GEOMETRIC GRID GENERATION

David Ives
Pratt & Whitney
East Hartford, CT 06108

ABSTRACT

This paper presents a highly automated hexahedral grid generator based on extensive geometrical and solid modeling operations developed in response to a vision of a designer-driven **one day** turnaround CFD process which implies a designer-driven **one hour** grid generation process.

THE VISION

To affect the everyday aerodynamic design processes, a CFD system must be capable of generating the grid, setting the flow boundary conditions, solving for the flow, and completing the graphics and performance post processing in less than one 20 hour period, leaving at least 4 hours for the designer to conceive and define a new geometry to improve on the current design. This new design may have a different topology than previous cases. The flow solver unattended running occupies 16 hours of this cycle. The 24 hour design-analysis cycle seems to be natural human cycle, and allows a large number of design iterations. Each of the CFD system components must be sufficiently automated to allow the designer to operate them with minimal specialized training and support from CFD "experts". This one day turnaround is also described in Reference 1 which states: "In aerodynamic design work for commercial aircraft, CFD calculation results for aerodynamic analysis should be produced ... with short turnaround times (for large, complex calculations, **1 day**, say)". Likewise, Reference 2 states "The significance of this is that the total elapsed time for redesigning and arriving at a wing which meets all aerodynamic and manufacturability requirements can now be envisioned to approach the order of **one day**!"

These requirements break down to allowing an hour for high level automated grid generation, which is the focus of this paper. The high automation level reduces variation in grids generated by different users. With this constraint in mind, a grid generation system was developed which includes **only** technologies compatible with the **one hour** limit. This approach essentially starts from the desired market-driven goal and works backward to select the technologies used, rather than making continuous incremental improvements to current technology to work forward toward the one hour requirement. This is a high-risk, high-payoff approach. The initial demonstration described below is for Euler flows only; similar viscous flow grid technology is currently being developed. This approach can also be used with conventionally generated grids to quickly add geometric details.

PROCESS OVERVIEW

The geometric grid generation process used here typically starts with a single component body-conforming base grid from a conventional grid generator, as illustrated in Figure 1. A surface defining additional geometry to be modeled, shown in Figure 2, is then embedded in the base grid. The first embedding step is to close the surface as in Figure 3. Each base grid cell is then examined to determine if it is inside or outside the closed surface; if outside, it is retained and if inside it is marked as solid (inactive in the CFD flow solver). The process at this stage is illustrated in Figure 4 where the embedded surface is represented as a stairstep. This stairstep surface is then distorted by moving all nodes to the surface of Figure 2. A smoothing is coupled with this process; the resulting hexahedral grid in Figure 5 conforms to the body. The following examines each of these steps in detail.

INPUT PROCESSING

The geometric grid generation process assumes that a valid boundary representation (B-rep) solid model which completely specifies the space to be gridded has been produced by the CAD system used by the designer.The B-rep solid model surface is received from the CAD system as a large number of small triangles. The most popular format for CAD output in this form is a sterolithography file, which is a simple universal standard (in practice) currently supported by most major CAD systems. This allows us to use our current CAD system to process IGES input. The NASA IGES and NGP translators of References 3 and 4 may also be used to provide a separate IGES interface.

In practice, the quality of these solid models varies with the quality of the CAD system, and some clean-up operations are currently necessary. As CAD systems mature, the need for these separate clean-up operations will decrease, being transferred to the CAD system itself. The current grid generation system supports the following high level clean-up and manipulation operations on the input triangle surface primarily in a UNIX command line controlled mode:

1) interactively select surfaces/patches of object to be gridded
2) cap open surfaces under interactive user control:
   a) cap single open or closed curve
   b) cap region between 2 closed curves
3) remóve small details deemed unimportant by using capabilities 1 & 2
4) topological analysis of input surfaces to verify B-rep solid model integrity
5) merge points differing within a tolerance
6) reflect surfaces about symmetry planes
7) extend open surfaces to resolve intersection tolerances
8) Boolean trim, union,. etc. of objects - i.e. trim a pylon surface with wing and nacelle surfaces, then union them to obtain a single wing/pylon/nacelle surface

Please note that most of the above operations are very high level, usually requiring at most a single number to describe the operation, and thus are amenable to command-line control. The exceptions are options 1-3 which require a visualization and selection of the different surfaces and patches. The SUN "leotool" visualization system is used in a separate synchronous UNIX process to select the surfaces and patches. All that is required is to write a visualization file in the Wavefront ".obj" format which separates the surface/patch components into groups. These groups can be toggled on and off in leotool, allowing each group to be identified for control of the surface/patch selection process.

Another way to receive a B-rep solid model is to read a VSAERO panel method input file of Reference 5, which can be produced by other means such as the ICEM system of Reference 6. Such surface models tend to be of high quality and require only very modest or no clean-up to achieve a perfect solid model surface specification.

The basic philosophy is to let the CAD system or other tools which the designer uses on a daily basis specify the boundaries of the space to be gridded. This approach reduces training and overall complexity as the designer does not need to learn two different CAD systems, one for everyday work and the other for CFD only. It is noted that the resulting grids are patch-independent; they depend only on the B-rep surface as defined by the triangulation used to create the sterolithography file and not on the B-rep patch edge locations unless the user specifies specific patch edges to retain in the final grid.

Base grid. - The flow region is determined by **embedding** the input surface in a base grid, which can

range from a simple uniform Cartesian grid composed of a rectangular collection of cubes to a grid generated by a current interactive grid generator such as ICEM, NGP, or others.The embedding has three steps: automatic blocking, surface snapping, and quality improvement. These steps are discussed below. For purposes of this discussion a "solid" point, edge, face, or cell lies in a region in which there is no flow, and a "fluid" point, edge, face, or cell lies in a region where there is flow. A "computational space smoothing" refers to moving a point to the average position of all points linked to the point. A "physical space smoothing" refers to moving a point to satisfy a Laplace equation on a stencil involving all points linked to the point and requires knowledge of which finite difference stencil points are opposite each other as described in Reference 7.

AUTOMATIC BLOCKING

The geometric grid generator automatically determines the required blocking in a batch workstation run on using UNIX command-line options to select the desired blocking rules. The blocking is done on a cell-by-cell basis, which is simple for a computer, rather than blocking on large rectangular collections of cells as is done in interactive grid generators. The normal blocking rule used here is a simple local geometric test based on inside/outside tests of the 8 vertices of each cell and on intersection tests for 12 edges of each cell. This blocking rule is:

1) if 5 or more vertices are inside the surface, then the cell is solid
2) if 4 vertices are inside the surface and the average of the 8 vertices is inside the surface, then the cell is solid
3) if 4 or more edges intersect the surface ,then the cell is solid
4) otherwise the cell is fluid

The intersection tests in the blocking rule allow the correct detection of thin surfaces such as the trailing edge region of wings. A more sophisticated blocking rule could count intersections along each edge. If there are two intersections and both end points of the segment are in a fluid region then the edge is solid, while if both end points of the segment are in a solid region then the edge is fluid. These edge states could be included in determining if the cell is solid or fluid. This would allow the detection of cracks as well as thin trailing edges.

The inside/outside tests are based on ray tracing as described in Reference 8. If a ray from the point being tested to positive infinity in the Z-direction intersects the surface an odd number of times then the point is inside, otherwise it is outside. In practice, rays are sent in two opposite directions and the code checks to verify that both rays agree on the containment. If they do not agree, then a new direction is chosen and the process is repeated. For difficult cases (ie: cases where tolerances are nearly the same size as roundoff error) an option is included to require consensus among six orthogonal rays issued from the point to infinity. Logic is included to ensure that a ray hitting a triangle edge is given half an intersection count for each triangle independent of roundoff by the use of special symmetric coding. Of course, each edge is shared by exactly two triangles since we currently require two-manifold surfaces. A new direction is chosen if a ray hits a triangle vertex, although we could have included logic to give an intersection count of 1/N, where N is the number of edges sharing the vertex. The technology described here is equally applicable to Chimera grid generation, having much in common with the overlapping stencil generation described in Reference 9.

Handling objects which extend outside base grid. - There are cases in which the object to be gridded extends outside the base grid in some areas, and is inside or on the base grid surface elsewhere. For each stairstep cube with a quad face lying on an external block boundary (ie: the quad occurs only once), which we denote as a "candidate quad", we must decide if the face is to be retained in the

stairstep surface quad list. This is controlled by an overall control which instructs the system to either retain all of the candidate quads, or remove all of them. A local control, applied on selected base grid block faces, instructs the system to do the opposite of the overall control. In practice, the overall control is usually set to reject all candidate faces and the local control is applied to retain quads in a very few block faces.

Computational speed. - The key issue here is computational speed in determining the containment of typically one million points within, and the intersection of three million line segments with, a surface that may be composed of up to 100,000 triangles. Fortunately, the extensive literature of computer graphics, data structures, and searching contains efficient ray casting and searching algorithms (References 8, 10-13) which require only a few minutes of elapsed time on a workstation to perform these operations even for a curvilinear base grid. It is surprising that these speeds can be achieved, given the very large operation counts (3,000,000 times 100,000) that a straightforward approach would require. Techniques such as those in Reference 14 can often substantially reduce the number of triangles required to define a surface to a specified tolerance by selectively removing triangles in regions of low curvature, substituting a smaller number of large triangles for a large number of small triangles. This can help in the initial code development phase by making acceleration techniques less crucial at first. The methods used in the current work are based on sorting, classifying, and range box tests to preselect candidates for geometrical operations. It turns out that the time and storage required for these processes is only weakly (logarithmically) dependent on the number of triangles defining the input surface, so we can easily cope with 100,000 input surface triangles. Thus we have not yet added the capability to reduce the number of input surface triangles which became available after the current grid code development had progressed beyond the initial stages. However, it seems that this technique would be well suited to help clean up input surfaces which often have poorly distributed (both in size and in aspect ratio) triangles defining the surface.

SURFACE SNAPPING

The next step is to specify the "goal surface" which is the portion of the input surface which lies inside the surface of the base grid. This is achieved by using the base grid outer surface, expressed as a collection of triangles, to clip (in a Boolean sense) the input surface, which is also expressed as a collection of triangles. This operation is necessary only if the input surface extends beyond the base grid surface, otherwise the goal surface and the input surface are identical. The surface shown in Figure 3 extends outside the base grid surface (ie: it extends inside the fan cowl.) The base grid outer surface is obtained by loading each of six faces of each base grid cell into an array, then removing those faces which occur twice in the array to leave only those (surface) faces occurring once in the array.

