AUTOMATIC BLOCKING FOR COMPLEX THREE-DIMENSIONAL CONFIGURATIONS

John F. Dannenhoffer, III*

United Technologies Research Center
East Hartford, CT 06108

ABSTRACT

A new blocking technique for complex three-dimensional configurations is described. This new technique is based upon the concept of an abstraction, or squared-up representation, of the configuration and the associated grid. By allowing the user to describe blocking requirements in natural terms (such as "wrap a grid around this leading edge" or "make all grid lines emanating from this wall orthogonal to it"), users can *quickly* generate complex grids around complex configurations, while still maintaining a high level of control *where desired*. An added advantage of the abstraction concept is that once a blocking is defined for a class of configurations, it can be automatically applied to other configurations of the same class, making the new technique particularly well suited for the parametric variations which typically occur during design processes. Grids have been generated for a variety of real-world, two- and three-dimensional configurations. In all cases, the time required to generate the grid, given just an electronic form of the configuration, was at most a few days. Hence with this new technique, the generation of a block-structured grid is only slightly more expensive than the generation of an unstructured grid for the same configuration.

BACKGROUND

One of the major impediments to the use of computational simulations for predicting the performance of real-world configurations is the time needed to set up a suitable computational grid. It is not uncommon for it to take well in excess of three months to generate a suitable grid for a new type of configuration. Clearly, such a long lead time makes the use of simulations practical for only a small number of important problems and for routine design work which involves relatively small changes from previous configurations (wherein the grid generation work can be amortized over many thousands of computational simulations).

Computational grids for fluid flows are especially hard to generate because of the stringent requirements imposed by the flow physics and subsequently the flow solvers (such as surface orthogonality and cell aspect-ratios of about 10^6). It has proved to be very difficult to adapt the automatic techniques used within the structures community to situations with such stringent requirements.

In order to understand why grid generation is such a bottleneck, it is useful to take a quick look back at how block-structured grid generation software has evolved to its present state.

The first generation of block-structured grid generators were batch-oriented systems which extended, in a rather straightforward way, the grid generation techniques which had previously been developed for

*Senior Research Engineer

two dimensions. They allowed highly-trained experts to generate grids over arbitrarily-complex three-dimensional configurations through many thousands of lines of command-line control information. The most widely known of the first generation systems was **EAGLE**[1].

The second generation systems (which represent the current state-of-the-art) are workstation-based techniques which rely on high-speed graphics to allow users to interactively generate the same control information. With these systems, the experts can generate grids in less time, and with fewer errors, than they could with the batch-oriented systems. But the grids are still manually generated piece by piece, over a period of many months. Example second-generation systems include **NGP**[2], **ICEM-CFD**[3], **GRIDGEN**[4], **RAMBO-4G**[5], **3DGRAPE**[6], **PATRAN**[7], and **FIDAP**[8]. All of these suffer the same important disadvantage, namely that the grid generation process is very labor intensive.

In order to determine where all the labor is consumed, it is useful to break the process into its major steps:

Prepare the configuration — as expected, the first step in the grid generation process is the preparation of the configuration to be analyzed. The first part of this step requires that the grid generation system acquire the geometry, most preferably by a direct link to the CAD system on which the configuration was designed. Unfortunately, the configuration which the CAD system describes is seldom that which one wishes to analyze; for example, the actual configuration generally contains many geometric details (such as a probe) which are not germane to the analysis being performed. In addition, one often needs to create "artificial" surfaces to bound the domain, such as in the far-field or over an engine inlet. This preparation step usually accounts for the largest expenditure of labor. (It should be noted that this step is exactly the same, regardless of whether a block-structured or an unstructured grid is to be generated.)

Design blocking plan — the user has to decide upon a suitable grid topology for the given configuration, keeping in mind both the expected flow physics as well as any restrictions which the flow solver may impose. The user typically meets these requirements by choosing near-body topologies which are locally consistent with each component of the configuration, and then connecting these blocks in a way that ensures, or at least encourages, grid line smoothness in the field. At present, this step is an art, requiring much intuition, practice, patience, and unfortunately trial-and-error. This step does not in itself require much labor, but any changes made here can result in large labor expenditures in the following steps.

Implement blocking plan — the blocking plan just designed has to be described to the computer so that the grid can be generated. Current approaches require that a user transform the design (which the User's Manuals typically suggest that the user "sketch") into numbered or named points, lines, and surfaces. In addition the user must prescribe how the primitive components are to be connected in the final grid. The user is also required to set up the number of grid lines in each direction in each block. This step can consume a large portion of the block-structured grid-generation effort.

Generate grid — the grid point locations for all surface and field points are defined, either algebraically and/or through a PDE-based grid generator. This is the part of the process at which most current multi-block grid generators excel. In many systems, this step is a push-button operation which requires very little human time (although it may require a rather large amount of CPU time).

Assess grid quality — the final computational grid is examined for irregularities such as excessive skew and stretch, cells which are too small or large, etc. If the analysis of the grid reveals any deficiencies, then the user must go back to one of the steps listed above, fix errors or make different choices, and repeat. Although tedious, this step is usually performed rather quickly; the major impact of this step to the overall effort is the time incurred in iterating back to one of the steps above.

Currently, the major bottleneck is in the design and implementation of the blocking plan, steps 2 and 3 above. For complex three-dimensional configurations, these two steps can consume several man-months of effort. The objective of this work is to develop tools and techniques for efficiently performing these two operations for arbitrary three-dimensional configurations.

TECHNICAL APPROACH

This paper presents a new method for designing, describing, and subsequently realizing, block-structured grids around complex three-dimensional configurations. It is assumed that the **Prepare the configuration** step described above has already been completed. Additionally, very little will be discussed about the **Generate grid** and **Assess grid quality** steps; both are discussed in detail elsewhere[9, 10].

Description vs. Prescription

The basic idea behind the automatic generation of block structures is to change the user paradigm from one which is *prescriptive* to one which is *descriptive*.

In a prescriptive process, the user has a set of low-level tools which prescribe exactly *how* the problem is to be solved. Examples of these low-level tools include drawing a line in space to serve as an edge of some block, generating an edge which is constrained to lie along a certain input surface, and connecting four edges into a face. While control at this level is certainly very powerful (and sometimes indispensable), the labor associated with such an approach grows very rapidly, especially in configurations with multiple interacting components.

Alternatively, a descriptive process describes *what* the grid should look like. Example descriptions include the fact that the grid should wrap around the wing leading edge, that the store grid should lie within the wing grid, and that the grid should be orthogonal to all solid surfaces with a user-defined off-body spacing to the first field point. Such descriptive statements are very powerful; in fact, they generally correspond to between ten and a hundred low-level prescriptive commands. Additionally, since descriptive commands represent high-level concepts, they provide a framework for divide-and-conquer approaches, wherein the grid in the vicinity of each component is described separately and the automatic blocker worries about interfacing them to achieve an overall block-structured grid.

