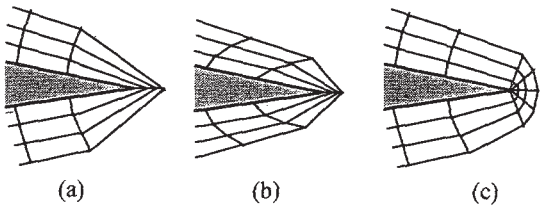


Fig. 4 Handling of highly concave surface: (a) basic, (b) with smoothing, and (c) with multiple advancing vectors (future development).

2.2. Cartesian Grid Generation

The Cartesian grid is generated around a base surface, which can be either a body surface or the outermost prismatic surface; both have the same format.

The grid generation starts with a single cell comprising the whole domain. The cell is then recursively refined until a certain level to increase the grid resolution around the base surface, using two criteria:

(1) A cell is within certain distance (in a certain direction) from the base surface. Here "box of refinement" is used to set more refinement downstream (Fig. 9 and Fig. 10).
(2) A cell contains a portion of surface that has a curvature beyond a preset limit. The curvature is indicated by the angular orientation between faces of the surface.

The grid refinement is isotropic, that is, a cell is refined in all axial direction, which results in eight sub-cells. It is thus very natural to use octree data structure here.

During the grid refinement, cells are divided into those that intersect the base surface and those that do not, that is, cut cell and non-cut cell, respectively. A linear check will take a very long time when the number of base surface's face elements is large. To increase efficiency, a cut cell stores the list of all face elements it intersects, and hands it down to all of its children. Thus the list of a given cell becomes smaller and smaller as the grid is further refined. When a cell is found to have no intersection with base surface, its list is emptied. It is noted here that the child of a cut cell may be a cut cell or a non-cut cell, whereas that of a non-cut cell is always a non-cut cell.

After the grid refinement process is completed, non-cut cells with no child are further classified into fluid or non-fluid cells. The resulting fluid cells are the Cartesian grid cells participating in flow computation. The classification is carried out using a "ray-casting" technique. A line is projected from the center of the cell to one of the corners of the computational domain. When the number of intersections between this line and the base surface is odd, then the cell is not a fluid cell. To improve efficiency, we reduce the number of ray-casting runs by propagating the status of a cell to its neighbors ("painting" algorithm).

Using the ONERA M6 wing example above, Cartesian grid is generated around the outermost prismatic layer (the 30 layer case is chosen here). Grid refinement is carried out 12 times, resulting in 161,113 cells (Fig. 11). About 100 Cartesian cells of the finest size (about 12mm) cover the span of the wing.

2.3. Cut-Cell Generation

The list of cut cells from the Cartesian grid generation process is taken as input. Each of the cut cells already contains the list of base surface's face elements that it intersects.

In the present method, a cut cell is constructed such that it represents the base surface as closely as possible, that is, no approximation of the surface is carried out. Thus, in general, the resulting shape of a cut cell is of arbitrary polyhedron (Fig. 5).

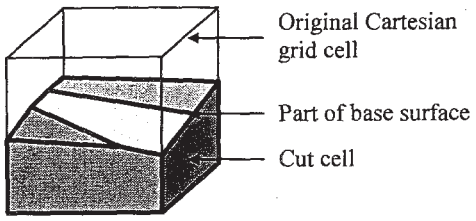


Fig. 5 A cut cell

Some issues that need special attention are:

(1) "Degeneration" in the cutting process, due to coinciding geometrical objects (face, edge, node).
(2) "Split cell," where the cutting process of one Cartesian cell results in two or more cut cells.
(3) Very small cells, which can be treated by merging the cell to a neighbor of bigger size to avoid very small time step.

These issues have been tackled to a certain degree in the previous research on Cartesian grid method.[4] Modifications are being introduced to improve the robustness and performance of the code. One of them is the use of integer (fixed-point) to represent coordinate. This change is motivated by the ambiguity during the determination of intersection when floating-point representation was used. The danger of using integer is the problem of overflow, which, unfortunately, is not reported by most computer operating system. To avoid this, when there is a possibility that such an error may occur, the number is first converted to a floating point before the actual operation takes place. Computation of intersection between an edge and a triangle is one of such examples. Its intermediate computation is open to the possibility of integer overflow, so conversion to floating-point number is carried out. The resulting intersection point is then converted back to integer.

Doubt may arise concerning the accuracy of integer compared to floating-point number. In most applications, this should not be a concern. For example, suppose that grid is generated around a 100m-long aircraft in a computational domain of 10km-long. If the whole domain is represented using integer from 0 to 2^{30}, then one unit of integer has the length of 9.3μm, which is certainly smaller than most manufacturing limitation. Note that the common length of an integer data is 4-byte, which is equivalent to a range of -2^{31} to $2^{31}-1$.

