[redacted]

N95- 28735

19951 22314 [illegible]

MULTIBLOCK GRID GENERATION WITH AUTOMATIC ZONING

Peter R. Eiseman

Program Development Corporation
300 Hamilton Avenue, Suite 409
White Plains, NY 10601

SUMMARY

An overview will be given for multiblock grid generation with automatic zoning. We shall explore the many advantages and benefits of this exciting technology and will also see how to apply it to a number of interesting cases. The technology is available in the form of a commercial code, GridPro®/az3000. This code takes surface geometry definitions and patterns of points as its primary input and produces high quality grids as its output. Before we embark upon our exploration, we shall first give a brief background of the environment in which this technology fits.

MULTIBLOCK GRID GENERATION

Multiblock grids provide the best data structure for a wide variety of analysis programs. Because the local coordinate pattern of grid points is regular, the accounting overhead for point to point linkages is low and is quite simple. This means that there are more algorithms available for analysis, and moreover, that those algorithms are generally more efficient. In addition, the local coordinate structure has been recognized to produce generally more accurate results. These advantages from a single coordinate grid, carry over into the more general setting where many blocks of coordinate grid are continuously glued together to form a multiblock grid. Without a continuous glue between blocks of grid, one has the problem of transferring data between either non-aligned or overset grids. This involves many issues which relate to the management of data and the preservation of accuracy. When the glue between blocks is continuous, then the data transfer problem is vastly simplified and accuracy is greatly enhanced. The accuracy is even further enhanced as the continuity level increases (e.g. continuous derivatives). Moreover, multiblock grids are also well positioned to take advantage of parallel computing environments.

The multiblock template provides zones for the various coordinate grid blocks to reside. This is a general structure and, as such, is one which can be used for arbitrarily complex configurations. It is, in fact, a coarse unstructured dissection of space into chunks. In distinction from what is typically called an unstructured grid, the cells define macro regions in space that are called zones and which generally have curved boundaries.

TRADITIONAL METHODS

In the traditional approach to multiblock grid generation, the creation of a grid is done in a piece by piece fashion. The process starts with the geometric region boundaries, continues by forming the zonal boundaries, proceeds by generating grids on each boundary section, and concludes with volume grid generation. The main distinctions between the

[redacted]

various traditional methods is the order in which the pieces can be put together, the numerical methods employed to treat each piece, the style of interactive graphics utilized to help the user deal with all of the pieces, and the ability to replay and edit parts of a session by the use of journal files.

In the course of applying a traditional multiblock generator, the user must not only insert rather large amounts of data but also must make judgments about that data. Such judgments often concern the relative spatial location of various objects. These can vary from something as basic as placing one zonal corner in space to more detailed matters such as determining both the shape and location of zonal edges. Unfortunately, the detailed judgments continue with such items as trying to appropriately adjust edge or surface grids on either side of a zone or region. Altogether, the traditional user is faced with the geometry and grid generation for corners, edges, and surfaces before he proceeds to volumes. In the usual progression of events, many judgmental errors are made and these translate into repeated operations. The result is great human inefficiency and loss of time. By the time volumes are considered, the real human work has usually been done.

THE AUTOMATIC METHOD

With the grid generation technology to be discussed here, there is a huge paradigm shift. There is no longer the rather manual construction and assembly of multiblock grids that is fraught with the error prone tendencies of human judgment. There is no longer the massive amounts of detailed data required of the user. There is no longer the huge reliance upon high end computer graphics to manage that manual activity. Instead, the amount of input data is enormously reduced, the detailed human judgment is gone, and the management of data is efficiently controlled with an appropriate language. Moreover, the quality of grids is much, much higher. It is the same mathematical underpinnings that reduce the user effort which also account for the substantial increase in quality. Basic grid quality appears in the form of near orthogonality, smoothness, low warpage, and curvature clustering for both concave and convex boundaries.

The discussion will focus upon the program called GridPro®/az3000 which is a commercial product and which is the most automatic and powerful multiblock grid generator available. Unfortunately, there are too many grid generators that are referred to as "automatic" and, as a consequence, user's are quite naturally confused. After all, a good number of the traditional multiblock grid generators are called automatic ones. To make some sense out of this situation, the real question that should be asked is "when does it become automatic? or "from what stage is it automatic?"

As the story is examined, it will be seen that GridPro®/az3000 is automatic from an earlier stage than any other 3D multiblock generator and produces higher quality grids as well. That stage, is defined by the presence of region boundaries, the pattern of grid points, and a small handful of scalar parameters. Surface grid generation, zone construction, and the intersection between surfaces are examples of the tasks that are solved as part of the automatic solution. Of the user inputs, the primary one is the pattern of points which is referred to as the grid topology. The language that organizes the pattern of points is called the "Topology Input Language" (TIL). TIL codes are extremely flexible items and can be used to generate grids about various classes of configurations. In this context, GridPro®/az3000 compiles and runs TIL codes. While future developments are likely to include both interactive and automatic TIL code generation, our concentration shall be on the TIL language, its benefits, and the grid generation capabilities. An overview of GridPro®/az3000 is given in Eiseman, Cheng, and Häuser [1] and a detailed account is given in the user's manual [2].

GridPro®/az3000

The practical industrial needs for engineering analysis are speed, accuracy, reliability, and realistic configurations. The latter often leads to massive complexity. Such complexity appears when configuration boundaries contain a very large number of geometric components. A large number of grid points is also desired but is balanced against the available computer resources. With massive complexity, comes the requirement to deal with an arbitrarily large number of blocks. For the traditional multiblock grid generators, this will require an enormous amount of human time, and thus, wall clock time to get the grid.

GridPro®/az3000 satisfies the practical needs of industry. Because of the high grid quality, various analysis programs have produced results on the first try, are more accurate, and are faster to convergence. Because of the automation, massive complexity is efficiently handled with a large number of blocks. The count can go to a thousand, ten thousand, or more depending upon the scope of the analysis.

THE TOPOLOGY INPUT LANGUAGE

The scope of analysis varies from small to large as various boundary components are assembled into successively more comprehensive region configurations. In a parallel sense, the topology input language (TIL) builds successively more complex grid point patterns by assembling various topological components. With this natural building process, the basic routine in TIL codes is called a "COMPONENT". These represent both the main program and the various subprograms (e.g. like a SUBROUTINE in FORTRAN). Like most other subprograms, variables can be imported and exported from COMPONENT's. In TIL, these variables are labels for corners, vectors, and surfaces. The actual corners, vectors, and surfaces can be specified within any component. While they are common to all components, they cannot be referenced unless a correct label is available in the calling component. The output labeling convention provides this. The call of a component entitled "name" is given by the syntax of

```
INPUT  n  name(input and output labels);
```

where n is a positive integer tag which becomes part of the new label that is used inside the component where this input appears. As the succession of components are called into action, the labeling process continues with successive labels. Thus, a particular item may be re labeled a number of times. The re labeling will stop once the main component is reached and this will then provide a unique trace back to its point of origin. As these traces can be valuable for finding topology errors, they are included in diagnostic output that is automatically given when such errors occur.

While corners, vectors, and surfaces have been mentioned above, there was no mention of their purpose. To consider the solution of some field analysis for some region, the first item considered is its boundary. These are composed of surfaces of various sorts. These may be defined by certain analytical or piece wise analytical formulas or by some digitization. These may come from CAD systems or from other forms of construction. In the case of digitization, the surface is defined by a data file. The analytic types provide some built in types together with a location for user specification for the rest. In TIL, the surface is "defined" by only one line. This may reference a data file, provide parameters for a built-in analytical type, or may link to the location of a user specified analytic type.

With the boundaries represented by the one line surface definitions, the next items are the corners. The collection of corners defines a topology sketch. This sketch appears in physical space (not an idealized abstract space). Within the physical space sketch pad, the task is to define a coarse hexahedral wire frame. Each wire frame cell defines the topology element for a block of grid. The entire wire frame is defined by a sequence of corners and should extend over most of the space. Certain of the corners must also be assigned to surfaces in order to tell the system which parts of the topology are to be on which parts of the physical region boundary. While each corner must be put somewhere in physical space, there is no precise requirement to give a certain location. The only demand is the fuzzy topological one that the placement be in "general position". In practical terms, the corners assigned to a surface should roughly follow it to reflect major variations in surface shape.