It should be noted that there is an exact equivalence between the prescriptive and descriptive approaches; the only real differences between the two is the economy of the expression which the descriptive approach provides.

Abstraction

The key to describing, rather than prescribing, a block-structured grid is the direct preparation (by the user) of an abstract "sketch", herein called an *abstraction*. The abstraction can be thought of as a "squared-up" representation of the given configuration and of the computational grid which is to be produced. As will be seen below, typical configurations are generally abstracted by brick-shaped elements which represent the various components of a configuration.

This idea was inspired by the earlier work of Allwright and his colleagues[11], who showed the utility of such an abstraction in describing arbitrarily-complex three-dimensional grid topologies. Related work by Shaw and Weatherill[12] has also used the concept of abstraction. In both cases, the configurations of interest were all airplane-like, consisting of components such as fuselages, wings, and nacelles.

Although there are many similarities between the present work and the works of Allwright and Shaw, there are some major differences which make the present work more amenable to general configurations

such as are found in internal flows (for example, ducts and pumps) and traditionally non-aerodynamic configurations such as elevator hoistways. (Details of these differences will be published at a later date.)

The exact technique used to specify the abstraction of both the configuration and grid and their properties in the current technique is through the drawing of *blocking objects* on an abstract *background grid.* These blocking objects, which come in a variety of types (cubes, wraps, attaches, holds, rulers, and spacers), closely correspond to the concepts which one "sketches" during the design of a block-structured grid.

In the sections which follow, each type of blocking object, and the role it plays, is described. For illustrative purposes, the configuration which is used is a two-dimensional airfoil in a wind tunnel. It should be noted that the tools and techniques are equally applicable to more complex configurations in two and three dimensions, as demonstrated in the **Applications** sections below.

The Background Grid

The space in which the abstraction is drawn is covered by an integer Cartesian grid called the *background grid*, which can be thought of as a piece of graph paper (in either two or three dimensions). The resolution of the background grid is chosen so as to be fine enough to describe the various components of the configuration and block structure, yet coarse enough to allow background grid operations to be performed efficiently. All blocking objects are placed on the background grid in such a way that their corners or endpoints lie exactly on an integer coordinate of the background grid (that is, where the lines on the graph paper cross). While this limits the number of locations and the sizes of the blocking objects which are possible, such a restriction is warranted since it makes it possible to efficiently determine the relative locations of various objects (such as object "A" is to the left of object "B" or that object "C" is contained within object "D"). Besides the ability to specify relative positions, the actual background grid coordinates at which an object is placed is arbitrary. That is to say, the absolute distance between points on the background grid is insignificant.

Abstraction of Configuration: Bound Cubes

The first step in describing a grid is the creation of an abstraction of the configuration around, or inside, which a grid is required. This abstraction, or "squared-up" representation, is built up with a series of brick-shaped objects which are call *bound cubes.* In the current implementation, the user "draws" bound cubes by selecting a pair of background grid locations to specify opposite corners of the bound cube; the rectangular region between these points is automatically filled in with the bound cube which is rendered as a solid-colored box.

As an example, consider the configuration made up of an airfoil and its wake in a wind tunnel; a suitable abstraction is shown in Figure 1a. Notice that the airfoil, wake, and outer boundary are each represented as a rectangle (even though the actual geometry is not rectangular). Of significance in this figure is the fact that the airfoil is completely contained within the outer boundary, that the wake is attached to the right of the airfoil, and that the wake is also connected to the downstream (right) portion of the outer boundary. Recall that since the size of each increment on the background grid carries no significance, the airfoil bound could have been placed anywhere within the outer boundary, yielding a completely equivalent representation.

At this point, a slight digression is in order to describe the basic interpretation of the abstraction. The default block-structured grid which is generated corresponds to an H-type grid. Blocks are generated to fill the portions of the background grid which are outside the bound cubes. Specifically, block boundaries will be generated to yield the minimum set of blocks which are both all rectangular (in the abstract space) and which have one-to-one face matches with all other blocks; additionally, each face of each block can at

most be associated with one configuration surface.

As in all H-type grids, the grid lines basically run horizontally and vertically throughout the domain. So for Figure 1a, there are some grid lines which begin at an upstream portion of the outer boundary (in the region labeled "a") which run horizontally and which pass below the airfoil/wake and end at a downstream portion of the outer boundary (labeled "g"). Similarly, grid lines which begin in the region labeled "c" pass above the airfoil and end in the region labeled "h". Finally, there are grid lines which begin upstream (to the left of) of the airfoil (in the region labeled "b") which end at the front of the airfoil surface.

The other family of grid lines (those which run vertically on the background grid), has similar behaviors. Some start at the lower portion of the outer boundary (in the region labeled "d"), pass in front of the airfoil, and end at the upper portion of the outer boundary (labeled "i"). Others start in the regions labeled "e" and "f" and end on the lower surface of the airfoil and wake, respectively. Still others begin on the upper airfoil and wake surfaces and end on the upper boundary in the regions labeled "j" and "k", respectively. The corresponding grid for this configuration is shown in Figure 1b. Note that seven blocks are created, with block boundaries emanating only from the "corners" of the airfoil.

In the above discussions, the terms "lower surface of the airfoil", "upstream portion of the outer boundary", etc., should not be taken too literally; the terms simply serve to indicate approximate parts of the configuration, with the exact locations determined automatically so as to make the grid as smooth as possible. Note also that terms such as "some grid lines" were used in the above discussion; the exact number of grid lines in each region is specified by another blocking object known as a ruler (which is described below).

Abstraction of Grid: Field Cubes and Wraps

As stated above, once the configuration has been abstracted with bound cubes, a default H-type grid can automatically be generated. In order to change the grid topology either locally or globally to another type, such a C-type, other blocking objects must be added to the background grid. Field cubes and wraps are those blocking objects.

A *field cube* is a rectangularly-shaped box which is placed on the background grid to force additional block boundaries to be generated. As a general rule, all grid blocks will lie either within or outside of each field cube. Notice that field cubes do not change the topology of the grid; they just give the user a means of breaking the default blocks into smaller blocks.

When the background grid is first generated, a special field cube called the outer boundary is placed around the periphery of the background grid. All other blocking objects must be placed on or within this outer boundary — a rule which is consistent with the notion of an outer boundary.

For example, consider again the airfoil/wake in a wind tunnel. If a field cube is added (as shown by the unfilled rectangle in Figure 2a), then the grid which is generated is still H-type (see Figure 2b), but now contains 18 blocks. In concept, breaking blocks in this way should not change the grid point locations; in practice, the locations are slightly different due to the fact that the elliptic smoother is not run until convergence.