3. Flow Solver

The flow solver is currently based on finite volume method for Euler equations, with provisions for further extension to Navier-Stokes equations. It is capable of handling a cell of arbitrary shape, in an unstructured format. A cell is allowed to have any number of faces. The face, in turn, can have any number of vertices, which can be listed without any particular order. Thus, all types of cell described above can be treated by this flow solver as a general case. A major advantage of the unified approach is that the code can be streamlined, and no modification is necessary when other types of cells are to be used in the future.

The code takes advantage of the existing codes in UPACS, particularly the flow solver for structured grid, which serves as a reference.[6] Some of the codes in UPACS are also reused where possible, for example, the routines for parallelization.

At its present state, both explicit and implicit methods are available for time integration, whereas for flux computation, Roe and AUSMDV methods are used. To achieve second order accuracy, the solution at cell center is extrapolated to cell face. The flow solution gradient is computed using a least square method, which is particularly well suited for unstructured grid containing cells of irregular shape.[7]

4. Data Structure and Programming Aspects

As mentioned above, the grid contains cells of arbitrary shape, which is especially true due to the presence of cut cells. This necessitates that connectivity is explicitly stored, as shown in Fig. 6. As a result, more memory is required per cell, as compared to structured grid. In return, we have a high degree of flexibility in grid generation.

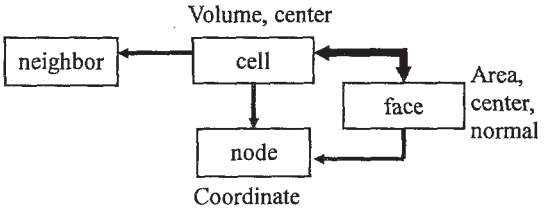


Fig. 6 Main data structure

In the process of grid generation, different data structures are used, depending on the cell type. In the case of Cartesian grid, octree data structure is used, because it naturally facilitates grid refinement. In the case of prismatic grid, a semi-structured approach is used, due to the advancing layer method. The grid is unstructured within a layer, following the body surface discretization, and structured in the advancing direction. This choice may have to be re-evaluated later for the case of more complicated body surface, where a simple advancing layer method may fail. At the end of grid generation, these data structure is converted to the unstructured one.

Arbitrary cell shape implies that memory allocation for a cell is not known in advance. Clearly, a programming language that supports dynamic memory allocation is needed for an efficient implementation. This capability ensures both code simplicity and low memory requirement, because allocation can be made just enough and just in time to match the demand. In this development, Fortran 90 is chosen, not only for the reason mentioned above, but also, and perhaps more importantly, to be able to reuse a large amount of existing code developed in-house.

5. Visualization

Grid and flow visualization are carried out using an existing tool called UPACS-GL, which is an implementation of OpenGL library in UPACS. The existing low level routines, such as that to draw line and polygon, can be used directly. High level routines, such as that to draw cell and grid, are being developed, because the existing ones are for structured grid.

6. Concluding Remarks

The present development status in hybrid grid method has been described. The grid consists of prismatic, Cartesian, and cut cells. Although yet to reach application level, the key concepts that will make the method unique can be identified as follows:

(1) Capability to generate different set of grids suitable for different types of geometry (smooth or highly irregular), and different types of flow (inviscid or viscous). This will allow user to employ a single tool to automatically generate grid for a specific need.
(2) The method accepts a grid cell as general polyhedron. This provides a unified treatment of cells of any shape, and thus allows the use of other types of cells in future development.

References

1) Kallinderis, Y., "Hybrid Grids and Their Applications," Handbook of Grid Generation, CRC Press, (1998), ch. 25.
2) Ito, Y. and Nakahashi, K., "Unstructured Hybrid Grid Generation based on Isotropic Tetrahedral Grids," AIAA-2002-0861, (2002).
3) Leatham, M., Stokes, S., Shaw, J.A., Cooper, J., Appa, J., and Blaylock, T.A., "Automatic Mesh Generation For Rapid-Response Navier-Stokes Calculations," AIAA-2000-2247, (2000).
4) Lahur, P. and Nakamura, Y., "Anisotropic Cartesian Grid Adaptation," AIAA-2000-2243, (2000).
5) Wang, Z.J., Chen, R.F., Hariharan, N., and Przekwas, A.J., "A 2^N Tree Based Automated Viscous Cartesian Grid Methodology for Feature Capturing," AIAA-99-3300, (1999).
6) Yamane, T., Yamamoto, K., Enomoto, S., Yamazaki, H., Takaki, R., and Iwamiya, T., "Development of A Common CFD Platform – UPACS –," Proc. Parallel CFD 2000 Conf., Elsevier Science, (2001), p. 257-264.
7) Aftosmis, M., Gaitonde, D., and Tavares, T.S., "Behavior of Linear Reconstruction Techniques on Unstructured Meshes," AIAA J., Vol. 3, No. 11, (1995), p. 2038-2049.