Association. - A geometric and topological analysis is then performed on the goal surface to identify corners, apexes, paths, circuits, and boundaries. A sketch describing these features is included in Figure 6. A corner is a point in the goal surface which must be represented as a point in the snapped surface. Sharp edges in the snapped surface must lie along edge paths in the goal surface. A sharp edge is formed when two adjacent input triangles meet at a shared edge with less than a user-specified included angle. A path is a collection of sharp edges. Paths may either stop on the surface (called a pendant path) or terminate at a corner. A corner is defined as the meeting of three or more sharp edges or the isolated end of a pendant path. An apex is the tip of a conical surface. A boundary is an intersection curve (if any) of the input surface and the base grid surface. A circuit is a collection of paths surrounding a surface. To retain geometric and topological fidelity, corners and apexes in the input surface must correspond directly to points in the generated grid. Likewise, paths and boundaries in the goal surface must correspond directly to interior edges and boundary edges in the

generated grid. For geometrical fidelity, it is undesirable to allow a sharp edge to be represented by the diagonal (rather than the edge) of one of the cells. The process of associating stairstep surface vertices with the matching input surface corners, apexes, paths, boundaries, and the surfaces enclosed by the circuits is called "association". The most difficult task in the current implementation of geometric grid generation is the automatic association of paths on the quad surface with the sharp edge paths on the input triangular surface. For base grids much finer than the path separations, this task is straightforward and reliable. It becomes difficult when the distance between paths is small compared to the base grid resolution.

Currently, the association is done as a three step process. In the first step, a "bead" is swept along the triangle path, and the quad surface points closest to the bead are collected. In the second step, adjacent duplicate quad points are culled and a connected path is built by linking points which are adjacent in the quad faces. Then the remaining gaps are connected up to a distance of three links (six links near the path ends). Then remaining gaps are connected across quad diagonals if necessary. All of this is done under the constraint that no existing quad path be crossed. In the third step,the quad path is improved by removing self-canceling segments and special handling is applied to the path ends where multiple paths may meet.A refined version of this method replaces the bead point by the closest intersection of the triangle surface normal at the bead point with the quad surface. This uses additional information (ie: the normal vector) to distinguish between possible quad paths which lie close to each other. This logic is essential at a wing/fuselage intersection.

The point associations are established first, with interior and boundary points on the goal surface (corners or apexes) being associated, respectively, with interior and boundary points on the stairstep surface. The association criteria is based on the shortest distance between points in the stairstep and goal surfaces which are eligible to associate. As soon as a point is associated, it is flagged as ineligible for further association. Then boundary edge curves in the input surface are associated with collections of edges in the stairstep surface by associating eligible stairstep vertices with the input paths. Interior edges are then associated between the two surfaces in a similar manner. Finally the areas contained within closed circuits established by collections of the goal surface sharp edge curves (which are in turn collections of edges which are in turn linked collections of points) are associated with vertices between the corresponding circuits in the stairstep surface. The result is a list of corresponding points, a list of corresponding curves matched with collections of stairstep vertices, and a list of corresponding areas associated with stairstep vertices. Corresponding paths for the surfaces of Figures 2 and 4 are illustrated in Figure 7.The vertices in the stairstep grid surface are then moved to their associated points by a process we call "snapping" (in analogy to the CAD meaning of snapping), or the the closest point on their corresponding curve, or to the closest point on their corresponding surface. The HEXAR grid generator from Cray Research described in Reference 15 also operates in this manner. This move-to-the-nearest-point operation is also referred to this as "conforming", "projecting" or "displacement"(References 15-17).

The snapping process can result in distorted elements, so the curve snapping process is repeated typically three times with a computational space smoothing inserted between each snapping step. Finally the surface snapping process is repeated typically 9 times, with a computational space smoothing between each snapping. All surface faces are quadrilaterals as can be noted from Figure 3., since all grid elements are hexahedra. A grid for a complex turbine blade geometry shown in Figure 8. demonatrates that the many corners, edges, and thin surfaces are faithfully captured even with a relatively coarse grid compared to the features. This case required under 20 minutes of elapsed time on a SUN Sparc 10 workstation and involved an 85x67x88 base grid containing 501,160 points and an input surface specified by 7,904 triangles. The final mesh contains 29,115 hexahedra and 14,364 surface quadrilaterals. Some of these elements must be broken up further for

use with current finite element structural analysis solvers as discussed later.

The above process was chosen to minimize the need for flow solver modifications. The only flow solver modifications needed for our Euler flow solver were to add a fluid/solid cell flagging capability and some logic to recognize the associated solid face boundaries. We retained our extensively calibrated strong conservation finite volume block-structured capabilities as is. Our block structured Navier-Stokes flow solver needed no modifications to use these grids to represent Euler flow (blockage) boundaries, as will be illustrated later.

Other choices could have been made at this stage to go from a stairstep grid to a grid conforming to the goal surface. Some of these choices are illustrated in Figure 9. One choice would have been to cut the stairstep cells with the goal surface, producing grid cells with other than eight hexahedral faces, as in Reference 18. This is a viable option for a solver that can handle tetrahedra or general polyhedra, and is only a minor change to the previous grid generation procedure.

Another choice would have been to cut the stairstep cells with the goal surface and feed the required areas and volumes directly to the flow solver as in References 19-20. In this technique the cells which intersect the surface (shown as dashed lines in Figure 9) are flagged to receive special treatment by the flow solver. Since most cells do not intersect the surface, this was shown in Reference 20 to incur only a minor increase in flow solver computing time. Since extensive Boolean geometric operations (clip, join, cut, union, diff) are already supported, this step would be easy to implement in the current grid generator but would require flow solver changes that are more extensive.

Yet another choice would have been to specify a stairstep design rule similar to the one discussed previously, but which defines all cells containing any solid point or intersecting any solid surface as solid. We could then "pave" or triangulate the stairstep to fill in the region out to the goal surface as in Reference 21. This introduces an unstructured grid layer at the surface and would have required a significant change in our production Euler flow solver.

Finally, we could use a combination of all the above. The extra freedom brought about could improve the grid quality near the surface and lead to "provable quality" grids as it has done for the tetrahedral mesh approach described in References 22-24 and used in the ICEM "TET mesher". Again, this would require substantial changes to our current production flow solvers.

All of these approaches start with a base grid (Cartesian, curvilinear, octree embedded, ...), determine containment of each cell, then perform operations only on cells near the surface (not throughout the volume). We call these methods **"Geometric Grid Generation"** since they all involve extensive CAD-like geometric operations which can be made very fast. Chimera grids also fit this category.

The goal surface creation and snapping operations nominally require the intersection of up to 100,000 input surface triangles with up to 100,000 base grid surface triangles, and the snapping of up to 100,000 points to the closest point on a surface of up to 100,000 triangles. The operation count for these processes can exceed 20 billion operations without using suitable data structures and search algorithms. In practice, these computations can be accomplished in a few minutes on a workstation by using suitably efficient methods (References 10-13). The snapping process is also accelerated by sorting the points to be snapped by color, and snapping them to the associated triangles sorted by color, where the color denotes to the surface to which the point is to be snapped. Randomly reordering the triangles within each color accelerated the snapping further; as soon as a random triangle gives a close distance, most remaining triangles are eliminated with a fast range check. These procedures accelerated the calculations by a factor of 6 to 12.

QUALITY IMPROVEMENT

The grid quality at this stage is excellent everywhere except for the distorted cells near the surface, since the remaining cells are either cubes or base grid cells generated by a conventional grid generator. There often exists considerable distortion of grid cells near the surface, leading some to suspect that this process will never work in practice. This section discusses the steps taken to ensure the grid quality control which makes the process work for Euler flows.

There are two steps to improving the quality. The first is establishing the proper predictive grid measures for flow solver accuracy and stability. The second is smoothing the interior grid to enhance these measures. The measures of grid quality are based first on the particular flow solver requirements and second on some general geometric measures.

Our Euler flow solver requires a positive volume for **every cell** to run successfully. A predictive grid measure called "flatness" is presented in Appendix A for finite volume Euler flow solvers. The flatness of a cell is the height of a rectangular parallelepiped with a square base measuring one unit in width and depth and which has the same ratio of "cell volume divided by the cell face area raised to the 1.5 power" as the cell. Our Navier-Stokes flow solver, due to existing special handling of cells near solid surfaces, prefers to run with the grid as it exists at this stage with no or minimal interior smoothing. All it requires is a positive Jacobian for all cells not at the surface.

Other measures of grid quality are well known in the structures community, while less common in the fluid dynamics community. The basic measures supported here include the fundamental quad face measures (skewness, taper, warpage, convexity, and aspect ratio) and the hexahedral cell measures (edge angle, aspect ratio, twist angle) described in References 25-26 and currently used for finite element diagnostics in the PATRAN 3.0 structural analysis system.

The interior (volume) points which are within a specified integer linking amount (typically 2 or 3) of the snapped surface are also be smoothed using the technique described in Reference 7. This Laplace smoothing in physical space can be used because the block structured base grid structure is retained in the geometric grid. This physical space smoothing is less likely to fold, since it is based on the maximum principle, and also works better with high aspect ratio grid elements. The smoothing is coded in a manner which makes the block internal boundaries transparent to the smoothing process; the internal block interface boundaries "float" with the smoothing. The data synchronization process used in this smoothing is described in Appendix B. This smoothing may involve points which are on inter-block boundaries where we need to know which of the six linked points surrounding the point to be smoothed are opposite each other in the computational IJK space. A technique was developed to determine opposite points, even in arbitrary interface configurations, as required by the physical space smoothing formula. The "design rule" to determine which of the six linked points are opposite is based on the simple observation that opposite points in different blocks do not share a common face, while all other pairings of the six linked points share a common face. The "same-face" logic requires only one page of coding. Any points which are linked to more than six other points, such as at the centerline of a cylindrical grid, are not moved

In some situations, portions of the base grid surface which are outside the added input surface must be smoothed as well, while constraining them to lie on the base grid surface. Without this extra smoothing, grid folding may occur on original surfaces at the intersection of the snapped surface due to the differing requirements of the stair-step containment decision (which is a 3D criterion involving 8 points and 12 edge segments) and the original surface points taken individually. This behavior requires that we smooth the original surface near the snapped surface. This smoothing must smooth

the original surface points distribution while keeping them within the original surface. It must also keep the original surface points which lie on any original surface sharp edges on those sharp edges. It must also not move original surface corner points. Those original surface points within a specified integer linking amount (typically 2 or 3) of the intersection of the snapped surface with the original surface are so smoothed. For the surface smoothing, the smoothed point is moved to the average position of it's linked points and then it is snapped to a saved copy of the original surface such that the outward normals of the snapped and original surfaces lie within 90 degrees. This normal logic prevents "snapthrough" which may occur when the smoothing the base surface grid near the trailing edge of a thin nacelle cowl where the circumferential grid size is much larger than the cowl thickness. In this case the simple nearest surface logic moves the point through the thin trailing edge region to the wrong surface .Edge points are smoothed by moving them to the average of the two linked edge points, and then snapped to a copy of the original edges. This computational space smoothing works well on the original surface and edges. There is no need to separately treat each edge path and each surface as in the snapping module. All that is necessary is to separate the corner, edge, and surface points and snap to either the set of all edge paths or the set of all surfaces.An example of this extra smoothing can be noted on the top rear of the core cowl adjacent to the pylon in Figure 5.