So far, the bound and field cubes have been sufficient to describe "streamline-like" grids, but not "wrap-around" grids, as are frequently required. To remedy this, *wraps* are added to the background grid.

Wraps are specified by selecting two cubes (either bound or field), and then automatically creating "diagonal" lines to connect their respective corners. Shown in Figure 3 are the nine different wraps which are possible in two dimensions; in three dimensions, the number grows to twenty-seven. The first wrap shown in the Figure, in which the inner cube is fully within the outer cube, corresponds to an O-type grid: one family of grid lines are bent by each of the diagonal corners such that they encircle the inner cube while the other family are basically radial. The second row of wraps in the Figure, which correspond to

C-type regions, result when one edge of the inner cube is coincident with an edge of the outer cube. The last row corresponds to corner wraps.

Figure 4 shows the effect of wrapping the airfoil bound cube within the field cube which was added in Figure 2. This results in a C-type grid around the airfoil, but within an outer H-type grid.

If one considers the trapezoidal regions generated by a wrap to be field cubes, then the concept of wrapping can be nested to create an almost unlimited set of block topologies for a given configuration. For example, one may want to create a wrapped grid around a fuselage. Then in the region next to the fuselage which contains the wing, another wrap may be desired to create a C-type grid on the wing. Then, the region below the wing can be further wrapped around a pylon to create a local C-grid there. Although it may seem confusing, users have reported that thinking about the block structure in this manner is much less confusing than the alternative of manually constructing complex block structures for such configurations.

Attachment between Abstraction and Configuration: Attaches

A new development of this work over that previously reported[13, 14] is the method by which the abstraction and configuration are attached. Presently, a very simple blocking object called an *attach* is used by which the user gives the correspondence between each corner of each bound cube (and the outer boundary) with the appropriate location on the configuration. For example, the attaches needed for the airfoil/wake configuration (discussed previously) are shown in Figure 5a. The easiest place to actually see the attaches are those associated with the airfoil and wake; the attaches at the corners of the outer boundary are not as obvious because the abstraction and configuration happen to be rendered at the same location, masking the line which connects the two. The attaches associated with the airfoil are of two types: those drawn as a star specify that the grid generator is required to put the grid point at the *exact* configuration point to which it is attached; those drawn as a circle specify that the grid generator can "slide" the grid point along the selected configuration surface so as to make the grid as smooth as possible. The latter is a particularly nice feature since it eliminates the need for the user to exact specify *a priori* all block corners which are on the interior of a surface.

This Figure also points out another feature of the abstraction. The mapping between the configuration and the abstraction is a many-to-one relationship. That is to say, each point on the abstraction corresponds to one point on the configuration, whereas each point on the configuration can correspond to many points in the abstraction. This is evident at the airfoil trailing edge, where the trailing edge point maps to both the top and bottom of the airfoil's abstraction.

Other Blocking Objects: Rulers, Holds, and Spacers

In the above descriptions, terms such as "some grid lines" were used to describe the number of grid lines in each region of the grid. A blocking object called a *ruler* is the method for exactly specifying the number of points in each region. Shown in Figure 5b are the rulers for the H-grid problem.

One of the useful features of the abstraction is that it is relatively easy to propagate sizes from one region to another; grid regions (blocks) which abut are required to have the same number of grid lines. Similarly, it is relatively easy to perform size arithmetic, that is to determine the number of grid lines in various regions by simple arithmetic calculations. The only restriction on sizes is that they be consistent (that is, all size arithmetic yield the same results) and that enough sizes be prescribed so that the number of grid lines in all blocks can be calculated.

Another blocking object called a *hold* gives the user a way of easily specifying geometry such as symmetry planes and outer boundaries. A hold represents a plane in physical space in which one of the space coordinates is held fixed; the value of the fixed coordinate is determined automatically by the attaches which are associated with the hold. Unlike surface-surface intersections which must be explicitly generated,

the intersection between two holds or between a hold and a surface does not have to be directly generated by the user; this makes holds an easy way of representing symmetry planes for half-body configurations. Because holds act as a surrogate for an actual configuration surface within the grid generation process, they will not be discussed further here.

The final blocking object, a *spacer*, is the method used to describe grid properties such as wall spacings and orthogonality. Rather than requiring that the user specify these quantities along the edges which are (logically) perpendicular to a surface, spacers allow the user to specify the surface directly. Spacers give the user the ability of directly describing the grid requirement the "grid lines should be normal to the wing surface and that the wall spacing should be x." Since spacers do not directly affect the blocking, but rather are passed along to the grid generator in the form of required edge spacings, they will not be discussed further here.

Transformation of Abstraction to Grid

Once the grid topology is specified by placing blocking objects on the background grid, a set of transformations, which are fully described in [13], are used to generate a suitable assembly of grid blocks. The key steps include:

- transform the field cubes and wraps into grid blocks;
- fill the remaining portion of the background grid with (logically-)rectangular grid blocks
- break grid blocks so as to establish one-to-one face matches (partial face matches are not allowed — see [15] for a method for overcoming this limitation);
- determine the sizes (number of grid points) in each block;
- eliminate degenerate blocks (that is, blocks which contain no grid points);
- associate the abstraction with the configuration surfaces and curves;
- allocate grid nodes for each block;
- use parametric transfinite interpolation to assign initial locations to all grid nodes which are constrained to lie on one (or more) of the configuration surfaces;
- use transfinite interpolation in space to assign initial locations to all other grid nodes; and
- apply a multi-block elliptic grid generation schemes to smooth the block boundaries (which were arbitrarily placed above).

TWO-DIMENSIONAL APPLICATIONS

This section begins by showing the three different grids around an airfoil/wake in a wind tunnel which were previously described in the **Abstraction of Configuration: Bound Cubes** and **Abstraction of Grid: Field Cubes and Wraps** sections. Figures 1, 2, and 4 show the abstractions and resulting grids for two different H-type and one CH-type grid. In all cases, the same configuration abstraction (bound cubes) and abstraction-to-configuration attachments (attaches) were used; the only differences were in the grid abstractions (field cubes and wraps). In other words, once the time-consuming task of setting up a configuration was done, the new technique allowed the user to quickly experiment with differing grid

topologies. This collection of grids required less than five minutes to specify and generate on an engineering workstation.

To show the generality of this abstraction technique, grids for two completely different configurations were also generated. Figure 6 shows the abstraction and grid for an internal-flow passage containing eight circular posts and Figure 7 shows the abstraction and grid for a cross-sectional cut through a gas turbine combustor. Both grids required less than 30 minutes to describe and generate.