It has been seen that the basic part of a TIL program is the pattern of points as represented by a coarse wire frame of corners with links and surface assignments. Along with the organization imposed by the use of components, there are global settings that can be made at the top of the program as well as the ability to add further TIL operatives by an include statement. The typical global parameters are the number of grid cells for each link and the dimension. The default number of cells is 8 and the default dimension is 3. The number of cells is then locally changed, as desired, within the components. Any change there automatically propagates to any effected links. The include statement permits the user to grow TIL code libraries to cover problem classes of interest. This happens naturally because COMPONENT's are reusable and can be called any number of times. The result is an efficient assembly process which can take advantage of recurrent structure. In addition, there are an assortment of other commands and options that are helpful.

THE CORNER DEFINITION STATEMENT

The basic elements of a corner definition are its label, its position, its assignment to surfaces, and its links to other corners. Its label is just a positive integer which appears after the key letter "c" that starts the corner definition line. The only rule to be obeyed is that as corners are defined the labels must increase in size. Gaps in the labeling are permitted and are sometimes desirable. The gaps can be useful when one wishes to subsequently edit the component by inserting some new corners between two previous ones. For cosmetic reasons, they may even help to add human clarity to a TIL component.

After the corner label is given, the next item in the corner definition is the position in physical space. In a direct sense, this is given by a sequence of three numbers to reflect the ordering for x, y, and then z. The delimiter between each item on the line of definition is just a space: there are no comma's! While such a direct definition is often convenient, it is not as flexible as the one which comes from vector operations. The "x y z" sequence is replaced by "@ vector operations" where the symbol "@" is the flag to tell the system to expect the operations with vectors. In practical terms, this allows one to create corners that are automatically positioned relative to other corners, be they within the current component or imported. Moreover, the positioning can be even more arbitrarily accomplished by the use of vectors that are either defined in the current component or imported from the outside. While all corners are vectors, the vectors are more arbitrary since they are not required to convey the specific details associated with corners.

After the corner position is given, the next item in the corner definition is the surface assignment. If there is to be a surface assignment, then the flag "-s" is given. After a space delimiter, the desired surface labels are given with spaces between them as delimiters. The order in the list of surface labels is unimportant. Each label points to a surface definition statement which in turn provides a desired surface association. A rule that must be obeyed here is that the surface definition must already exist. There is simply no meaning if one should designate a surface that is not there! The available surfaces for assignments are those which are either defined in the current component or which are defined by an import into the current component. The latter is slightly more abstract and also more general. To provide for an import, the surface is defined in the component argument list. However, it will not really exist unless an actual surface label is inserted in that spot when the component is used. If the spot is given as a black in the component application (i.e. in the INPUT statement), then the assignment to will also be a blank. This, however, offers the opportunity to design components which can be applied in many more situations.

After the surface assignments are given, the final item is the specification of linkages to other corners. If there are to be a links to other corners, then the flag "-L" is given and is followed by a list of the corner labels with spaces for delimiters. The order in the list is again unimportant. The only requirement is that the corners must already exist by either a local definition within the same component or by a defined input to the same component. As in the case with surfaces, there is the opportunity to design more generality into the components by using blanks when there is no corner link at the INPUT stage.

The end of line for any TIL statement is given by a ";" and the corner definition is such a statement. To illustrate the corner definition syntax, an example is given. Consider the statement

```
c 5 1.2 3 4.5   -s 1 3   -L 3 4 2;
```

This defines corner 5 to be located at the position (1.2, 3, 4.5), to be assigned to surfaces 1 and 3, and to be linked to corners 3, 4, and 2. Notice that there is no detail about how the links to other corners are made. For simplicity, they can be considered as straight lines. This lack of detail is what distills the wire frame down to the level of just a sequence of corners. Also, one may notice that there is no detail given for the surface assignments. The surface labels point to the surface definitions which are single lines. A block face will be placed upon a particular surface if the corresponding four topological face corners are assigned to that surface. Moreover, a block edge will be placed upon the intersection of two surfaces if the two topological corners for the edge link are assigned to the two surfaces. This is how the user gets the intersections between surfaces.

SURFACE DEFINITION STATEMENTS

The basic parts of a surface definition statement are the label, the type, the parameters or file name, the orientation, the cluster intensity, and transformations. Spaces are the delimitors between the various parts and associated parameters. The surface definition line starts with the key letter "s", has the delimitor of one or more spaces, and is followed by the label. The surface labels are positive numbers given in an increasing order. Gaps in the numbering are allowed and can be desireable for certain occasions. After the surface number, the definition type is given and is followed by its parameters or the associated file name, as is required of the specifed type. Beyond this are the other items. The orientation is given by the direction of the unit normal vector to the surface. The rule is that the normal vector must point into the region to be gridded. As there are two sides to any bounding surface and as each such surface is given a natural normal direction, there is only the need to reverse the orientation should the natural normal vector violate the rule. Orientation reversal is given by the flag "-o"

With the properly oriented bounding surfaces, the next items are to cluster to that surface or to transform it. The flag for clustering is currently given by either "-c" or "+c" to denote minus and plus cluster groups, respectively. The actual clustering for either group is given as a real number which follows the flag after a space or so. The transformation flag is "-t" or "-R" for a directly stated translation or linear transformation, respectively. Both can be used in the same statement. The translation is given by a succession of three real numbers while the linear transformation is given by a succession of nine real numbers. However, when vectors are used both are combined into one item which is expressed as "-t @( )". Within ( ) a single vector is inserted if it is only a translation and four vectors are inserted if it is a general transformation. In the general case, the first vector is the translation while the remaining three are the respective rows of the 3x3 linear transformation. Because vectors can be defined elsewhere, this is a variable assignment. Aside from these options attached to the surface definition, there are a few remaining technical options which are not frequently used.

To illustrate the surface definition syntax, a few examples are given. First consider the statement

```
s 2 -plane(0 0 1 -2) ;
```

This defines surface 2 as the plane $z=2$ with orientation in the positive z direction. The type is "-plane". The first three numbers represent the normal vector (0, 0, 1) while the last gives the constant value of 2. The ";" gives the end of line. Because of the orientation, the grid is assummed to be above the plane.. The plane is defined by the function

$$f(x,y,z) = ax + by + cz + d$$

as the level surface $f(x,y,z)=0$. Level surfaces of f are those surfaces which are determined by constant values of f. Since $f=c$ is the same as $(f-c)=0$, it is always possible to replace f with f-c ; and thus, there is no loss of generality in considering only the constant values of 0. With the same constants (i.e. 0), the expression alone then carries the total definition of the surface.

As a second example, consider the statement

```
s 5 -ellip(0.5 2 0.25 3.2) -o -t 0 0 4 ;
```

This defines surface 5 to be of (super-) ellipsoidal type. The first 3 parameters are the recipricals of the semi-axis lengths while the fourth is the exponent which determines the degree of squareness. Thus, the semi-axial lengths are 2 along the x-axis, 0.5 along the y-axis, and 4 along the z-axis. The squareness of 3.2 is greater than that of 2 which would define a pure ellipsoid. The natural orientation which points to the outside of the super-ellipsoid is reversed with "-o" in order to consider a grid on the inside. With the basic "-ellip" type being defined about the origin, it is lifted up by 4 units in the z direction with the translation vector (1, 0, 4). For surface 5, the parameters are a=0.5, b=2, c=0.25, and n=3.2 in the function

$$f(x,y,z) = |ax|^n + |by|^n + |cz|^n - 1$$

which defines the level surface. As n varies from 2 to infinity, the surface varies from a pure ellipsoid to a brick. It is also easy to see that a natural normal direction is in the outward convex direction since this is also the natural direction of the gradiant of f. It is also worth noting that the multiplicative nature of a, b, and c is quite convienient since cyclinders can also be readily specified by setting any of these parameters to zero.

SURFACE GEOMETRY INPUT

The surface geometry can enter the process in a wide variety of forms and must be distinguished from any particular form. This is especially true if the geometry is given by a point data set.. Often a misinterpretation can occur because the set appears in the format of a surface grid. The necessary distinction is that this input grid is only used as a data set to define geometry: there is no requirement to place points at any particular position for the act of grid generation. The only requirement is to represent the geometry accurately enough to permit a decent grid generation. This means that there must be an accurate enough resolution of curvature.

In the current implementation, GridPro®/az3000 can take both implicit and explicit surface data types. The implicit surfaces are the level surfaces. They are implicit since the evaluation of points on such surfaces comes from the solution of an algebraic equation f(x,y,z)=0. By contrast, points on explicitly defined surfaces are evaluated directly from their parameters. That is, for a given (u,v), the surface point is just (x(u,v), y(u,v), z(u,v)). In many instances, there are a number of sections for an explicit surface. The evaluation of a point then involves a search to find the desired section after which the direct evaluation transpires.