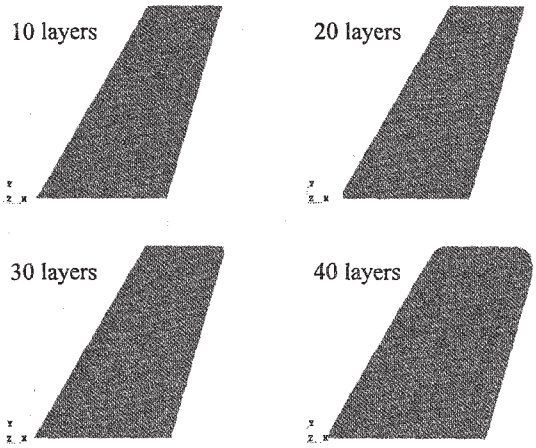


Fig. 7 Prismatic grid around ONERA M6 wing.

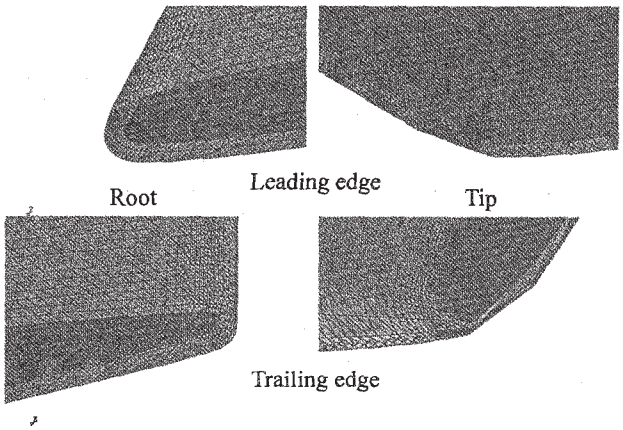


Fig. 8 Close-up view of prismatic grid.

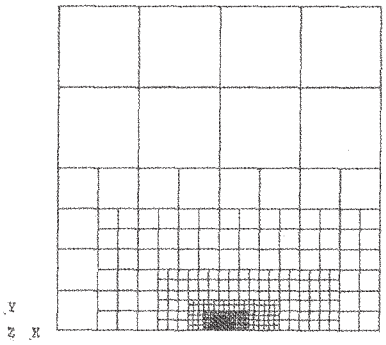


Fig. 9 The whole computational domain.

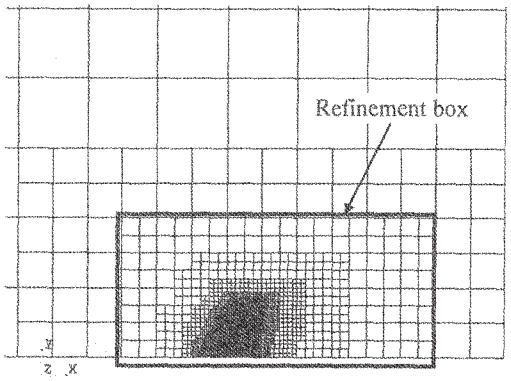
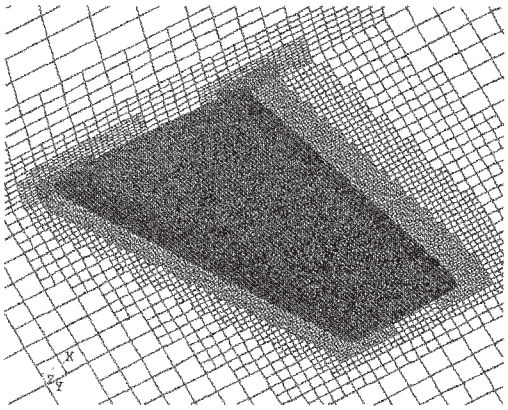


Fig. 10 Cartesian grid around ONERA M6 wing.



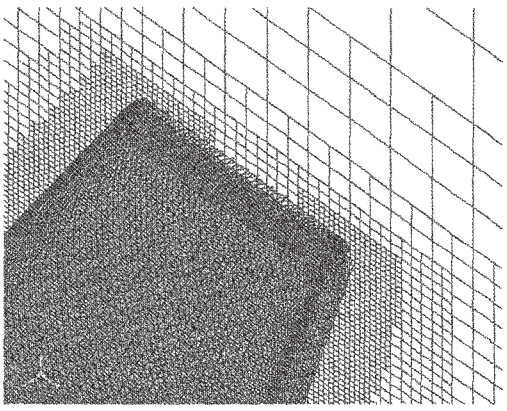


Fig. 11 The hybrid grid.