THREE-DIMENSIONAL APPLICATIONS

The first demonstration presented here is the full three-dimensional model of a gas turbine combustor, as shown in Figure 8. This configuration was abstracted in steps. First, an abstraction from an equivalent two-dimensional configuration was generated and then extruded into the third dimension. Second, the hole associated with the fuel injector was cut into the abstraction, with the grid blocks automatically readjusting themselves. And third, the eight dilution holes (four each in the upper and lower liners), were added with the grid blocks again responding. This multi-step process significantly reduces the number of items which the user has to consider at any one time, and thereby helps limit the number of user errors.

The second three-dimensional configuration is for a generic elevator hoistway with a streamlined car (Figure 9). This configuration, which is very different from the above, shows that the current set of blocking objects is flexible enough to handle very diverse configurations. This grid was generated in less than a day.

The third configuration is a generic fighter with a side-mounted inlet (shown in Figure 10). This configuration, which is more typical of those found in the aerospace industry, exhibits complications in the area where the engine duct and the horizontal tail intersect. (This is the reason why the abstraction of the horizontal tail is fatter than the wing.) In generating the configuration abstraction for this case, the configuration was conceptually broken into a series of sub-problems, each of which were attacked independently. The solutions to these sub-problems were stored as templates which were recalled and combined to tackle this complex problem. The entire grid for this case, including the development of the templates, took less than 3 days to complete, given just an electronic specification of the configuration.

Finally, a grid was generated for another fighter which was topologically similar to that shown in Figure 10, but which had a wing with a different planform and airfoil section. This grid (not shown because it essentially looks the same as that in Figure 10) was generated *completely automatically* with the exact same blocking objects as were used above. This demonstrates another power of this technique — parametric grids can be generated automatically for classes of configurations once the first grid has been generated.

CONCLUSIONS

A new technique which enables a user to describe, rather than prescribe, block-structured grids in two and three dimensions is presented. This new technique has many advantages over current techniques, namely:

- the blocking for a completely new configuration can be done in about an order-of-magnitude less time than is typically required using more traditional techniques. This makes block-structured grid generation again competitive with unstructured grid technology for a wide variety of problem types;
- since the configuration and/or grid topology can be easily modified, the user is encouraged to employ a divide-and-conquer approach to complex configurations. That is, the current technique allows users to build up configurations slowly, thereby reducing the complexity added at any step (and the associated chance of making an error);

- since a grid topology can be modified rather simply, the trial-and-error process which is unfortunately part of blocking design, is easily accommodated, with the end result being higher quality grids;
- templates of abstractions can be created for many frequently-occurring configuration components, thereby reducing the blocking problem for multi-component configurations to the concatenation and nesting of the appropriate templates. In fact, previous work in automatic grid generation[16] used optimization techniques and expert system technology to automatically generate grids in two dimensions; and
- the technique described herein is independent of the multi-block grid generator used, and thus is immediately applicable as a front-end to many current systems.

ACKNOWLEDGEMENTS

This work was conducted under the United Technologies Corporate Sponsored Research Program and is currently implemented both as a stand-alone system (called `MKBLOX`) and as an application module within the National Grid Project (`NGP`). The author would like to thank Prof. Nigel Weatherill for the many hours of discussions about this work. Also, the author would like to thank his many colleagues who suffered through many demonstrations of half-baked ideas; their comments were invaluable in making this technique applicable to real-world problems.

REFERENCES

[1] Thompson, J.F., Lijewski, L.E., and Gatlin, B., "Program EAGLE User's Manual," AFATL-TR-88-117 (Volumes I, II, and III), September, 1988.

[2] Gaither, A., Gaither, K., Jean, B., Remotigue, M., Soni, B., Thompson, J., Whitmire, J., Dannenhoffer, J., Weatherill, N., "The National Grid Project: A System Overview," NASA-CP-3291, May, 1995.

[3] Akdag, V., and Wulf, A., "Integrated Geometry and Grid Generation System for Complex Configurations," NASA-CP-3143, pp 161–171, April, 1992.

[4] Steinbrenner, J.P., Chawner, J.R., and Fouts C.L., "The GRIDGEN 3D Multiple Block Grid Generation System," WRDC-TR-90-3022 (Volume II), July, 1990.

[5] Widhopf, G.F., Visich, M., Than, P.T., Boyd, C.N., Huang, S-C., Oliphant, P.H., Stall, D.J., Swedberg, G.D., Shiba, J.K., and Coons, R.E., "Rambo-4G: An Interactive General 3-D Multi-Block Grid Generation and Graphics Package for Complex Multi-Body CFD Applications," Aerospace Report ATR-91(9990)-1, October, 1990.

[6] Sorenson, R.L., "The 3DGRAPE Book: Theory, Users' Manual, Examples," NASA-TM-102224, July, 1989.

[7] "PATRAN Plus User's Manual," PDA Engineering, July, 1987.

[8] "FIDAP Program" documentation, Fluid Dynamics International, Inc., January, 1990.

[9] Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W., *Numerical Grid Generation: Foundations and Applications*, North-Holland, 1985.

[10] Parmley, K.L., Dannenhoffer, J.F., and Weatherill, N.P., "Techniques for the Visual Evaluation of Computational Grids," AIAA-93-3353, June, 1993.

[11] Allwright, S.E., "Techniques in Multiblock Domain Decomposition and Surface Grid Generation," in *Numerical Grid Generation in Computational Fluid Mechanics '88*, Pineridge Press Limited, 1988.

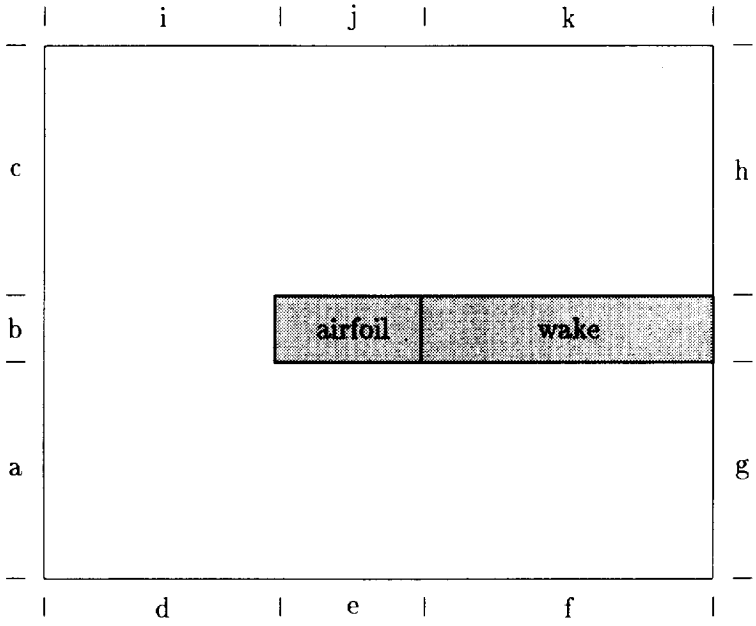
[12] Shaw, J.A., and Weatherill, N.P., "Automatic Topology Generation for Multiblock Grids," *Applied Math. and Comp.*, vol 53, pp 355–388, 1992.