The built-in implicit surfaces are

(1) planes
(2) super-ellipsoids, and
(3) cylinders

In the TIL syntax, the surface definition types are denoted by "-plane ', "-ellip", and "cyclind" respectively. Examples of the first two were given in the previous section. The third type is a cyclinder in the form of a periodic surface.

As already noted, the explicit surfaces have a parametric nature since surface points are determined by local coordinates once the local surface segment is found. The current ones are

(1) A single coordinate grid
    This has bilinear coordinate parameters for each element and is brought into action with the type flag "-linear".

(2) Multiple coordinate grids
    This has bilinear coordinate parameters for each element and is brought into action with the type flag "-compos".

(3) Unstructured quadrilateral elements
    This has both a plain format and a MSC/NASTRAN format in the forms CQUAD and CQUADR. It is brought into action with the type flag "-quad".

(4) Unstructured triangular elements
    This has both a plain format and a MSC/NASTRAN format in the forms of GRID, CTRIA3, and CTRIAR. It is brought into action with the type flag "-tria".

(5) Surface of revolution around an arbitrary center curve

This is given by a data file which lists a sequence of ordered quadruplets (x,y,z,r) where the each (x,y,z) is a centerline point and r is the radius of the circular disk normal to the centerline point. The first line in the file is the total number of quadruplets which is just a positive integer. This type is brought into action with the flag "-tube".

Unlike the implicit types, the explicit ones require data files for the surfaces. After the type indicator, each data file is given between quotes. Without extra options, these appear in the surface definition statements of TIL in the format

```
s  n   -type  "name.dat" ;
```

for surface number n. Here -type is any of -linear, -compos, -quad, -tria, or -tube as described in the above list. After the type designation flag, the user chooses "name" for the corresponding data file "name.dat".

The built-in implicit types are very convenient to apply since surface definitions are given by a small number of defining parameters. While additions to the list of built-in implicit types will add further convenience, there will always be cases that are not covered. Thus, provision has been given to user defined implicit types. The window of opportunity for non-built-in type appears in the form of a C-code that is supplied to the user. For a fixed surface, the user simply writes out the equation for the level surface desired. This writing is done in only one specific location. In the case where periodicity conditions are necessary, more locations must be used to prescribe those conditions. All of this data is inserted into a file called "name.h" where the user chooses "name" and inserts his data in the right spots. After the type flag (-implic) in the surface definition line, the user gives "name.h" and continues with other items to complete the line.

In a slightly more abstract sense, a floating surface can also be defined. It is abstract since there is no defined position or condition for this surface. This type of surface is used to define clustering to any block face regardless of where it is. This clustering can be applied to either side of the surface or to both sides. To bring this into action, the flag "-float" is used in the surface definition line; next, there is a cluster group flag (either "-c" or "+c"); and then there is the desired spacing for the first grid point on either side. Once the surface definition line exists, the clustering takes effect when the corners of a chosen block face are assigned to this surface. If clustering is desired for only one side of a floating surface, then the sides are selected by inserting a number 0 or 1 just after the type flag "-float".

Altogether, we have 3 modes of surface prescription. The first is a surface that has a fixed position in space. The second is the periodic surface which seeks its own position in space to optimize grid quality while accommodating periodic boundary conditions. The third is the floating surface which has no conditions other than the clustering specifications attached to its definition.

Unlike other schemes, there is no dependence upon the details of surface geometry definition. The geometry is simply viewed for its trace of points in (x,y,z) space. There is no concern for any specific surface parameterization and how it might abut another set of surface parameters. However, there is a need for the geometry to be well defined. A well defined geometry is one which gives an accurate specification of the solid objects. It must be free of constructive irregularities such as unexpected holes which would make hollow objects out of ones that should be solid. In additon, if two independently defined surfaces are to intersect, then they must actually do so in the region of interest. Otherwise, one is left with holes that ill define a solid object. A further desire is for the geometry definition to be compatible with the chosen pattern of grid points. This situation can occur if adjacent grid sheets are chosen to wrap over a region of sharp surface curvature. In such cases, it is best to have a smooth surface even if that smoothness is only seen on a micoscopic level. An alternative is, of course, to select a different topology. Altogether, because of the freedom from detailed geometry description, virtually any form of surface definition can be addressed and built-in. This provides a healthy growth path.

THE SCHEDULE FOR RUNNING THE GRID GENERATION

Once the TIL code is finished and put into a file, the next task is to run it. This is done by creating a schedule file for the run. The schedule can vary from a straight run to one which is dynamically changed. The option to change the course of a run allows the user to optimize the run.. For example, the run could be started with a fairly modest number of grid points which are then increased in stages as the grid is relaxed towards its equilibrium that progressively satisfies the underlying variational scheme. As one could imagine, there are a number of parameters in the schedule that provide the user helpful options. One of these provides the bounds for curvature clustering. This is in effect a geometry adaptive process. Upon activation, the volumetric grid points distribute themselves about the boundaries in proportion to the boundary curvature.

FUTURE DIRECTIONS

With the component structure of TIL, there is the opportunity to make TIL components in an object oriented sense. This can quite naturally feed into an interactive environment where the topology set up can be done. Given the non-unique nature of grid topology selection, some choices are clearly better than others. These can vary with the physical problem or region geometry. This will lead to the application of expert systems to help in the guidance. In addition, there is also the consideration for a turn key creation of the coarse wire frame for topology definition. This would then automatically generate the TIL code and thus the grid. The automatic generation of the coarse wire frame can certainly be done with today's methods. Here, one simply does a coarse unstructured tetrahedral grid generation (via advancing front, octtree, etc.) and then breaks up each tetrahedron into 4 hexahedrons. While this approach will certainly define a topology through a coarse wire frame, it will also yield too many severe singularities. Thus, there is a need to do the unstructured generation with more quality. This means the automatic generation of hexahedral grids with appropriate quality and desirable pattern for the targeted applications. Examples of pertainent activity are paving (2D) and plastering (3D). Altogether, these various directions for topology generation are in the realm of research and development.

EXAMPLES

In this section, we will look at a selection of examples. These will cover a cross-section of industries and show the type of grids that are created. This will explicitly show the quality level. Beyond these pictures, it is natural to ask for the amount of computer resources that are needed as well as the length of the TIL code. These items will be given. The next question may very well be to estimate the length of human time. This is more difficult since the user must give a region geometry, create a topology and cast it into TIL. In fact, geometry specification and topology creation are a part of any multiblock grid generator. In GridPro®/az3000 the ease with which various topologies can be considered also means that the user has a more plentiful supply of choices. While each choice can be addressed with a traditional multiblock grid generator, it is so painful to do so that it is, for all practical purposes, not a real option. A typical case of this sort is the application of compact enrichment. Here, coordinate surfaces are made to stay within the local region requiring enrichment. In effect, certain surfaces are steered back to the same subregion. In the case of geometry specification, GridPro®/az3000 can accept exceedingly general surfaces as input. This includes surfaces digitized in an unstructured format (e.g. NASTRAN) as well as implicit surfaces. The use of implicit type surfaces is extremely powerful and general. An examples of it is given in the companion paper by Cheng and Eiseman [3].

The examples given here start with a blade row for turbomachinery. Then a grid is generated inside a single turbomachinery blade. This illustrates the effect of clustering to concave and convex curvature in the simple 2D setting. Next, the same sort of curvature clustering is witnessed in the 3D case of a automobile fuel tank. The fuel tank geometry was given by an unstructured triangular mesh while the fuel surface level was given by an implicit form for a plane. After this, compact enrichment is viewed with a case for an airfoil over ground. Again the simplicity of 2D makes it easy to see the concept of compact enrichment where coordinate curves are steered back to surfaces. This case also shows a branch cut off of the trailing edge. In continuation, compact enrichment is examined in 3D with a grid for the ONERA M6 wing. In distiction from the prior case, this shows the enrichment in the setting where coordinate surfaces wrap around the entire wing. The enriched locations are around the entire perimeter of the wing planform. Shifting our focus back to automotive engineering, we next examine the case of an air induction system and then a two port cylinder. This is then followed by a manifold configuration which is a case with four tubes emerging from a large chamber. Each of the tubes emerges from the chamber with an abrupt intersection. Once again, we return to compact enrichment and look at only one tube abruptly intersecting the chamber. The enrichment appears as a collar about the curve of intersection and can be easily witnessed in the figure. While the last two cases examine abrupt intersections, there are parallel cases with filletted intersections. Fillet creation and its use is examined in Cheng and Eiseman [3].

