AN ADVANCING FRONT GRID GENERATION SYSTEM FOR 3D UNSTRUCTURED GRIDS

Lars G. Tysell
The Aeronautical Research Institute of Sweden (FFA)
Box 11021 S-161 11 Bromma, Sweden

Abstract

A program package for the generation of three-dimensional unstructured grids around complex geometries has been developed. The grid generator is based on the Advancing Front algorithm. Tetrahedra of variable size, as well as directionally stretched tetrahedra can be generated by specification of a proper background grid. The ADT (Alternating Digital Tree) data tree structure has been implemented in order to reduce execution times. The CPU time follows the asymptotic Nlog(N) behaviour of the ADT algorithm (N is the number of tetrahedra). A restart possibility has been implemented, as well as modules for smoothing and "colouring". The geometry is defined by a set of surface patches. Each patch is represented by a quadrilateral network of points. The surface patch connectivity is computed by the program. The input files to the program are these surface patches and a background grid. From this information the surface triangle grid and volume tetrahedra grid are automatically generated. The program can also be used for the generation of two-dimensional grids.

1. Introduction

This paper gives a description of all programs included in the TRITET program package. The package is used for the generation of unstructured grids around arbitrary 2/3-dimensional geometries by use of the Advancing Front method.

During the last decade, considerable amount of resources have been spent on flow simulations around complex configurations. Different types of computational grids have been used. Today, the two major approaches are patched multiblock grids and unstructured grids.

The last generation of structured multiblock grid generation programs are general purpose codes[1-5]. They rely heavily upon advanced graphics on interactive workstations. The user can build the grid step by step, inspect it and gradually improve it. But even with the most advanced program, the amount of manual input may be considerable. Thus, to generate a multiblock grid around a complex configuration will always require a person with a lot of experience on grid generation and about a month of work.

Grid generation programs for unstructured grids are normally based on Delaunay triangulation[6], or the Advancing Front method[7,8]. The major advantage of using unstructured grids is that the grid generation process can be automated to a much higher extent than for a multiblock grid. The grid generation time can be reduced from months to days. Another important advantage of using unstructured grids is the possibility to efficiently implement adaptive grid refinement. In this manner it is possible to improve the resolution of flow gradients during the flow computation. The disadvantage of using unstructured grids is that the computational time will be longer than for computations using a multiblock grid. The disadvantage of a slower grid generator and flow solver is, in our opinion, of much less importance than the advantage of a much more simple and userfriendly grid generation procedure. Unstructured grids will facilitate the flow computation around complex geomtries. It will be much easier and faster to generate grids for complex geometries and flowfields than with structured grids.

2. General Description

The TRITET program package is written in ANSI Fortran 77. The program package, which is almost complete, consists of 11 modules. The program is based on the Advancing Front algorithm. It can be used for generation of triangles for 2-dimensional grids and tetrahedra for 3-dimensional grids. The ADT[9] (Alternating Digital Tree) data tree structure has been implemented in order to reduce execution times. The CPU time follows the asymptotic Nlog(N) behaviour of the ADT algorithm (N is the number of tetrahedra). The geometry is defined by a set of surface patches. Each patch is represented by a quadrilateral network of points. The input files to the program are these surface patches and a background grid. From this information the grid is automatically generated. The Advancing Front technique is well known and we have used much the same methods as described in[7-11]. Hence, some formulae and

algorithms are not described in detail in this paper. Instead we will concentrate on our software implementation of the method.

The main features of the final 3-dimensional grid generator will be:

- The geometry is defined by a set of surface patches. Each patch should be of such a quality that it is possible to fit a spline surface to it.
- The background grid nodes are specified in an interactive session on a graphics workstation using the mouse. The size and directional stretching of the grid tetrahedra are specified at each node.
- The background grid is automatically generated by a Delaunay algorithm. This will reduce the time spent by the user giving the input.
- The edges of each surface patch are automatically subdivided into one-dimensional straight elements. Spacing is interpolated from the background grid. This will be the initial fronts for the surface grid generation.
- The surface triangulation is generated by the Advancing Front algorithm in a parametric space of each surface patch. Spacing and stretching are interplotad from the background grid.
- The volume grid is generated by a 3-dimensional Advancing Front algorithm.

2.1 Geometry definition

The geometry is defined by a set of surface patches. Each patch is represented by a quadrilateral (m×n) network of points. The points may be irregularly distributed. The only restriction is that the networks should be of such a quality that it is possible to fit a bicubic spline surface. One side of the network may be collapsed into a point. Each surface patch is stored on a separate file. The modular structure of the program package makes it easy to add a module (if decided) converting NURBS-surfaces to these spline surfaces. A CAD-interface is probably necessary anyhow in order to convert the geometry definition to surfaces suitable for CFD calculations.