[13] Dannenhoffer, J.F., "A Block-Structuring Technique for General Geometries," AIAA-91-0145, January, 1991.

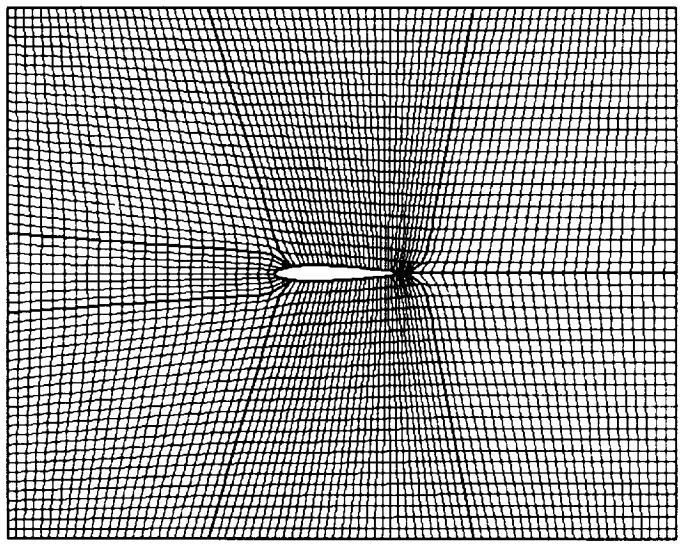
[14] Dannenhoffer, J.F., "A New Method for Creating Grid Abstractions for Complex Configurations," AIAA-93-0428, January, 1993.

[15] Dannenhoffer, J.F., "A Technique for Optimizing Grid Blocks," NASA-CP-3291, May, 1995.

[16] Dannenhoffer, J.F., "Computer-Aided Block-Structuring Through the Use of Optimization and Expert System Techniques," AIAA-91-1585, June, 1991.

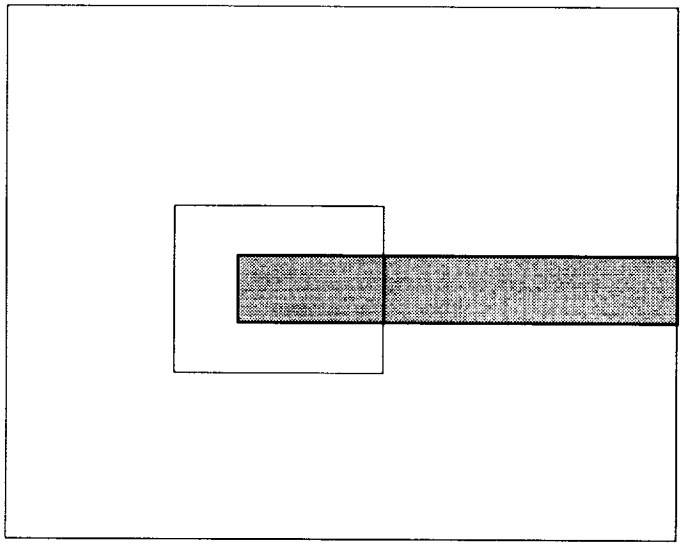


a) abstraction

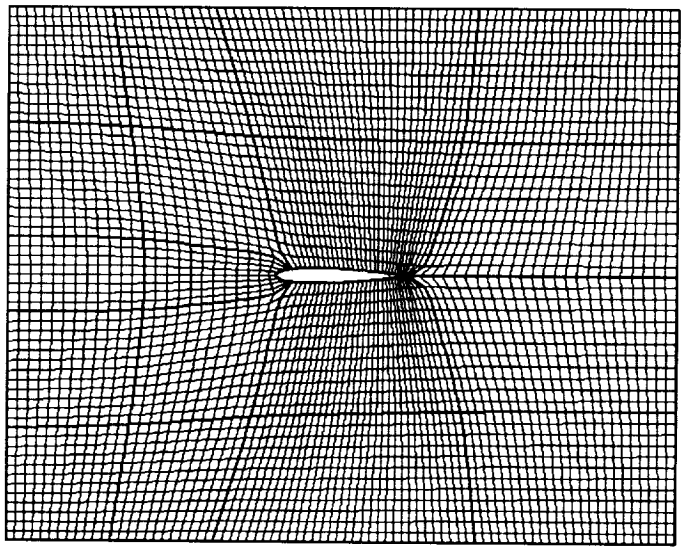


b) grid

Figure 1: H-grid around airfoil in wind tunnel.



a) abstraction



b) grid

Figure 2: H-grid (with field cube) around airfoil in wind tunnel.