CONCLUSION

The power of multiblock grid generation with automatic zoning has be explored along with the associated program: GridPro®/az3000. The diverse nature of the options and applications have been witnessed as has the flexibility of the program. Some future directions have also be charted for the creation of still other programs the generate input for GridPro®/az3000. Altogether, the future for this technology is very bright.

REFERENCES

1. Eiseman, P. R., Cheng, Z., and Häuser, Applications of Multiblock Grid Generation With Automatic Zoning. Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Ed. by N. P. Weatherill, P. R. Eiseman, J. Häuser, and J. F. Thompson, Proceedings of the 4th International Conference held at Swansea, Wales, 6-8 April, 1994.

2. Program Development Corporation, GridPro®/az3000 User's Guide and Reference Manual, Program Development Corporation, White Plains, NY, September 1994.

3. Cheng, Z and Eiseman, P. R., Examples of Grid Generation with Implicitly Specified Surfaces Using GridPro®/az3000: Filletted Multi-Tube Configurations, this NASA Conference Proceedings, May 1995.

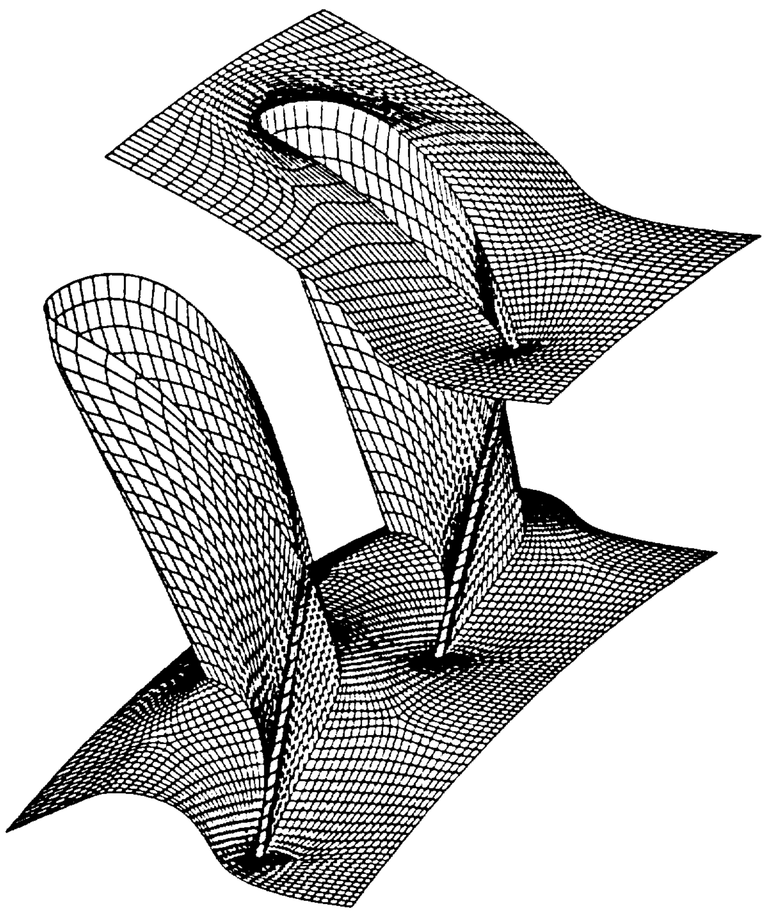


Figure 1: A row of turbomachinery blades. The choice of toplogy provides line of sight from upstream and downstream directions. That is, coordinate curves go directly from these directions onto the blade surface. The grid pattern appears as a polar like wrap for the blade which is smoothly integrated into a streamwise type grid. Periodic boundary conditions are employed around the axis. Those boundaries come from the user specification for the number of blades. Grid orthogonality and smoothness are maintained regardless of solidity: the blade spacing and angle of attack can be made arbitrarly severe without deterioration in grid quality. The TIL code is one page and the computation time is less than one hour on a medium grade workstation.

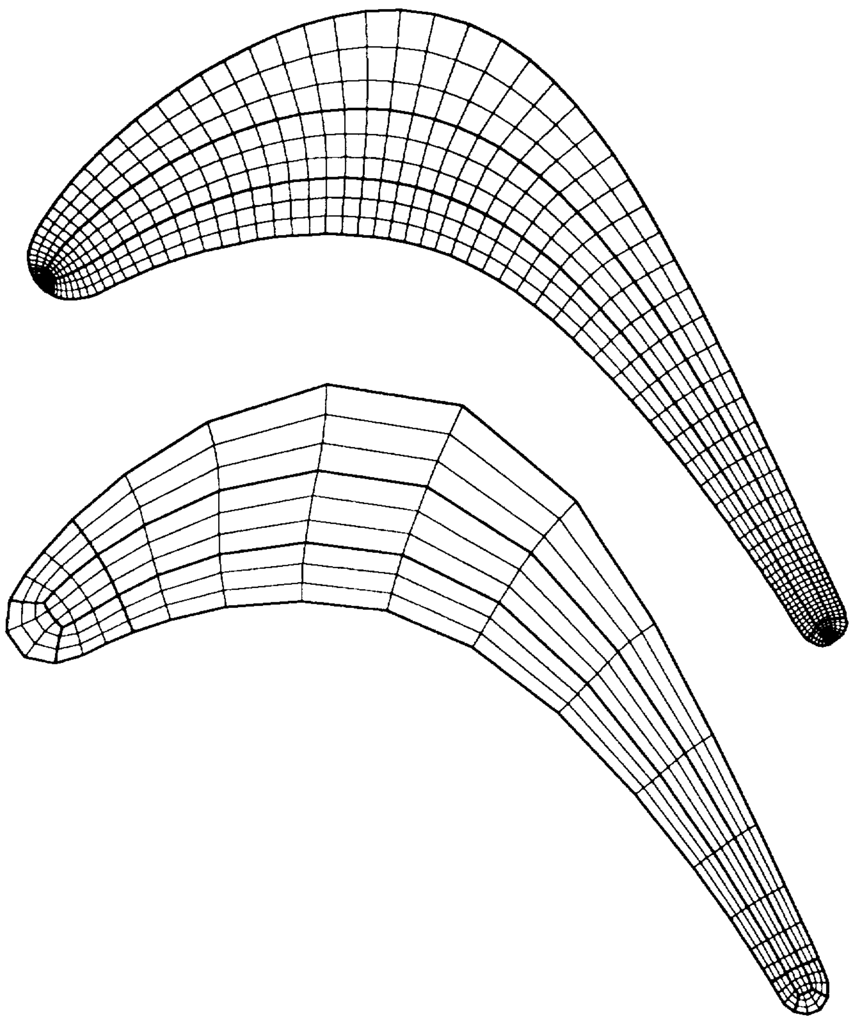


Figure 2: Inside a 2D blade contour. The effect of curvature clustering can be readily witnessed. The most intense clustering is at the blade trailing edge, the next most intense clustering is at the leading edge, and is followed by lesser clustering on the top and bottom of the blade. In contrast, the grid is thinnest at the location where the blade is nearly straight (i.e. flat). While most of the curvature is concave the bottom of the blade is the only convex part. To see how **GridPro/az3000** deals with a small number of grid points, the second grid shows the same attributes as the fine one. The TIL code is one half a page and the computation time is less than one hour on a medium grade workstation.