The centroid/volume weighted Laplace smoothing described in Reference 27 was also tried in the hope that the closer the smoothing matches the flow solver, the better. The flow solvers used so far have all been finite volume flow solvers. This approach did not prove immune to occasionally producing cells with negative volumes. All of the above smoothing methods can produce negative volumes, particularly near concave surfaces. In that case, all of the support points and/or volume centriods can lie outside the flow region and any weighting (be it computational space weighting, physical space weighting, or centroid/volume weighting) can move a base point outside the fluid region.

The major operation counts for the smoothing step consist of up to 1,000,000 Laplace volume smoothing operations repeated typically 10 times and up to 100,000 surface snapping operations repeated typically 10 times. These operations can all be completed in a few minutes on a workstation.

One characteristic of the current geometric grid generation method is that the vast majority of grid cells are identical to the base grid cells, which are presumably of good quality. Only a few cells near the surface can have difficulties as described above. One solution to this is to treat only these relatively few cells individually along the lines described in Reference 15. In the pending implementation of this method, the cell having the lowest flatness value is chosen and improved by solving a optimization problem where the function to be maximized is the minimum flatness of the cell and all directly abutting cells. The variables are the interior node positions of the cell to be improved.

Even the smoothed distorted elements near the surface currently often exceed the bounds set by finite element structural solvers. Structural solvers may require a positive Jacobian at as many as 27 integration points. The geometric grid generation process can be modified by breaking up each significantly distorted element into a collection of less distorted (more convex) elements using published methods. One method, implemented here, breaks up distorted hexahedra into a collection of wedges, pyramids, and tetrahedra. The applicability of this method depends on whether or not the structural solver being used supports pyramids; some do not (at least easily). Pyramids are a crucial matching element between tetrahedra and hexahedra. Sometimes this breakup approach leads to the propagation of "pencils" of wedges all the way from a surface to an exit elsewhere on the surface. Another method in Reference 17 breaks up distorted hexahedra into a limited number of hexahedra all lying near the surface. The idea is to break up the distorted elements without adding a large number of new elements.

Flow solver data structures. - The NASTAR Navier-Stokes flow solver uses point "flag" values to control boundary conditions. It examines a group of point flags to deduce where solid cells are. The minimum solid cell configuration is a 2x2x2 cell; there must be at least two solid cells in each of the three coordinate (index) directions for a solid to be detected. The Ni Euler flow solver uses cell "flag" values to indicate solid cells, and uses cell information for the two cells straddling a face to determine the flow algorithm (flow or no flow) to use for the face. It also uses an "index table" to assign boundary conditions to each block face. This requires that all inflow and outflow cell faces lie on a block boundary, but allows solid regions as small as one cell thickness in each direction. Both of these techniques require the storage of flags approximately equal in number to the number of grid points. An alternate technique is to assign flags to face elements of the cells. Internal faces which share a common set of four points need only be represented once, so this scheme requires the storage of flags approximately equal to three times the number of grid points. However, this allows a thin solid (such as a jet engine mixer) to be represented as a membrane of zero thickness. It also allows the treatment of non-manifiold geometries and helps to keep track of inflow and outflow boundary conditions. The geometric grid generator can produce grids suitable for all three of these flow solver data structures.

DEMONSTRATION & VALIDATION

The first validation is the Euler transonic flow calculation in a 2-dimensional channel containing a circular bump known as the "Ni bump" case. The results from a conventional curvilinear grid and from the geometrically generated grid are presented in Figure 10. The geometric grid is distorted at the bump surface, yet it accurately predicts the transonic flow even on this relatively coarse grid at a near-choked condition.

The second validation is the Euler flow over a wing/pylon/nacelle where the pylon was geometrically added to an existing wing/nacelle grid as illustrated in Figure 11. The excellent agreement between the Euler calculations and NASA experimental measurements from Reference 28 at a transonic section of the nacelle very near the pylon/nacelle intersection is shown in Figure 12. Another case (not shown here) at a lower free stream Mach number shows excellent agreement with the VSAERO subsonic panel method solution from Reference 29 and NASA experimental data for this case where substantial wing/pylon effects exist.The author is grateful to Ron-Ho Ni who modified his Euler flow solver described in Reference 30 to accommodate the flagging of fluid and solid cells for these calculations, and to Bill Siddons, Bob Miller, and Ed Migliaro who provided the wing/nacelle base grid and the pylon geometry definition.

This technique was also used to predict the inviscid flow from the inlet of a helicopter, into a plenum, into a bellmouth, and finally into the inlet of a gas turbine engine. A fine grid is shown in Figure 13, and flow direction vectors from an earlier coarse grid (without a cylindrical pipe through the plenum which was added to the fine grid) are shown in Figure 14.

As a demonstration of use with a Navier-Stokes flow solver, the grid generator was applied to predict the flow blockage produced by inserting a probe in a compressor cascade blade row. The probe, defined by an IGES file, was read into the Unigraphics CAD program which produced a sterolithography file specifying the input surface. This probe was added in an Euler flow mode (slip flow boundary condition on the probe) to a viscous cascade base grid. The base grid surface and the snapped grid representation of the probe (as a shaded image) are illustrated in Figure 15. The author is indebted to Tom Rogers, Bob Zacharias, and Chae Rhie for providing the probe surface definition, the compressor passage base grid, and instruction in running the NASTAR Navier-Stokes flow solver described in Reference 31.

The final demonstration addresses structural grid generation. The example shown in Figure 16 is a MARC finite element solution for the VonMises stress in a loaded bracket assembly. The distorted elements have been broken up mainly into wedges. This example is included to demonstrate that structural grids can be successfully generated by this method, so that coupled fluid-structural solutions will be possible using geometric grids.

FUTURE DIRECTIONS

When this work was started, it was not known how good the flow solver predictions would be with these grids. The accuracy has exceeded our initial expectations and the extensive geometric operations required were successfully accelerated to reasonable times on a workstation.

It is anticipated that this technology can be extended to generate viscous grids by the process sketched in Figure 17 with less technical risk than was incurred in generating the Euler grids. The steps include generating a hyperbolic marching grid to a finite distance from the body as in References 32-34, and then constructing a geometric Euler grid from the outer hyperbolic surface. Then the geometric grid nodes on this surface are interpolated back to the body using the hyperbolic grid cells as an interpolation basis, finally the combined grid is smoothed. Many capabilities needed for this viscous extension are already coded for current grid generation capabilities. Examples include modules for computing surface normals and modules to determine the distance from a point to the nearest surface (including the surface identification number and the matching outward normal requirement). Combining these with the considerable literature and existing programs for hyperbolic marching grid generation is looked forward to with anticipation. It is noted that these viscous grids will contain extra directionally structured hexahedral collections of cells and will require either that our flow solvers be modified to match or that other flow solvers be used.

To reduce the current dependence on using simple conventionally generated base grids, it is necessary to extend the geometric grid generation process to include adaptive grid technology. It is expected that this will be straightforward due to the cell-by-cell approach used in the current implementation. Then the initial construction of a base grid may be replaced by a simple automatic coarse cubical grid enclosing the flow region, to be enhanced locally as needed.

It is noted that crucial enabling technologies allowed this work to be accomplished. These enabling technologies include fast workstations with large core and disk storage, and the attendant utilities such as make and dbxtool to manage and debug large coding projects. Another significant enabler is the ability to view complex surface images on the desktop, such as with the SUN ZX card and "leotool" graphics program. This project would have been unthinkable with the limited core shared-mainframe hardware, next-day graphics, and remote character terminals of ten years ago!

CONCLUSIONS

This paper demonstrates that highly automated geometric hexahedral grid generation for unique complex geometries can be accomplished in **one hour** on a workstation. The resulting grids yield demonstrated accurate Euler flow simulations as compared to experimental data, panel method calculations, and Euler flow simulations with standard curvilinear grids.

"**Geometrical Grid Generation**" is suggested as a descriptor for approaches that determine containment on a base grid and then operate only on cells near the surface to complete the gridding.