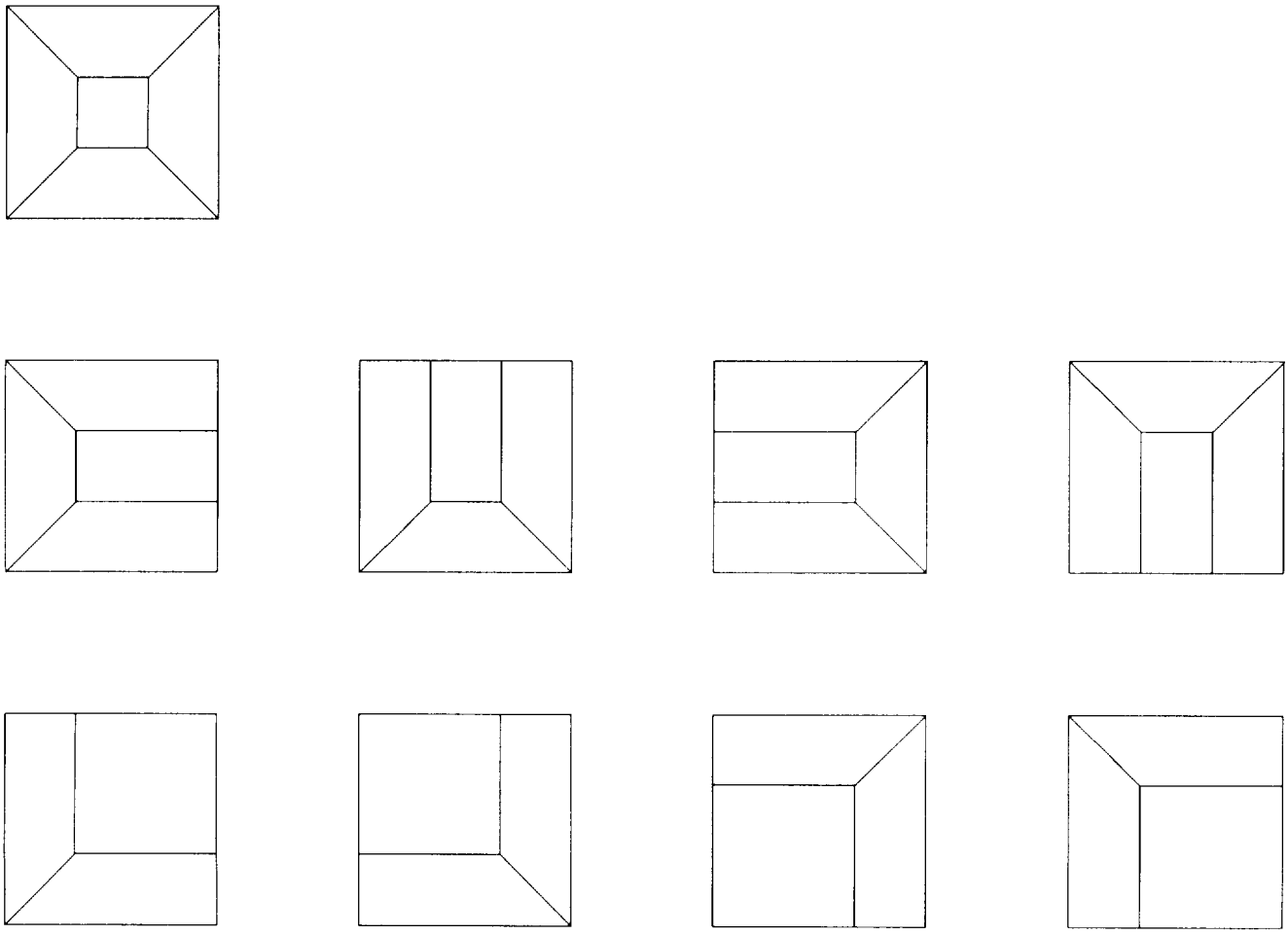
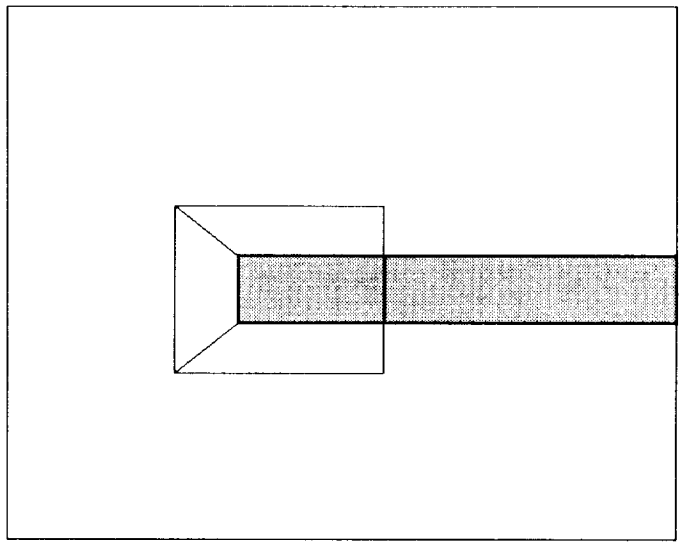
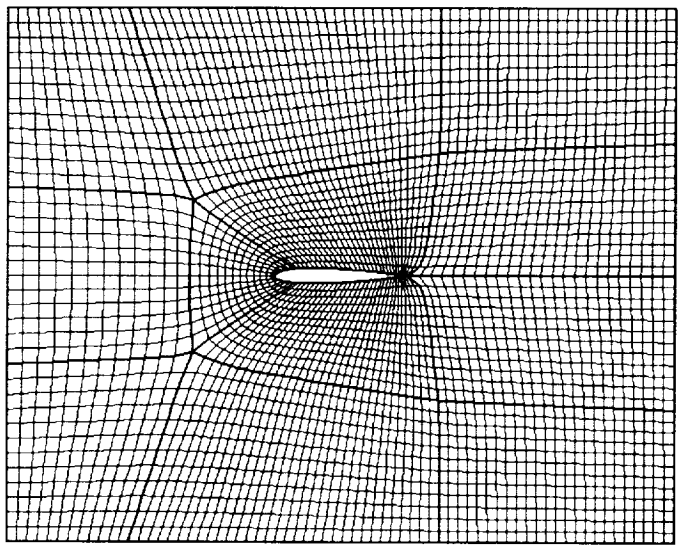


Figure 3: The nine possible wraps in two dimensions.

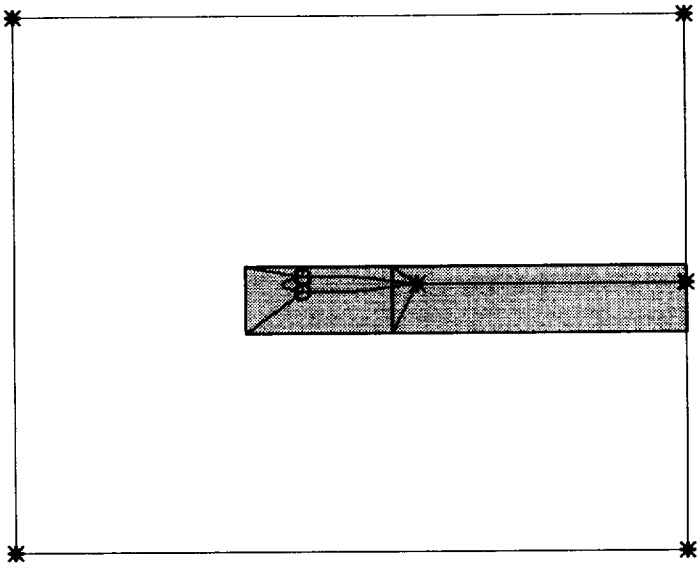


a) abstraction

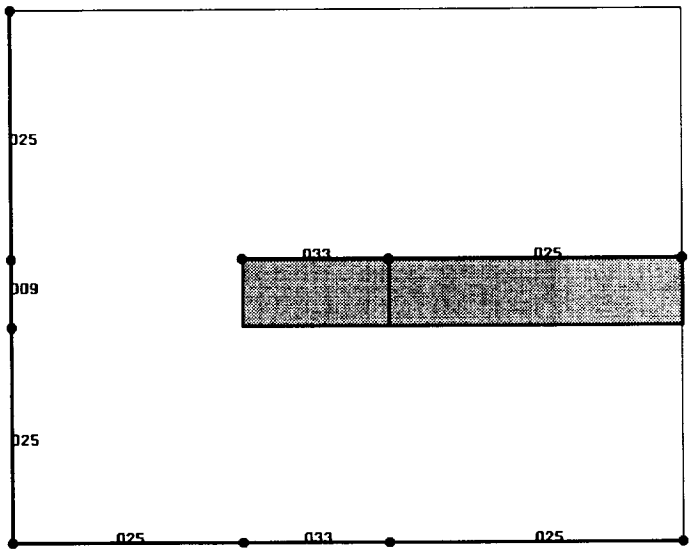


b) grid

Figure 4: CH-grid around airfoil in wind tunnel.