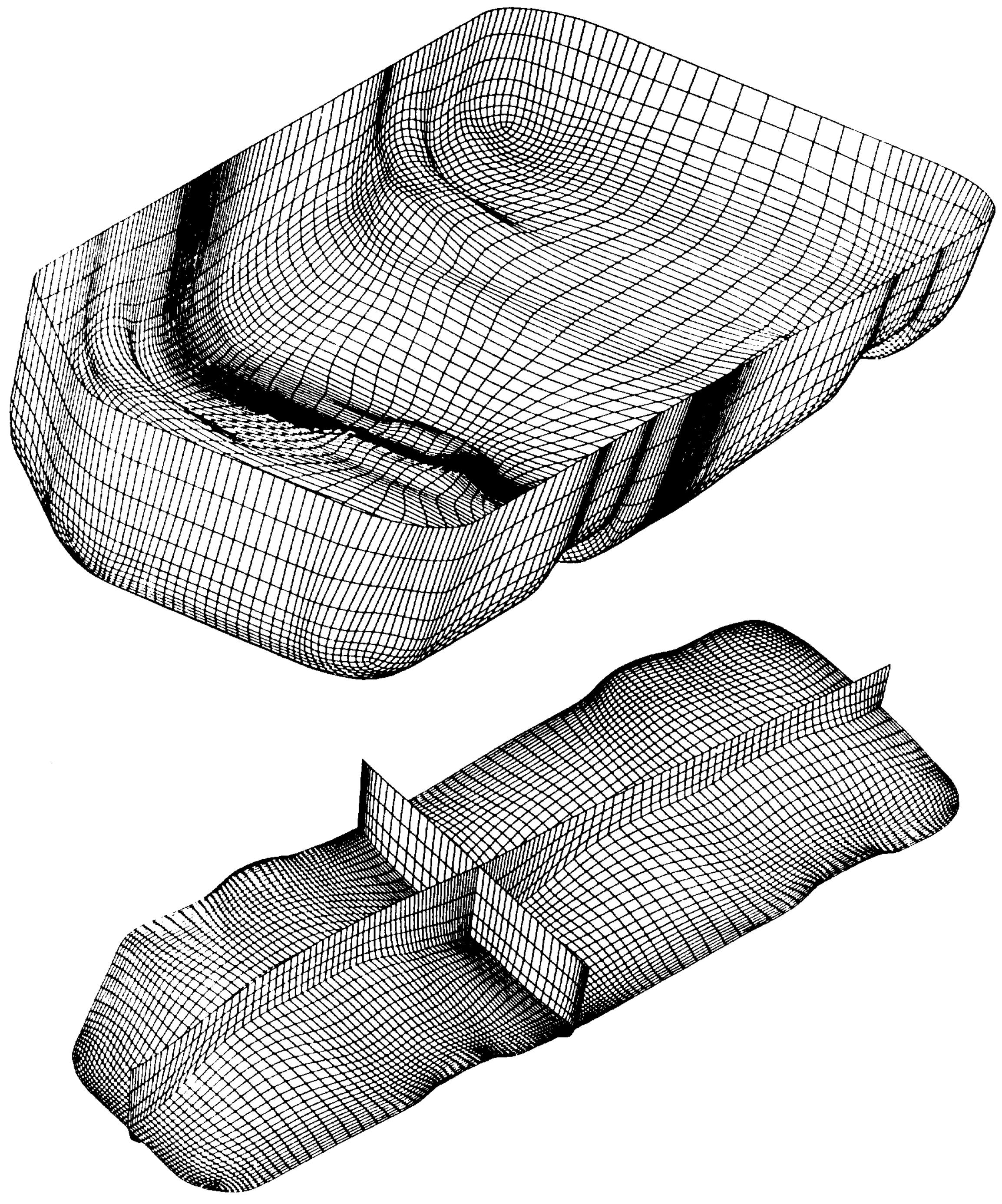
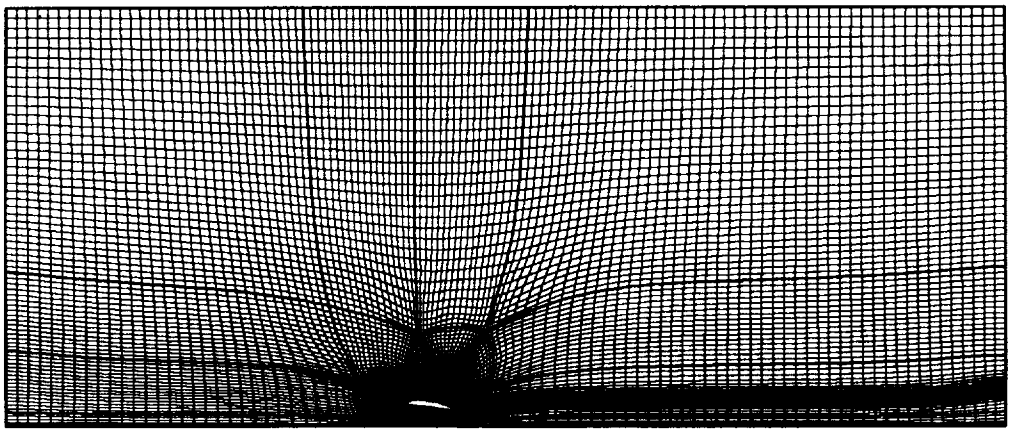


Figure 3: A half full fuel tank for an automobile. This grid is for a crash analysis application.(with LSDYNA3D). The fuel surface is given by a plane as a built-in implicit type. The fuel tank geometry is given by an unstructured triangular mesh (e.g MSC/NASTRAN type CTRIA3) and has both concave and convex curvatures. Those curvatures appear at the tank corners and at the locations of the drain and of the straps which hold it in place under the vehicle. Clustering to both concave and convex curvature is easily seen as is the smoothness and near orthogonality. Because of the application in the area of structural analysis, the multiblock output of **GridPro/az3000** was expressed in NASRTRAN format. The TIL code is less than a page and the run time is about one half an hour on a medium grade workstation.



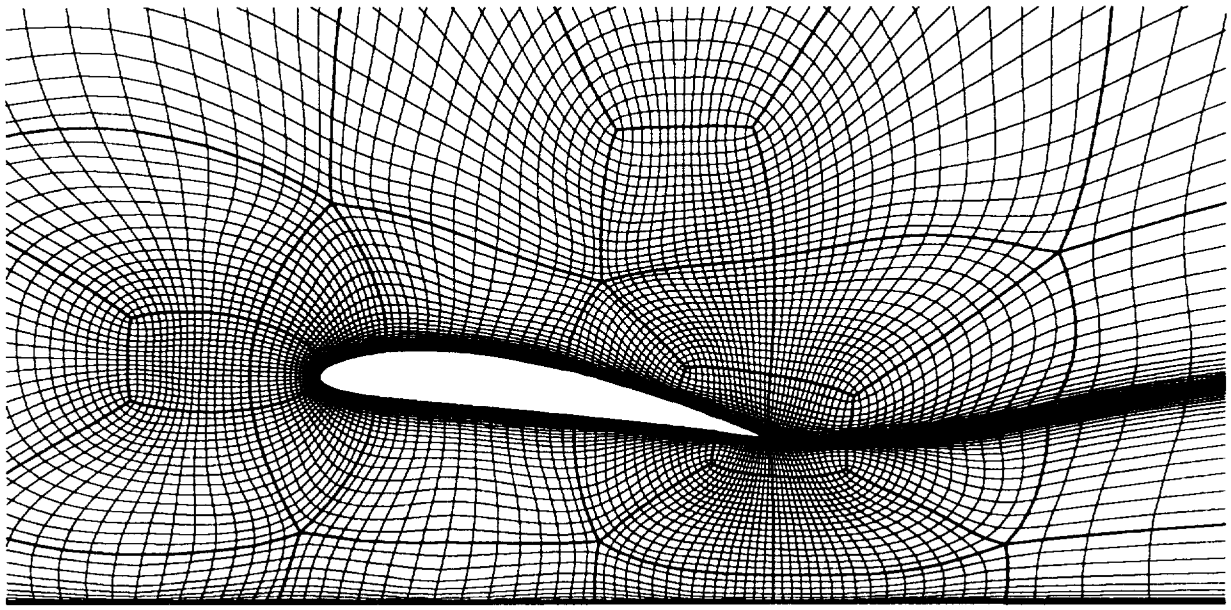


Figure 4: An airfoil close to the ground. The grid is for a study of ground effects on the perfomance of an airfoil. The airfoil contour is given by a digitized data set. This is an example of the use of compact enrichment. In the global picture, it is clear that there are no undesireable clusters of points on the boundaries either above or upstream of the airfoil. While a cluster appears for the full length of wake branch cut, it too could have been limited to some distance before the downstream boundary if desired. However, this option was not chosen. In the vicinity of both the leading and tailing edge, the coordinate curve density is at its highest. One can readily see that the chosen pattern of points (grid topology) sends curves looping from the bottom of the airfoil to the top about both leading and trailing edges. It also can be seen that similar loops appear from the top of the airfoil and end upon the ground surface. By contrast, the bottom of the airfoil was left to be that of a simple block. We often call such loop structures "clamps" or as in the case of the trailing edge structure "a collar." The TIL code for this case was a little over one page. The run time is less than one hour on a medium grade workstation.