REFERENCES

1. Boerstoel, J.W., Spekreijse, S.P., and Vitagliano, P.L., "The Design of a System of Codes for Industrial Calculations of Flows around Aircraft and Other Complex Aircraft Configurations," 10th AIAA Applied Aerodynamics Conference, Paper # AIAA-92-2619-CP, Palo Alto, CA, June 1992.
2. Rubbert, P., "CFD and the Changing World of Airplane Design," AIAA Wright Brothers Lecture, Anaheim CA, Sept 1994.
3. Blake, M. W., Kerr, P. A. and Thorp, S. A., "NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA IGES)," NASA-RP-1338, April 1994.
4. Remotique, M. G. and the NGP Team, "The National Grid Project: Making Dreams into Reality," 4th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, pp. 429-439, Pineridge Press,Swansea, United Kingdom, April 1994.
5. Maskew, B., "Prediction of Subsonic Aerodynamic Characteristics: A Case for Low-Order Panel methods," Journal of Aircraft, Vol. 19, No. 2, pp.157-163, 1982.
6. Akdag, V., and Wulf, A., "Integrated geometry and Grid Generation System for Complex Configurations," Software Systems for Surface Modeling and Grid Generation, NASA CP-3143, pp. 161-171, Hampton, VA, April 1992.
7. Zacharias, R. M., Ives, D. C., and Siddons, W. D. Jr., "3D Grid Generation Tools," AIAA/SAE/ASME/ASEE 26th Joint Propulsion Conference, Orlando, Florida, July 1990.
8. Glassner, A. S., "An Introduction to Ray Tracing," Academic Press, Harcourt Brace Jovanovich, New York, New York, 1989.
9. LaBozzetta, W,F, et al, "MACGS - Towards the Complete Grid Generation System," AIAA-94-1923-CP.
10 Franklin, W. R., Chandresekhar, N., Kankanhalli, M., Akman, V., and Wu, P., "Efficient geometric Algorithms for CAD," Geometric Modeling for Product Engineering, pp. 485-498, Eslevier Science Publishers BV, North-Holland.
11. Samet, H., "The Design and Analysis of Spatial Data Structures," Addison-Wesley, Reading, Massachusetts, 1989.
12. Bonet, J. and Peraire, J., "An Alternating Digital Tree (ADT) Algorithm for 3D Geometric Searching and Intersection Problems," International Journal for Numerical Methods in Engineering, Vol. 31, pp. 1-17, 1991.
13. Sedgewick, R., "Algorithms," Addison-Wesley Publishing Company, Reading MA, 1983.
14. Hamann, B., "A Data Reduction Scheme for Triangulated Surfaces," Computer Aided Geometric Design, Vol. 11, pp.197-214, 1994.
15. Taghavi, R., "Fully Automatic Grid generation from CAD on Vector-Parallel and Massively Parallel Supercomputers," Societe des Ingenieurs de l'Automobile, SIA Conference, Massay-Palaiseau, France, February 1993.
16. Burantynski, E. K., "A Fully Automatic Three-Dimensional Mesh Generator for Complex Geometries," International Journal for Numerical Methods in Engineering, Volume 30, pp.931-952, 1990.
17. Schneiders, R., and Oberschelp, W., "Automatic Generation of Hexahedral Element Meshes for the Simulation of Metal Forming Processes," pp. 223-233, in "Numerical Grid Generation in Computational Fluid Dynamics and Related Fields," Wetherill, N. P., et al., Swansea, Wales, April 1994.
18. Wang, Z. J., "A Fully Conservative Structured/Unstructured Chimera Grid Scheme," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0671, Reno NV, Jan 1995.
19. Landsberg, A.M., Young, T,R.,Jr., Boris, J., "An Efficient, Parallel Method foe Solving Flows in Complex Three-Dimensional Geometries.", AIAA 32nd Aerospace Sciences Meeting & Exhibit, Paper # AIAA-94-0413, Reno, NV, January, 1994.
20. Melton, J. E., Berger, M. J., Aftosmis, M. J., and Wong, M. D., "3D Applications of a Cartesian Grid

Euler Method," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0853, Reno NV, Jan 1995.
21. Kao, K. H., and Liou, M. S., "Direct Replacement of Arbitrary Grid-Overlapping by Nonstructured Grid," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0346, Reno NV, Jan 1995.
22. Moore, D., "Simplicial Mesh Generation with Applications," Department of Computer Science, Cornell University, Ithaca, New York, December 1992.
23. Moore, D. and Warren, J., "Multidimensional Adaptive Mesh Generation," Rice COMP TR90-106, Department of Computer Science, Rice University, Houston, Texas, February 1990.
24. Moore, D. and Warren, J., "Adaptive mesh generation II: Packing Solids," Rice COMP TR90-139, Department of Computer Science, Rice University, Houston, Texas, March 1991.
25. Robinson, J. and Haggermacher, G., "Element Warning Diagnostics," Finite Element News, Volume 3, pp.30-33, June 1982; Volume 4, pp.19-23, August 1982.
26. Robinson, J., "Some new Distortion Measures for Quadrilaterals," Finite Elements in Analysis and Design, Volume 3, pp. 183-197, 1987.
27. Stephenson, M. B., Hsu, S. S., and Benzlet, S. E., "Automatic Mesh Generation for Free-Form Deformation Solids," COMPUTERS IN ENGINEERING 1990, Proceedings of the 1990 ASME International Computers in Engineering Conference and Exposition, Boston, Massachusetts, August 1990, pp. 277-286.
28. Carlson, J.R. and Compton, W.B., "An Experimental Investigation of Nacelle-Pylon Installation on an Unswept Wing at Subsonic and Transonic Speeds," NASA Technical Paper 2246, February 1984.
29. Lord, W. K. and Zysman, S. H., "VSAERO Analysis of a Wing/Pylon/Nacelle Configuration," AIAA/ASME/SAE/ASEE 22nd Joint Propulsion Conference, Paper# AIAA-86-1523, Huntsville, AL, June 1986.
30. Ni, R. H., "A Multiple-Grid Scheme for Solving the Euler Equations," AIAA Paper 81-1025, Palo Alto, California, June 1981 and AIAA Journal, Volume 20, No. 11, November 1982.
31. Rhie, C. M., "Pressure Based Navier-Stokes Solver Using the Multigrid Method," AIAA Journal, Volume 27, Number 8, pp. 1017-1018, August 1989.
32. Karman, S. L. Jr., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0343, Reno NV, Jan 1995.
33. Kallinderis, Y. and Ward, S., "Prismatic Grid Generation for 3-D Complex Geometries," AIAA Journal, Vol. 31, No. 10, pp.1850-1856, Oct 1993.
34. Chan, W. M. and Buning, P. G., "A Hyperbolic Surface Grid Generation Scheme and its Applications," AIAA Paper 94-2208, AIAA 25th Fluid Dynamics Conference, Colorado Springs, Colorado, 1994.
35. Press, W. H., Flannery, B. P., Teukolsky, S. A., Vettering, W.T., "Numerical Recipes", Cambridge University Press, New York, New York, page 146, 1986.
36. McConnell, S., "Code Complete," Microsoft press, Redmond WA, 1933.
37. Monoit, R, "**ftnchek**," available by anonymous ftp from research.att.com (192.20.225.2) in the directory /netlib/fortran or by electronic mail by sending the message "send ftnchek from fortran" to the internet address: netlib@ornl.gov

APPENDIX A - FLATNESS

This appendix defines a "flatness" measure to **predict** the suitability of a grid cell for use with finite volume solvers. Finite volume solvers distribute the flux entering all the faces to the volume contained within the cell, so a likely measure relates the volume to the total face area. Since volume has three dimensions, and area has two dimensions, we choose a form of relation involving volume/area$^{1.5}$ to produce a dimensionless measure. We also desire the measure to vary from -1 to +1 and to have a simple physical interpretation. Such a definition of cell "flatness" is:

$$VA=volume/(|area|/6)^{1.5}=flatness/((1+2*|flatness|)/3)^{1.5}$$

This flatness is physically interpreted as the height of a rectangular cube with a unit square base which has the same value of [volume/|area|$^{1.5}$] as the element. The rectangular cube has volume=flatness and area=(2+4|flatness|). The 6 in the formula ensures a flatness range of -1 to +1. It is our experience that as long as the minimum flatness exceeds .02, the Ni Euler flow solver of Reference 30 does not encounter grid difficulties. A flatness of -1 represents a cube whose faces are oriented counter-clockwise-in (negative volume). Face areas are positive by definition. A ratio of +1 corresponds to a cube whose faces are oriented counter-clockwise-out (positive volume). A flatness of zero corresponds to a hexahedron with no volume, for example a hexahedron with all its' vertices lying within a plane. The rate of change of VA with respect to flatness is zero at |flatness|=1, so the simple VA ratio does not provide good resolution for cells with |flatness|~1. The flatness equation can require the solution of a cubic equation. For |VA|<.707, we use Newton-Raphson iteration to solve for VA, otherwise we use the method described in Reference 35. The two-way implementation of the relation between VA and flatness is given below to encourage wider use of the flatness measure. This FORTRAN code is rapid, stable, and accurate to about +/- .000001.

```
      FUNCTION FLATNESS (VA)
      REAL*4    FLATNESS,VA,A1,A2,A3,Q,R,TH,S,ERR
      INTEGER*4 I
      IF(ABS(VA) .LE. .707) THEN
         FLATNESS=0.
         DO 1 I=1,8
         ERR=FLATNESS/(SQRT((1.+2.*ABS(FLATNESS))/3.))**3-VA
         IF(ABS(ERR).LT.1.E-7) GOTO 2
         S=SQRT((1.+2.*ABS(FLATNESS))/3.)
1        FLATNESS=FLATNESS-ERR*3./(1.-ABS(FLATNESS))*S**5
      ELSE
         A1=(1.500-3.375/VA**2)
         A2=.750
         A3=.125
         Q=SQRT((A1**2-3.*A2)/9.)
         R=(2.*A1**3-9.*A1*A2+27.*A3)/54.
         TH=(ACOS(R/Q**3))
         FLATNESS=-2.*Q*COS(TH/3.+3.14159265*4./3.)-A1/3.
      ENDIF
2     FLATNESS=SIGN(FLATNESS,VA)
      RETURN
      END

      FUNCTION VA (FLATNESS)
      REAL*4    VA,FLATNESS
      VA=FLATNESS/(SQRT((1.+2.*ABS(FLATNESS))/3.))**3
      RETURN
      END
```

APPENDIX B - GENERIC ISSUES

This appendix contains discussions of technical issues which involve all of the above processes.

Tolerances

In the geometric grid generation process, we first determine which points are coincident in the base grid, and keep track of them as a single unit. After this determination all canceling of cubes or quads takes place using only pointers, and not the coordinates themselves, to guarantee insensitivity to tolerances. We thus separate the geometry from the topology. This is a key issue for single precision manipulation of inviscid grids where the length scales can vary by five orders of magnitude, such as for a wing/pylon/nacelle/fuselage. Essentially, the coincidence of base grid points is checked initially with a zero tolerance and then all following operations on the base grid are performed with zero tolerance settings. This presumes that the input base grid is constructed that any coincident points are coincident, not just near each other. The ICEM grid generator of Reference 6, used here for many of our base grids, can produce non-coincident points, but it also produces an integer face connectivity table which can be used to enforce strict coincidence of face points. This then allows us to operate with zero tolerances and create single precision Euler quality grids for a fuselage/wing/nacelle/pylon installation flow simulation involving grid element sizes which vary by 5 orders of magnitude.