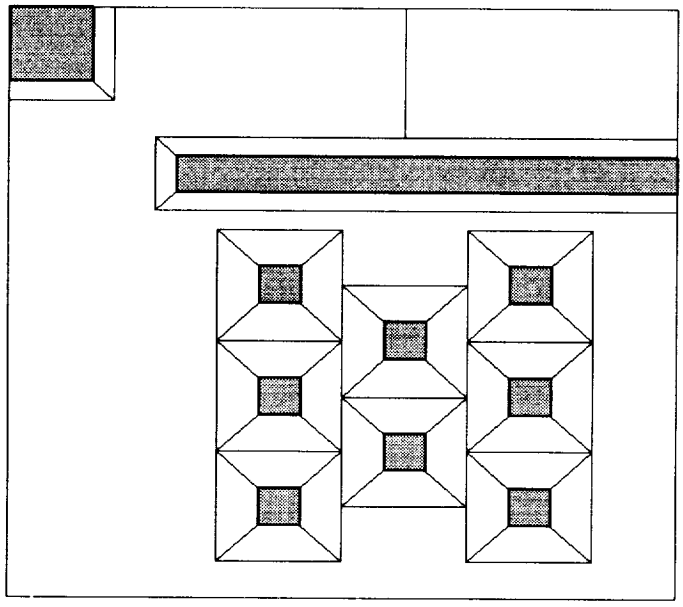


a) abstraction and configuration with attaches

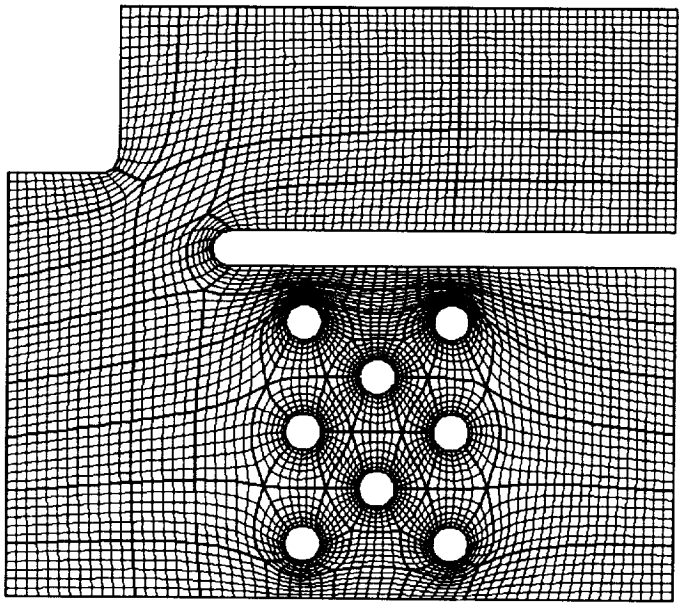


b) abstraction with rulers

Figure 5: H-grid around airfoil in wind tunnel.

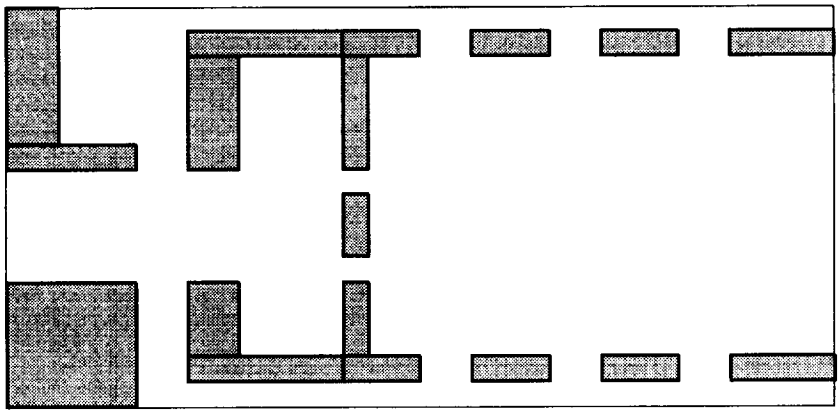


a) abstraction

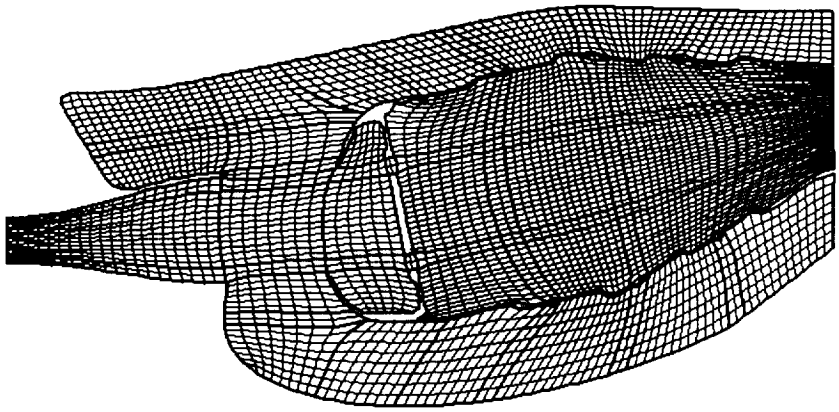


b) grid

Figure 6: Block-structured grid around passage with eight posts.