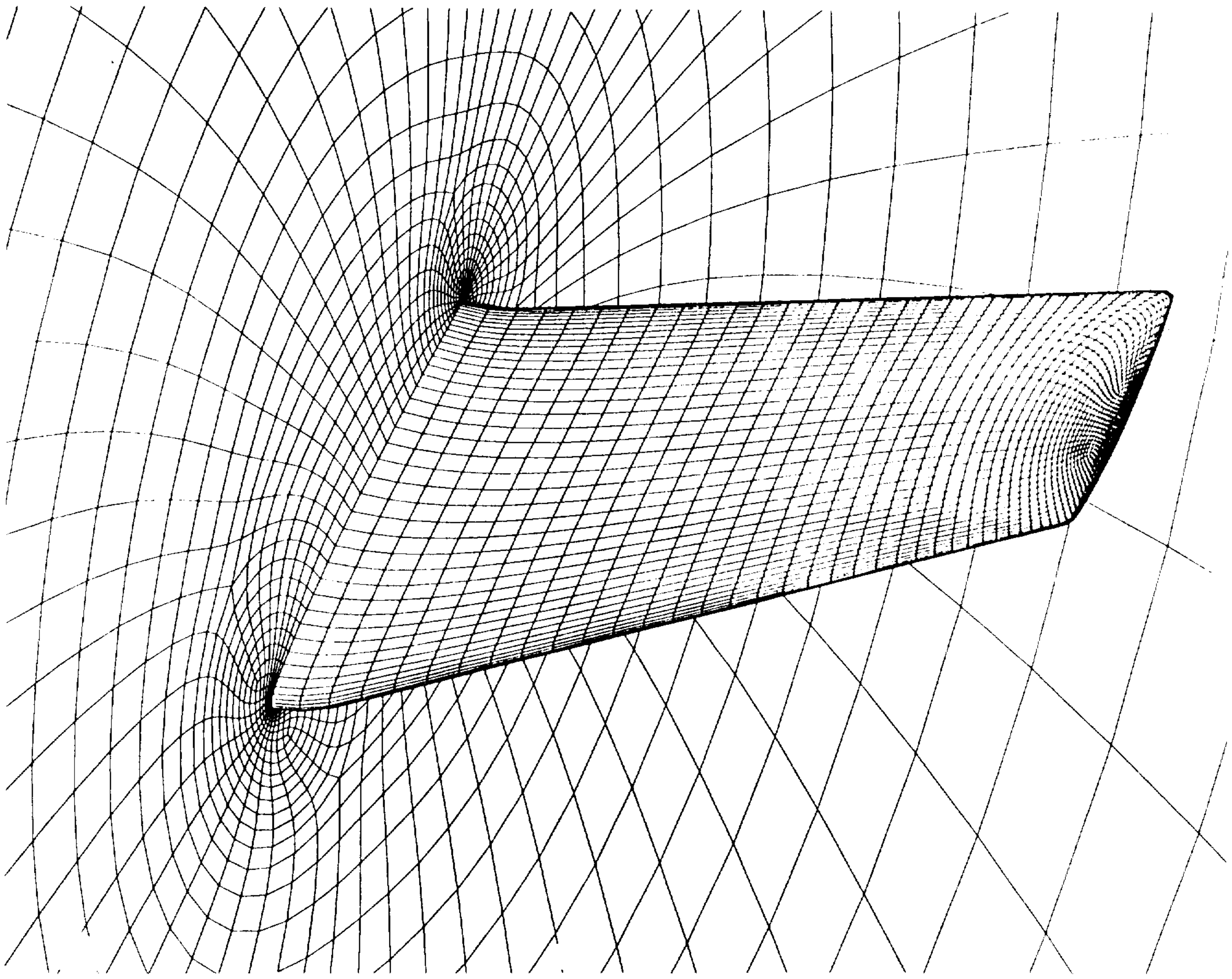


Figure 5: The ONERA M6 wing. This is a standard test case for the study of a basic wing. The basic grid topology is chosen to wrap around the whole wing. The pattern can easily be seen in the symmetry plane. Within the wrap around structure, compact enrichment has been applied to the entire perimeter of the wing planform. In the symmetry plane, it is easy to detect the compact enrichment at the leading and trailing edges. That enrichment appears in the form of curves that loop over a regular block and connect the wing surface with itself. We call this structure a "clamp." It is this compact structure which is carried from the leading edge up to and around the wing tip and then back to stop at the symmetry plane trailing edge. Altogher, the looping coordinate curves look like the shells of a distorted half torus or, more figuratively, like the surface of a half an automobile tire. The geometry of the M6 wing was given in a digitized form and the TIL code was about two pages. The grid generation was less than one half an hour.

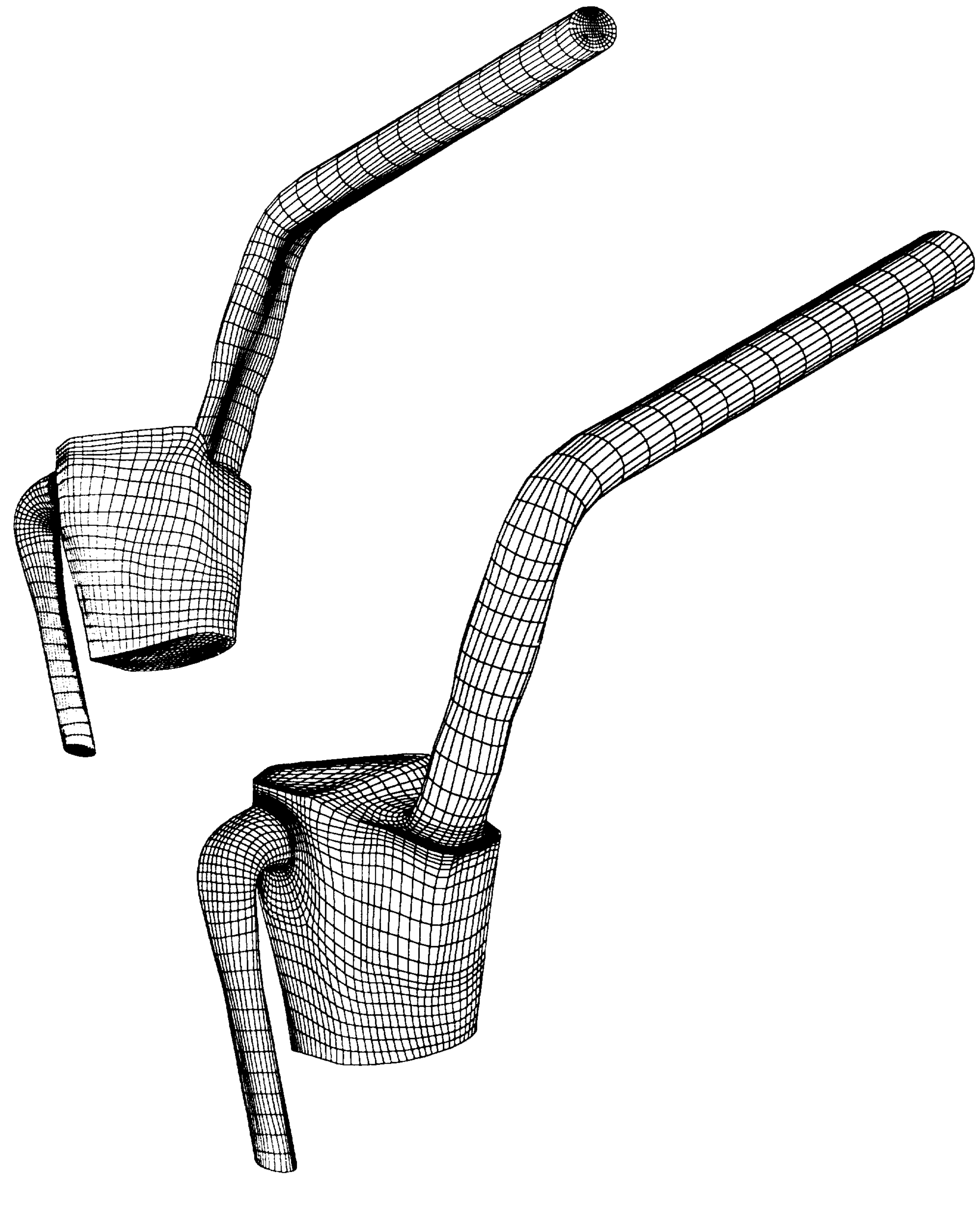


Figure 6: An air induction system for an automobile. This is an example of one sort of plumbing that is common in the automobile industry. The surface was given in by an unstructured quadrilateral mesh in MSC/NASTRAN format. The TIL code was 4 pages and the run time about 15 minutes.