Data Synchronization

In handling multi-block grids, the a point may be shared between multiple blocks at common interfaces. The problem then arises of how to keep all of these different images of the same point identical. One way, commonly used in CAD systems, is to deal exclusively with pointers to a triad of coordinates and store the coordinates only once. This is efficient for CAD, which has relatively few such points, but can be inefficient for multiblock grids with often millions of points. For multiblock grids only relatively fewer grid points lie on block interfaces than the total number of grid points, so we only must handle these fewer points in a synchronous manner. The technique used here is simple to implement, and guarantees that all images of a point match. The method allows a "read" of any of the equivalent points, and when a "write" is needed all images of the point (kept in a separate list) are written. This guarantees that all images of the point are identical. This is somewhat simpler to implement than referring to all interface points by pointers or reference and all interior points by value. Thus all points are referred to by value for read operations without any special handling and only the write operation need be modified from a code which does not enforce synchronicity. For write we intercept the standard write operation and test if the point being written is an interface point. If it is not an interface point, it is simply written as before. If it is an interface point, all the pointers to the corresponding image points are used to write the identical value at the corresponding locations. In essence, we read by value and write by image references. This method can be described as "read any / write all". In essence, we use I,J,K,L computational space indices for points interior to the blocks, and the equivalent of pointers to block face points.

The block boundary image point determination is made by examining the supplied base grid for points having the exactly coincident coordinates. Pointers to those points which share the same location are saved in an "images" list which is constructed for rapid recall of all pointers equivalent to a given pointer. In practice, some base grid generators may not produce exact grid point coincidence at common boundaries. In this case we either may either merge block boundary points which are coincident within a tolerance or merge block boundary points specified by a block face matching table produced by the base grid generator. At least one base grid generator, Reference 4, guarantees by construction coincident points at common block interfaces.

Clipping

The surface to be clipped can be non-manifold and things still work perfectly, since only one clipped surface triangle is processed at a time. All that is required is that the clipping surface (which comes from skinning a usually well-behaved grid) be nice. The logic used always gets the edge intersections from low to high, so that two triangles sharing a common edge are treated identically. The logic also completely avoids the use of the square root operator. An approximation is used to simplify the process: namely the clipped surface edges are intersected with the clipping surface and the clipped surface edge intersections are connected. No account is taken of the variation (other than linear) of the clipping surface within the triangles to be clipped. No account is taken for the intersection of multiple parts of a surface within the triangles to be clipped. This method cannot be used for Boolean unions, as the unions will not join continuously. However, this method is often fully suitable for trimming the goal surface with the base grid skin. The code is fast, requiring only 21 seconds on a Sparc 10 for a clipping a surface of 29,564 triangles with a clipping surface defined by 29,552 triangles.

Cache Coherence

When the geometric grid generator works with a structured base grid, the information needed by a flow solver is still as adjacent in memory as it is for a structured grid generator. Then, with modern workstations having large amounts of (sometimes even two level) cache memory, it is believed that cache misses will be small and the present grids will show a negligible increase in CPU time requirements due to cache misses and paging. For adaptive grids it is anticipated that if we place the embedded grid storage for each coarse grid cell adjacent to the data for the coarse grid cell, we will likewise have efficient cache memory usage.

Visualization

The higher level of operations in this process, compared to current interactive CAD based approaches, requires that surface visualization be available to inspect the results. Without this visualization, it is very hard to discern where the process failed. This is particularly important when checking the input surface file and the base grid file, and when deciding which operations to perform. For example, if the input surface is not closed, it must be zippered as described above. If one forgets to check for openness (either with diagnostic code or with a quick visual look), then the automatic blocking process will fail badly. Visualization is crucial at times like this. The code currently is controlled by standard UNIX command lines with options enabled or set by command line arguments. A few command-line responses are required within the runs; these are being removed as feasible for the production version. A GUI shell can then be added later to further simplify use.

Coding Quality

The approach described here consists of a sequence of high-level operations, and it is imperative that each operation work **perfectly** or the overall results will be useless; all it takes is one bad cell out of a million cells to cause the flow calculation to diverge! This places stringent demands on the approximately 70,000 lines of FORTRAN code which implement this process. It is necessary to code very defensively, checking data and **assumptions** whenever possible. The major operations are currently separated by files (which one module produces and the next module reads) to decouple the operations and to simplify debugging.

The main type of bugs currently encountered are "concept bugs" which typically involve implicit assumptions overlooked in the initial formulation. Often an assumption is made that one has

imagined all combinations possible, and accounted for them, but in fact some have been missed. This often happens in topological circumstances. One example is the assumption that each edge of a triangle in one surface being cut by another surface can intersect the other surface in only one point. This assumption led to code which uses containment logic to slice the triangle, and if both end points of an edge are on the same side of the cutting surface concludes that the edge does not intersect the surface. This works most of the time, but fails on occasions when the assumption is violated. The fix is to use intersection logic, rather than containment logic, in this particular case. Concept bugs are generally harder to rectify, since they affect the basic formulation.

Coding memory allocation errors are prevented by an interface between the programs and the FORTRAN *malloc* and *free* calls as suggested in Reference36. Other coding errors tend to be removed by using different compilers and the ftnchek FORTRAN source code checker from Reference 37. The maximum optimization level that is used on the SUN is opt2, since opt3 produced erratic results for pointer variables resulting from dynamic memory allocation and for embedded "DO" loops containing "GOTO" statements pointing to a common "CONTINUE" statement. With these quality conformance tools, combined with defensive coding, the primary source of errors is concept bugs.