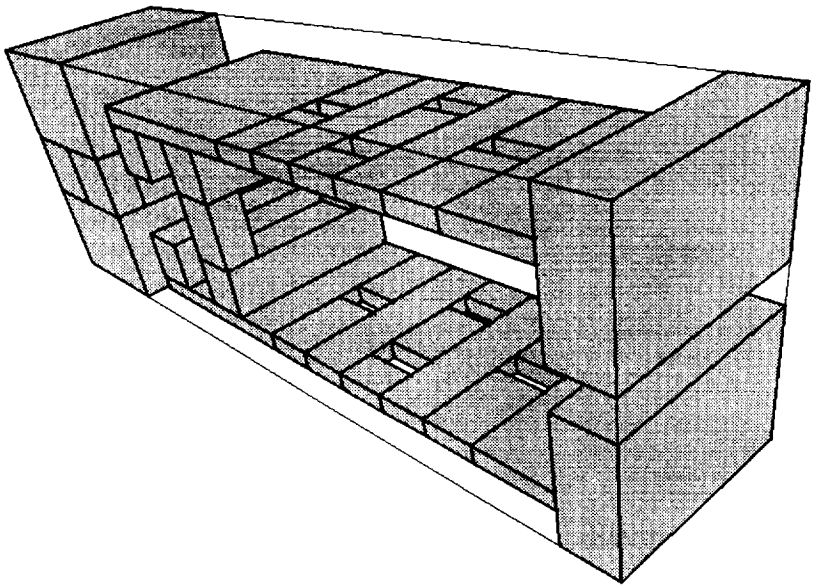


a) abstraction

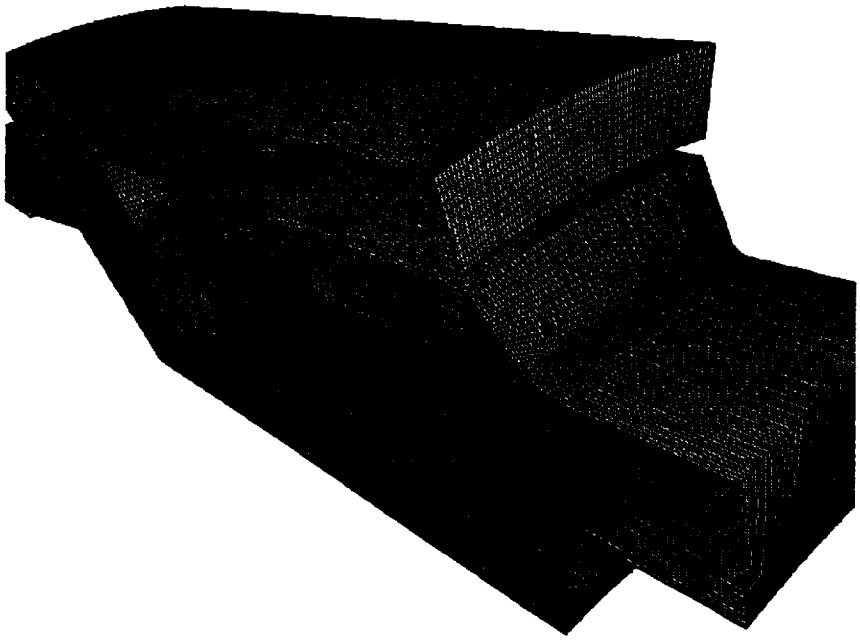


b) grid

Figure 7: Grid for combustor cross-section.

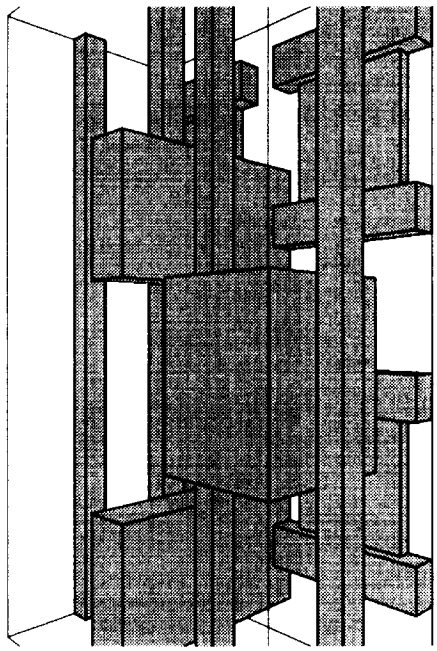


a) abstraction

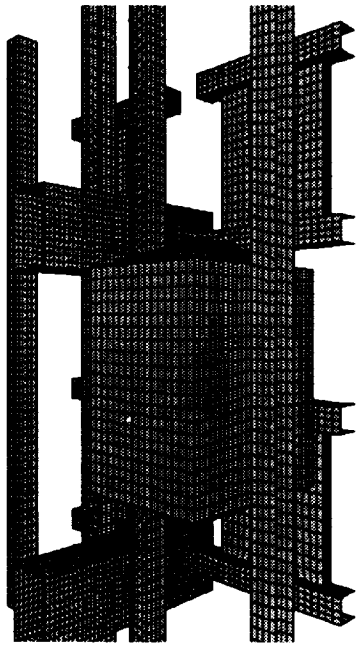


b) grid

Figure 8: Grid for three-dimensional combustor.

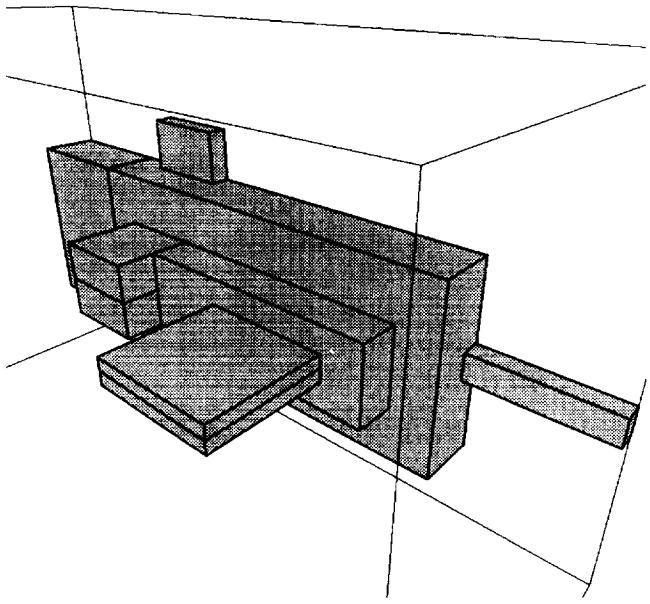


a) abstraction

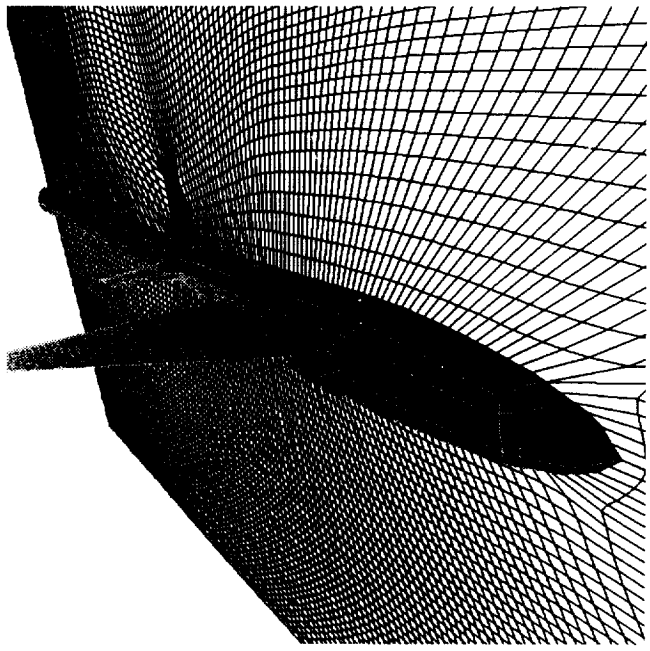


b) grid

Figure 9: Grid for elevator hoistway and streamlined car.



a) abstraction



b) grid

Figure 10: Grid for generic fighter.