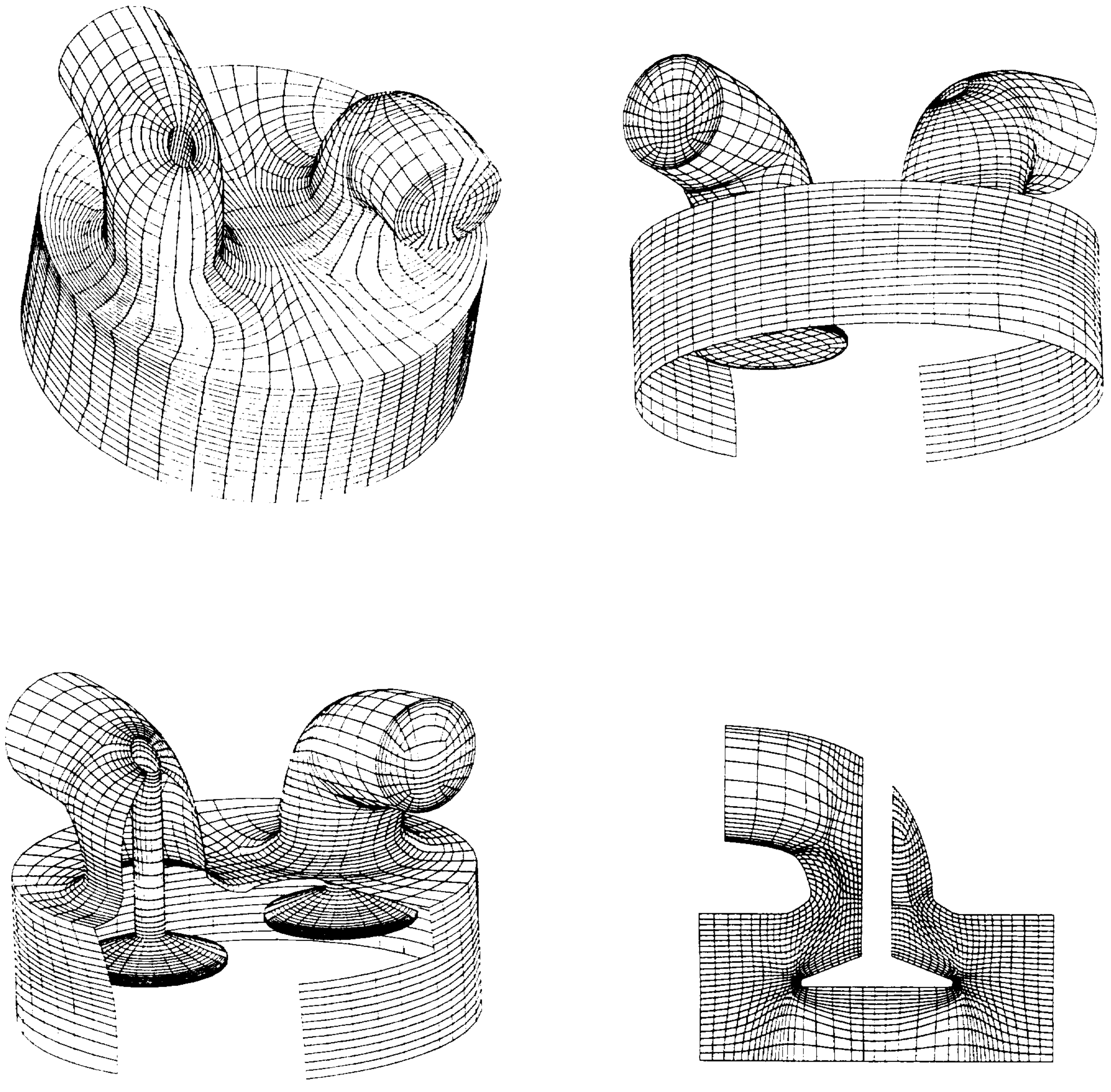


Figure 7: A two port cylinder for an automotive engine. This is a generic case for the class of two port automobile cylinders. In a prior application, there was a need to generate a grid in a pipe bend that had a rod inside it that started in a parallel manner at the entrance but continued straight up and intersected the pipe as it went through the bend. The topology for that prior case was a single page of TIL code and was in a separate file. Noting that the prior case could be also used for the port in the pictured application, the first line of the TIL code was an INCLUDE statement for this file. The remaining TIL code was just a few pages. The generation of the grid was about 10 to 15 min of computer time.

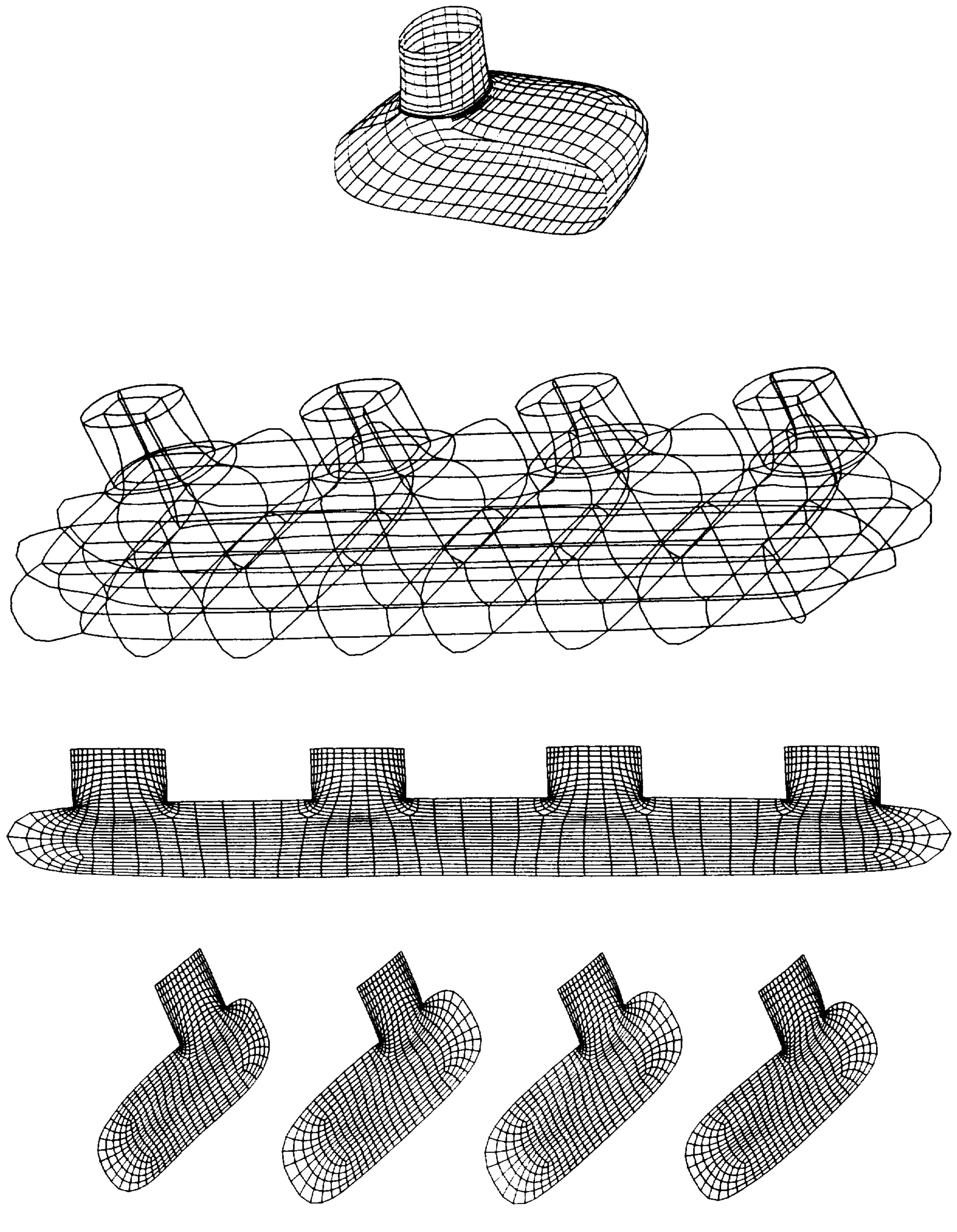


Figure 8: A generic manifold for an automobile engine. The geometry consists of four tubes which abruptly intersect an elogated chamber from above. With the freedom to choose a generic geometry, the chamber and the tubes were rapidly assembled by using implict functions of the type "`-ellip`." The TIL code was 4 pages and the grid generation was about 15 min. on a modest workstation.