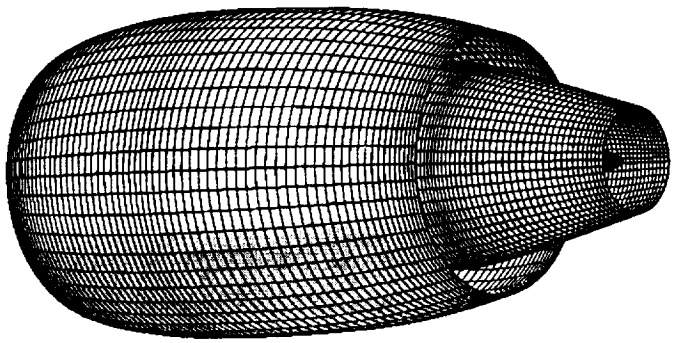

Figure 1. Base Grid Surface for a Nacelle

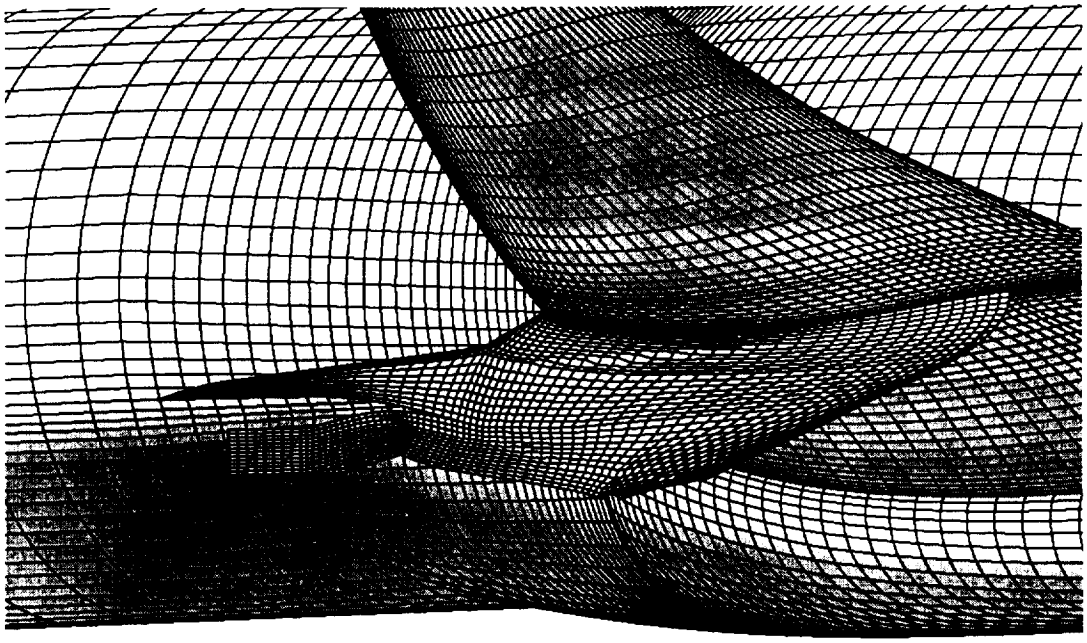

Figure 2. Fuselage/Wing/Pylon Surface from VSAERO File

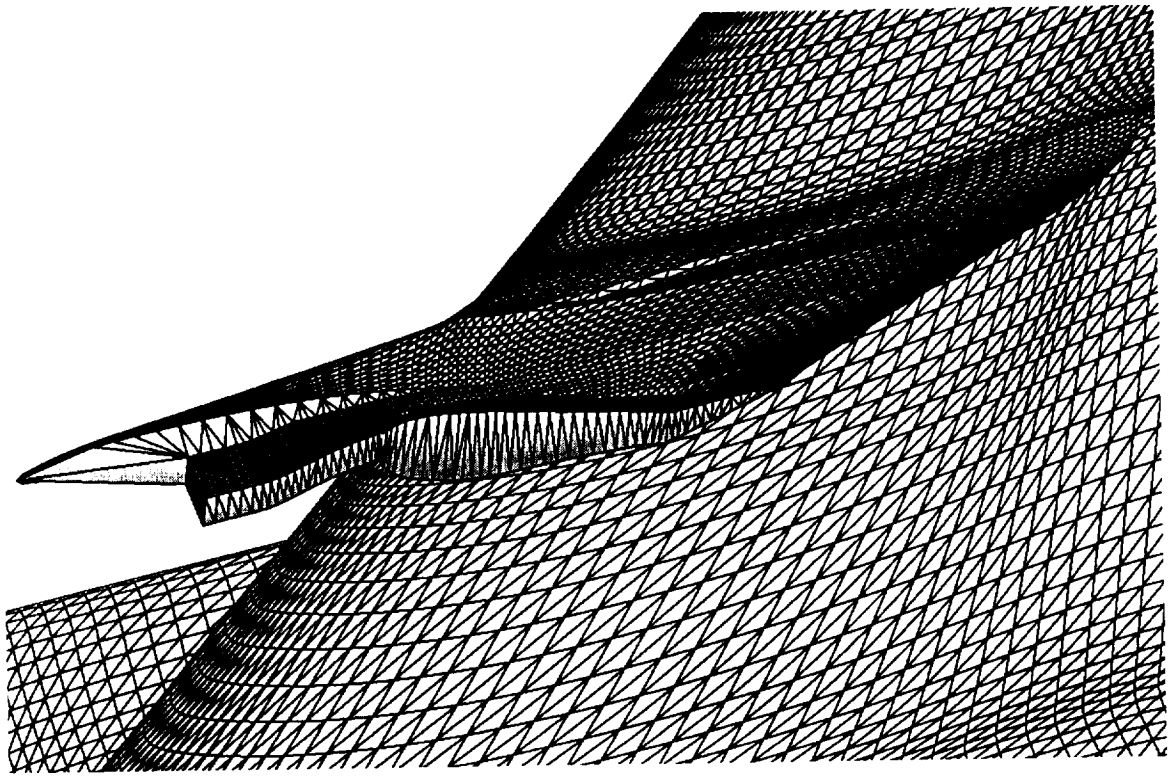

Figure 3. Closed Surface

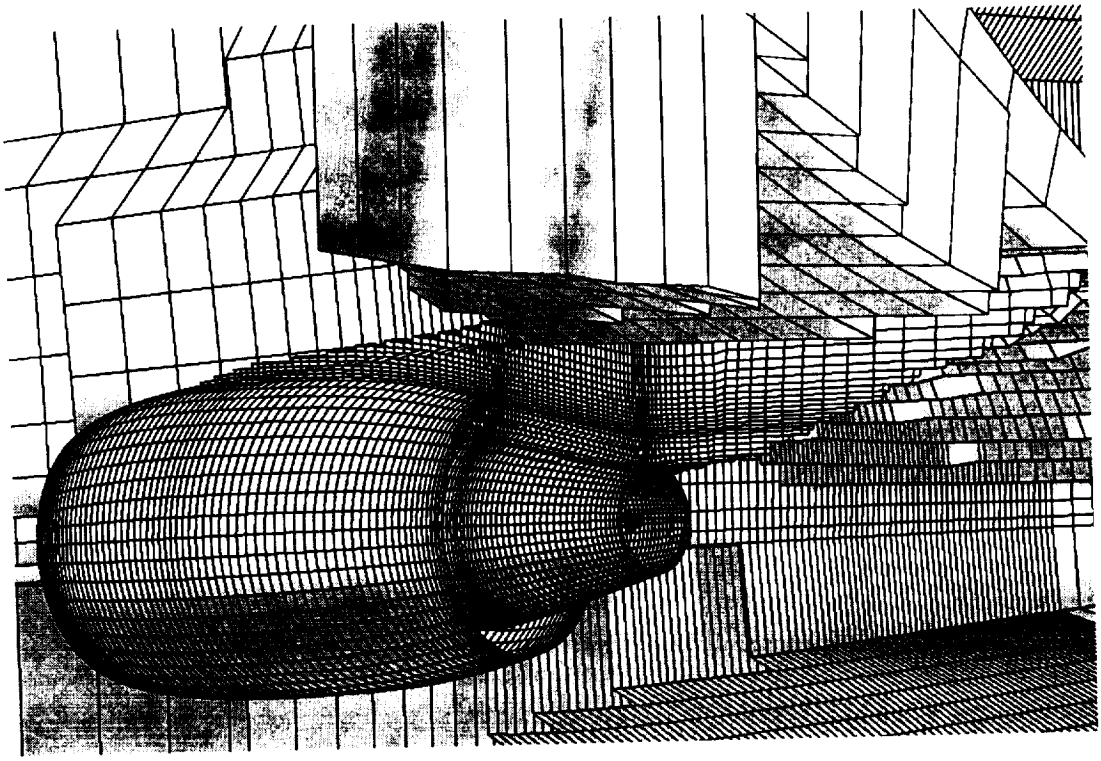

Figure 4. Stairstep Grid Surface

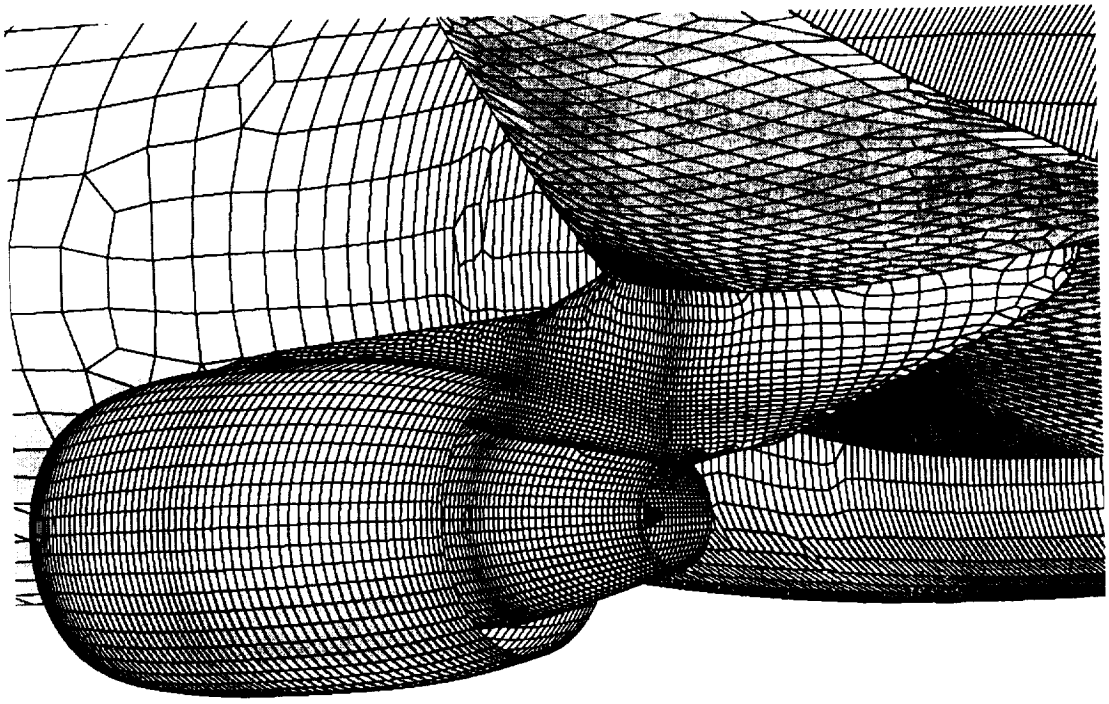

Figure 5. Snapped and Smoothed Grid Surface

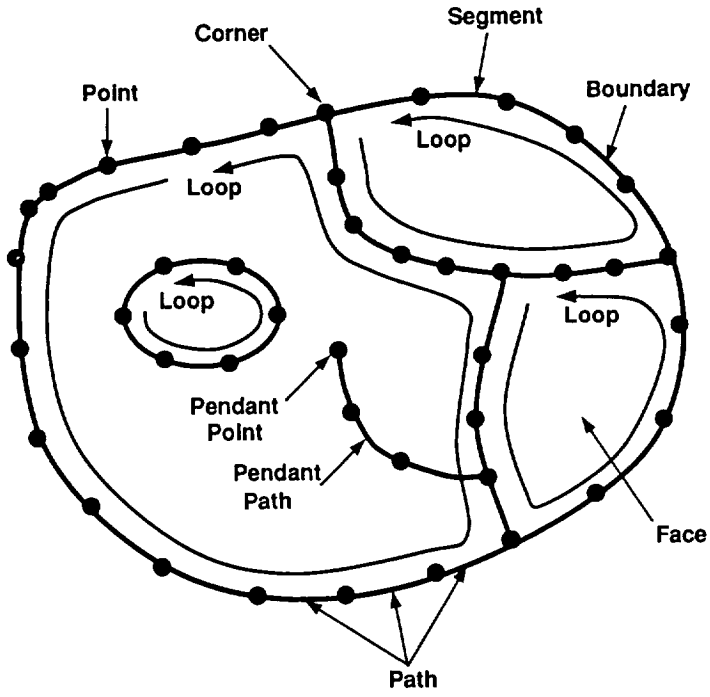

Figure 6. Grid Features

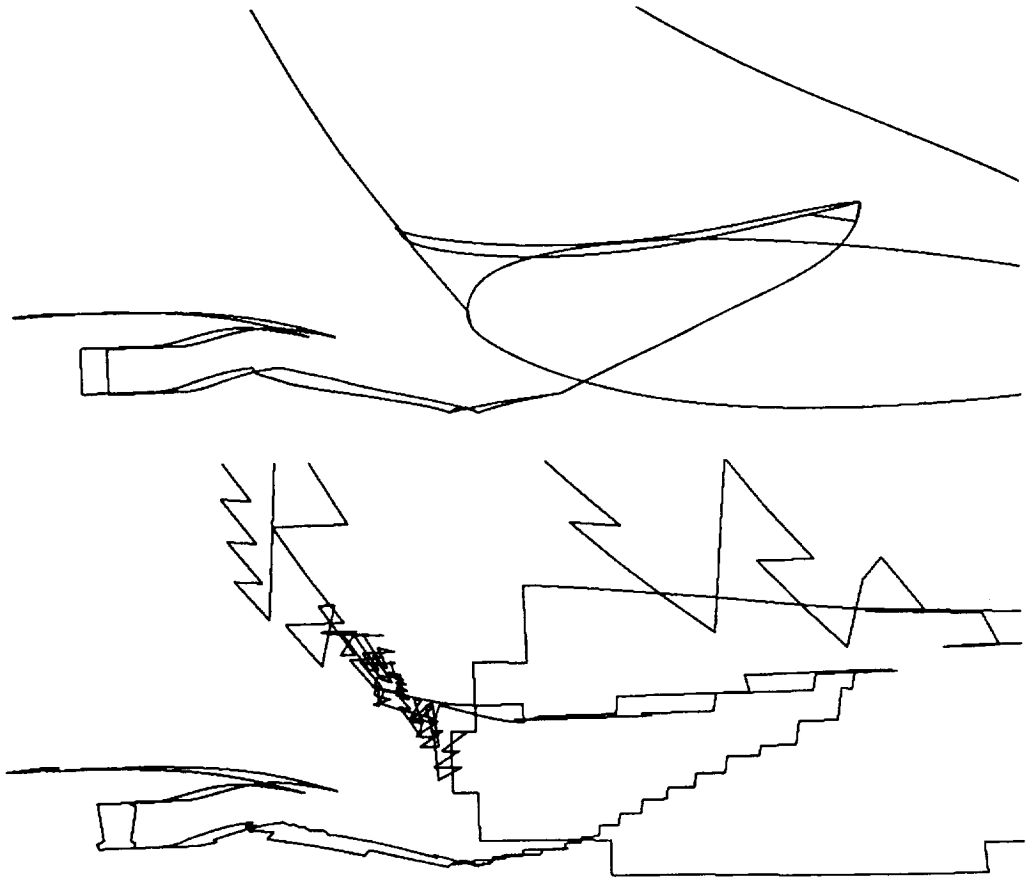

Figure 7. Associated Paths

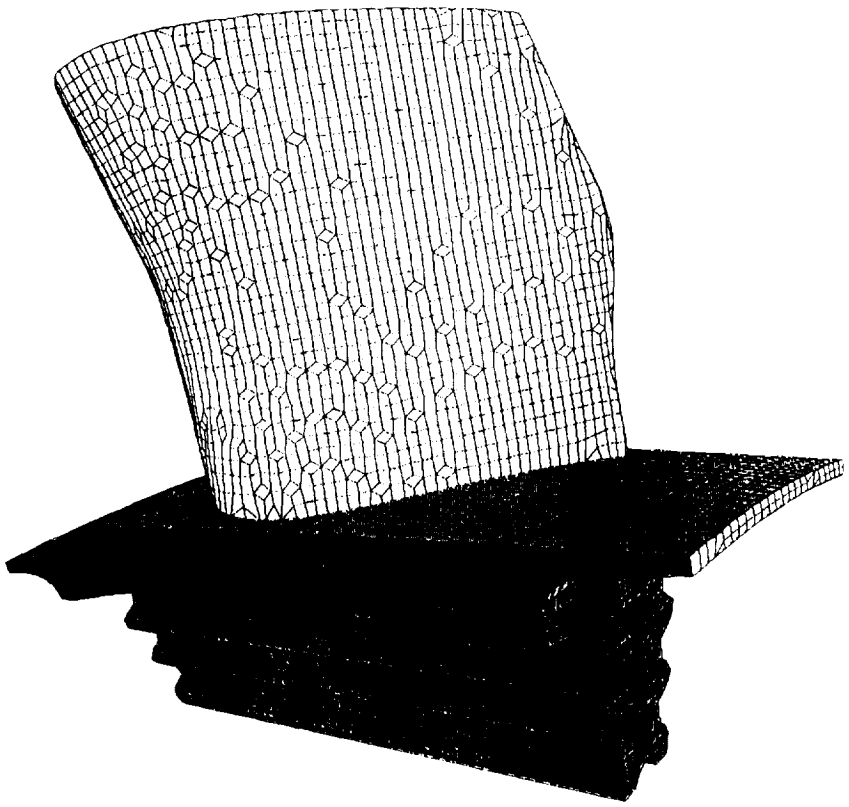

Figure 8. Turbine Blade Grid Surface

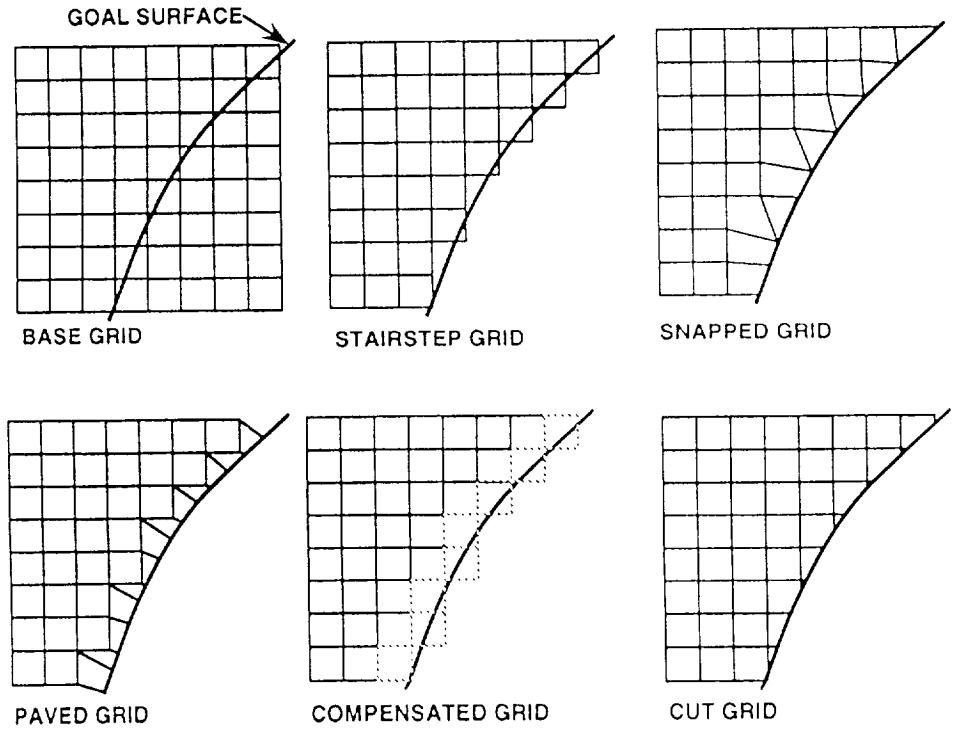

Figure 9. Surface Cell Methods

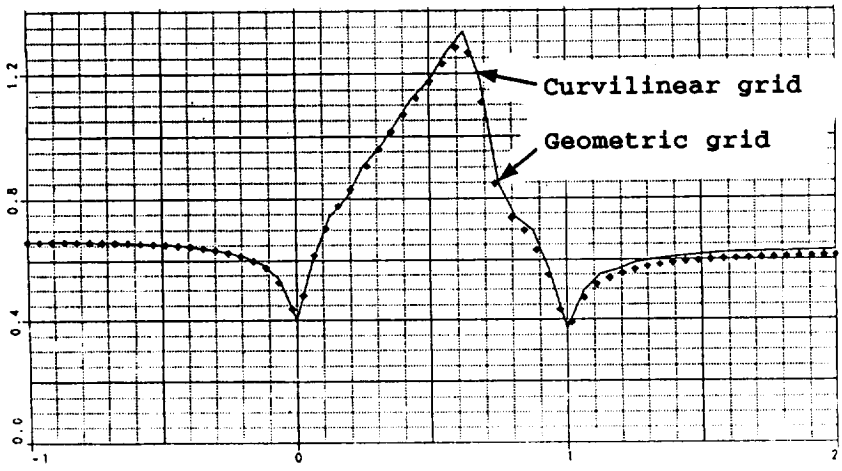

Figure 10. Flow over a Circular Bump in a Channel

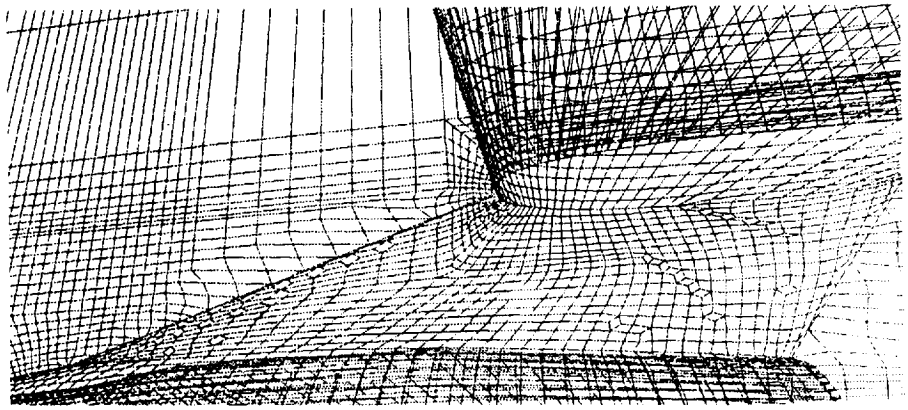

Figure 11. Pylon added to Wing/Nacelle Grid

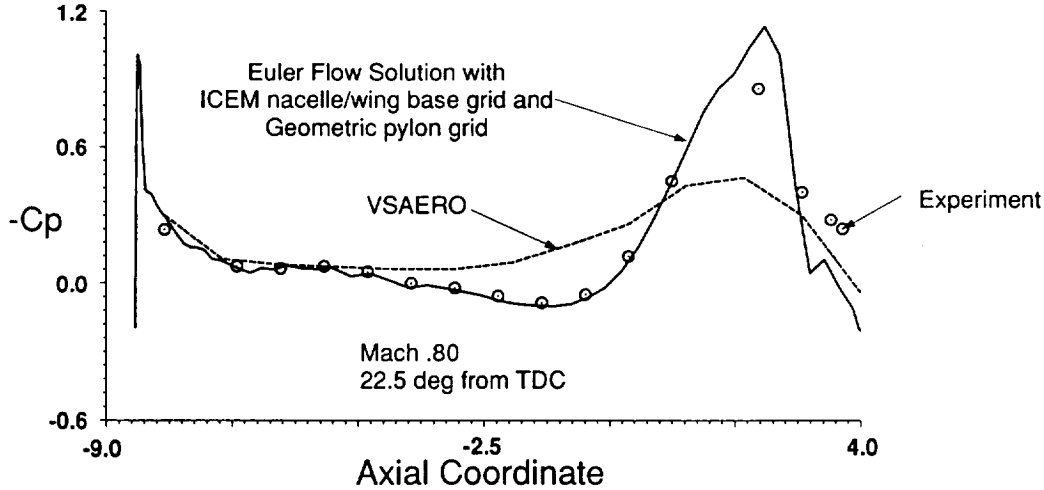

Figure 12. Wing/Pylon/Nacelle Flow Comparisons