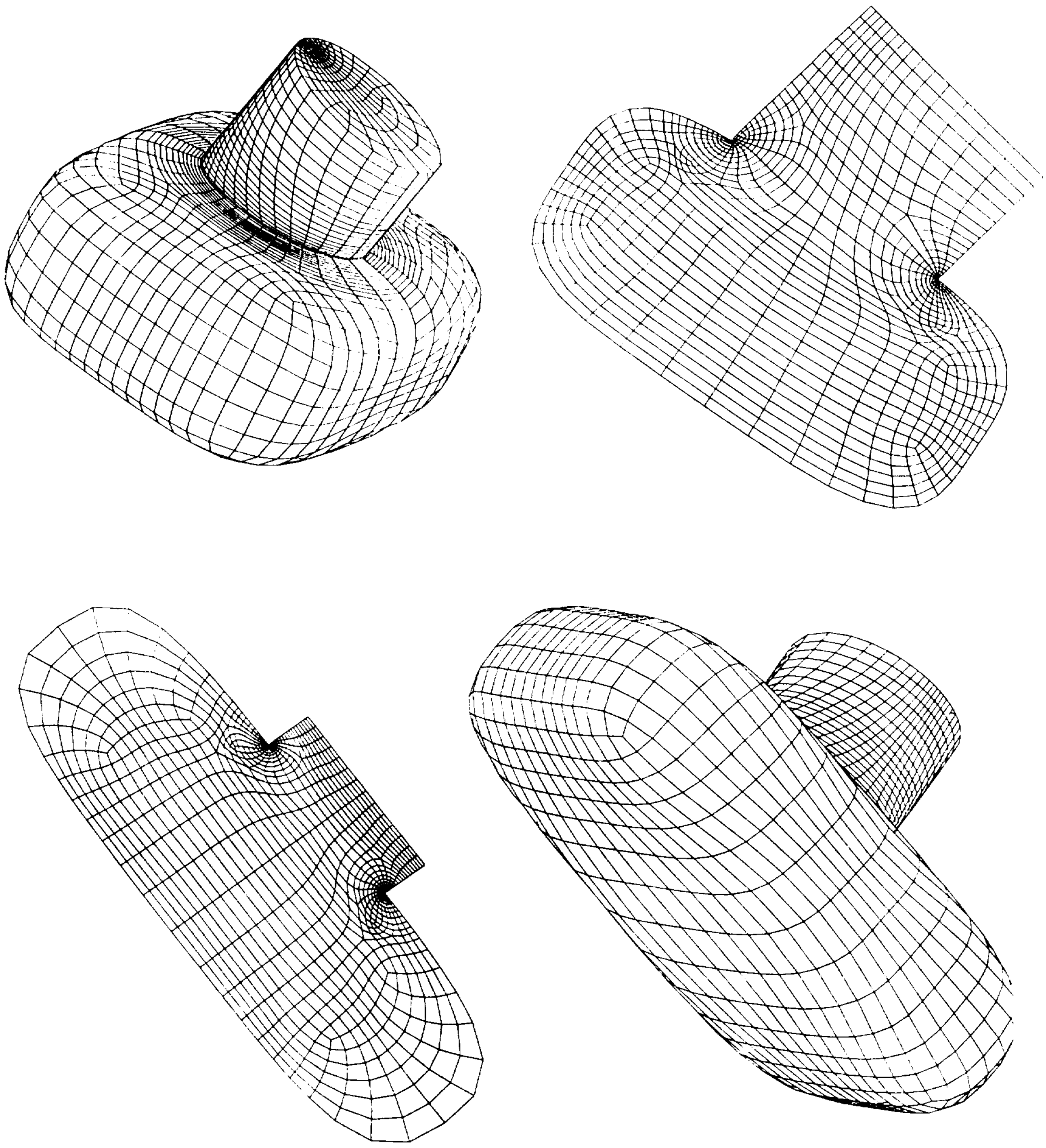


Figure 9: An abrupt intersection between a tube and a chamber. In the context of an abrupt tube intersection, the element of compact enrichment is shown. This is seen in the form of the "collar" structure which wraps around the entire curve of intesection between distinct surfaces. In addition, the associated TIL code was developed in an object oriented style. In so doing, the length of TIL code was longer than the normal few pages. However, the "objects" are very flexible in their reuse and adjustment. The main component in this TIL code was then reduced to about on fifth of a page. Major configuration changes could then be done with just a few in line alterations. In this case, such change are the addition of exta tubes into the chamber. It is this style which is the seed for a more complete object oriented library in TIL. With this type of library in place, the role of interactive graphical assembly can be empowered.

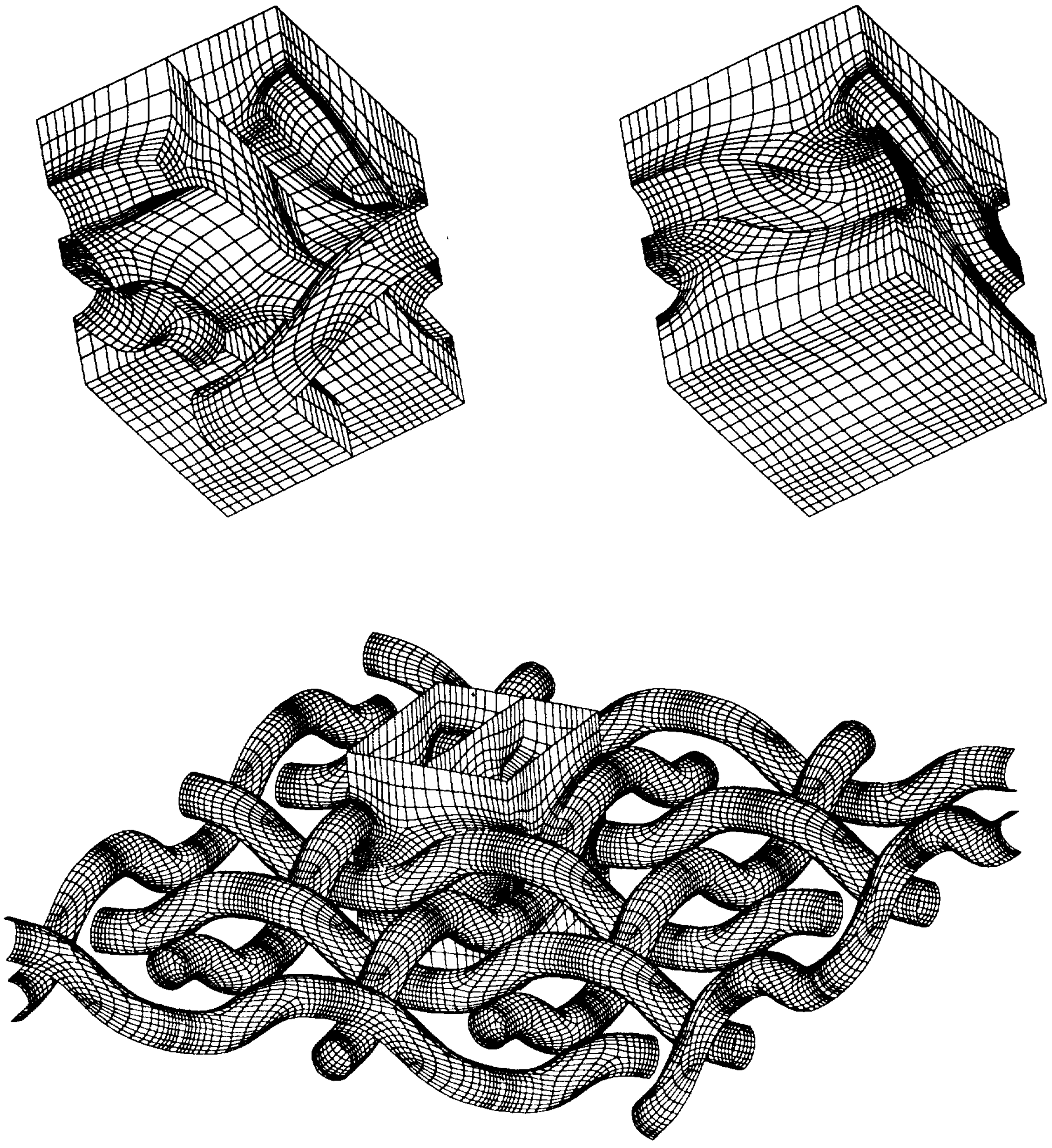


Figure 10: The field about cloth fibers. The grid is generated in the region about interwoven fibers that are bounded by planes above and below the cloth. This configuration has applications in filtration, in molding processes, and in the apparel industry. The geometry of the fibers is given by the type "-tube" with the actual tube geometry given by a data file of centerline coordinates along with cross-sectional radius. While this case was executed for fibers of a constant radius, a simple change of data file is all that is needed to do a variable radius. The grid is readily seen to be smooth and nearly orthogonal. The TIL code is two pages long and the run time on a slow workstation is about 15 min.