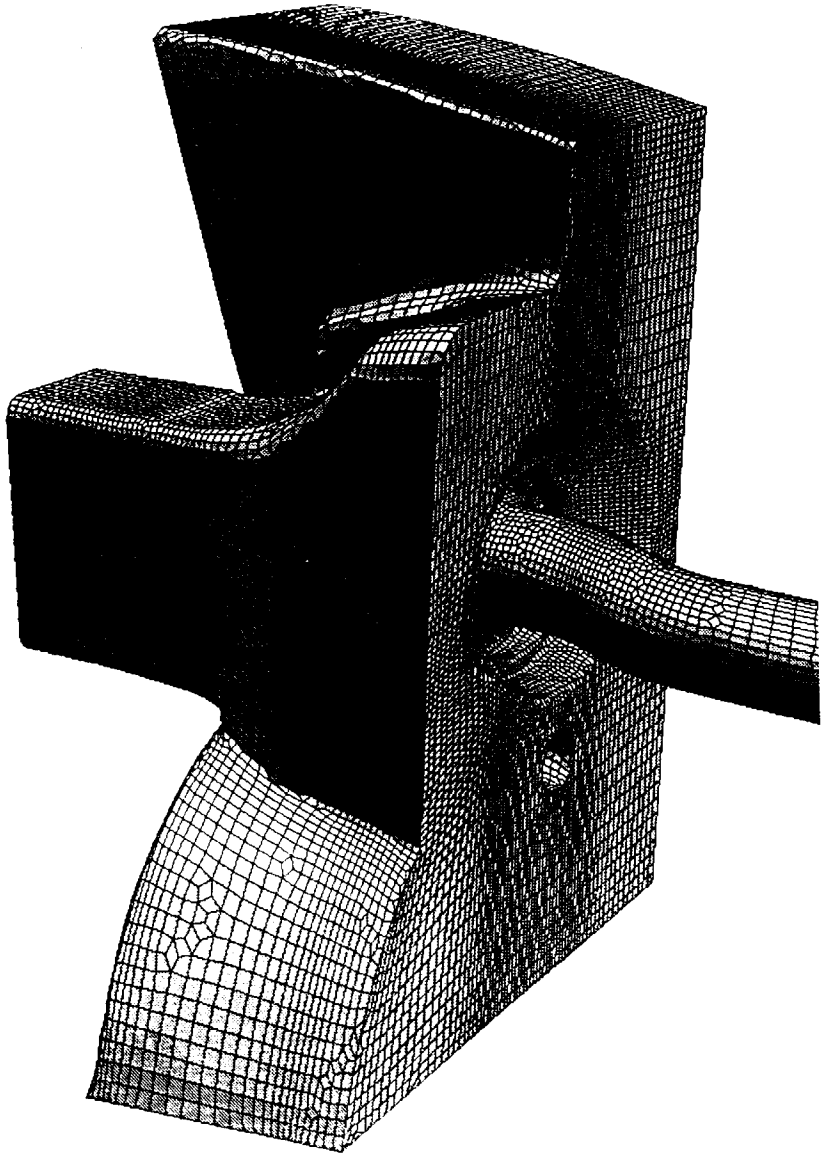

Figure 13. Helicopter Inlet Flow System Grid

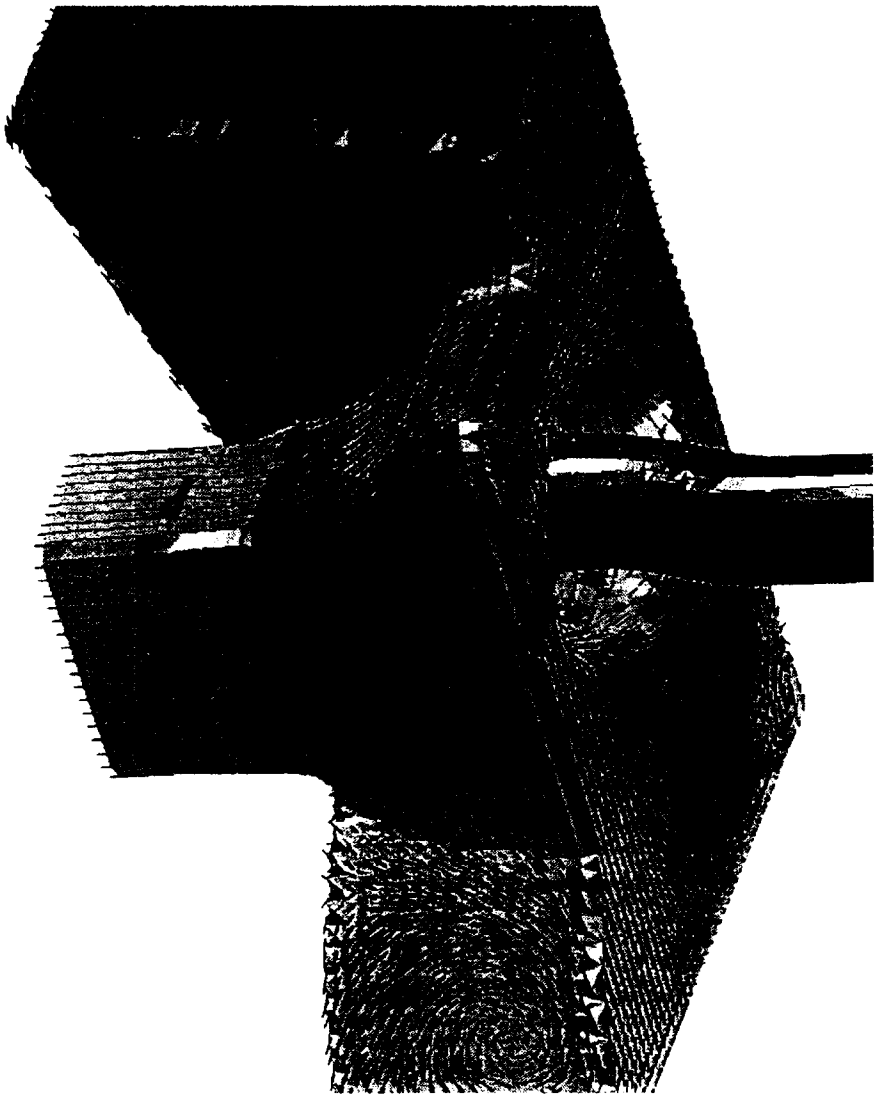

Figure 14. Helicopter Inlet Flow System Tufts

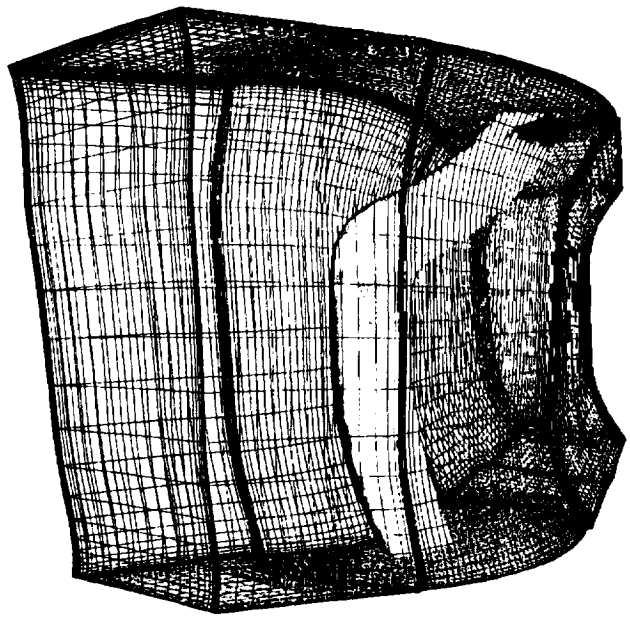

Figure 15. Probe Added to Compressor Grid

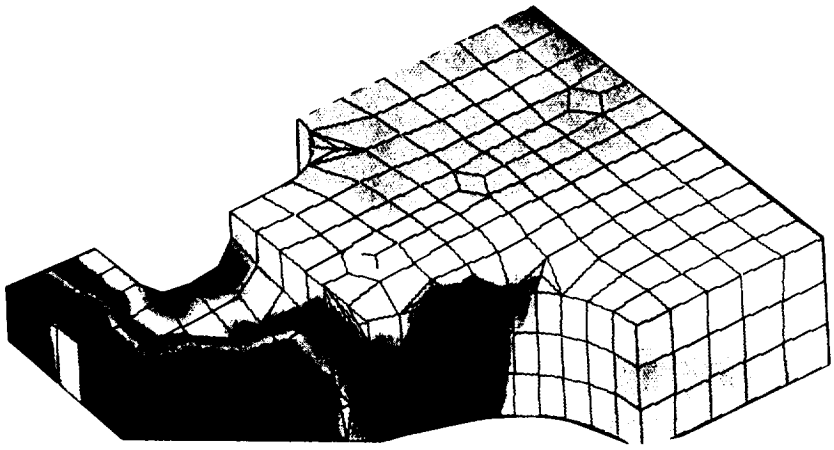

Figure 16. Structural Analysis of Bracket

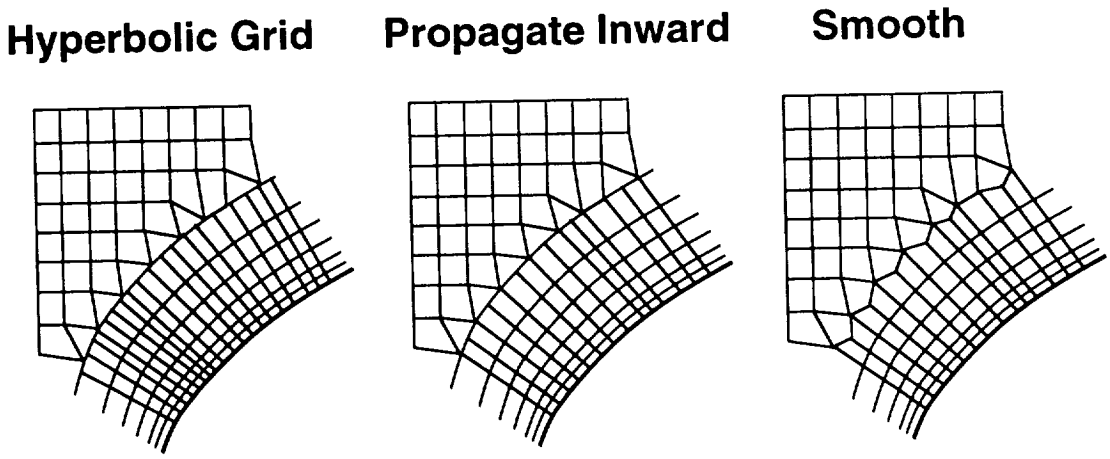

Figure 17. Viscous Grid Generation Process

AUTOMATIC ZONING FOR STRUCTURED MULTI-BLOCK GRID