3. Program Description

For the sake of simplicity and to make it easy to exchange modules in the future, each step in the grid generation process is represented by a separate program. In this section all programs currently included in the program package will be described in detail. The programs are linked together with Unix scripts.

Figure 1 shows a simple example of a 2-dimensional grid. The background grid is composed of five nodes only, one at each corner of the outer boundary and one in the centre. The boundary is composed of eight spline curves. The grid on the boundary is computed by program DIVEDG, then the grid in the interior is computed by program ADVFRO. We see

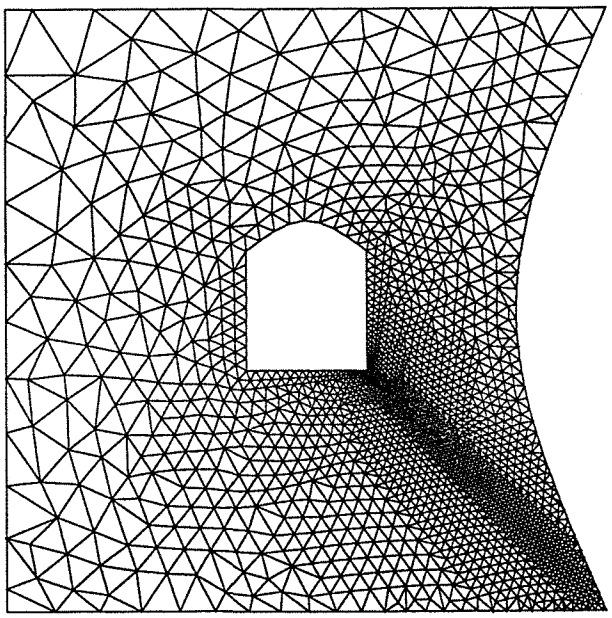

Figure 1: **Simple example of a grid.**

that we have a nice transition in triangle size from the boundary into the interior of the grid.

3.1 Program ADVFRO

This program generates tetrahedra for 3-dimensional grids and triangles for 2-dimensional grids. We will describe only the 3-dimensional grid generator. The algorithm works in a very similar way for 2-dimensional grids. The program is based on the Advancing Front algorithm. There are three input files, one for the background grid, one for the triangulated boundary grid, and also a file specifying some input parameters. The background grid is a coarse grid of tetrahedra used in order to specify the tetrahedron size and stretching of the grid to be generated. The transformation matrix $\mathbf{M_3}$ (symmetric 3×3) of the grid to be generated is specified at each node in the background grid. The transformation matrix depends on the desired tetrahedron size and directional stretching. The formula to calculate the transformation matrix is given in[8]. A tetrahedron with unitary edges in the transformed (normalized) space will have the specified size and strecthing in the physical space. The data structure of the program is based on eight lists. They are given below:

1) Store the coordinates for all nodes in the background grid.

2) Store the four node numbers for all tetrahedra in the background grid.

3) Store all tetrahedra in the background grid in an Alternating Digital Tree (ADT). A detailed

description of the ADT data structure is given in[9].

4) Store the coordinates for all nodes in the grid.

5) Store the node numbers of the face, and the tetrahedra numbers adjacent to the face, for all faces in the front.

6) Store all faces in the front in an ADT data structure.

7) Store all faces in the front in a heap list depending on their size. A detailed description of the heap algoritm is given in[10].

8) Store all faces in the front which were not able to be the base for a new tetrahedron. This is called the list of bad front faces.

The Advancing Front grid generation technique implemented in this program consists of the following steps:

1.1) Store the background grid in an ADT data structure. This will make it easy and fast to search for the proper background tetrahedron. The time to search for a tetrahedron located in a certain region will be propotional to log(N), where N is the number of tetrahedra stored in the tree. Before the tetrahedron is stored in the tree the node coordinates have been converted to integers.

1.2) Store the faces in the initial front (the boundary) in an ADT data structure.

1.3) Store the faces in the initial front in a heap list depending on their size. The size of a face is taken to be the height of the tetrahedron to be generated with the face as the base face.

2.1) Select, from the heap list, the front face with the smallest size. This face will be the base face for the next tetrahedron to be generated. Calculate the centre point of the new base face. The face is deleted from the heap list.

2.2) Select, from the ADT data structure, the background tetrahedron inside which the centre point of the base face is located. Interpolate, by linear interpolation, the transformation matrix at the centre point from the values at the nodes of the tetrahedron. This is called the **current transformation matrix**. A detailed description of the linear interpolation is given in[11].

3.1) Use the current transformation matrix to transform the nodes and centre point of the base face to the normalized space.

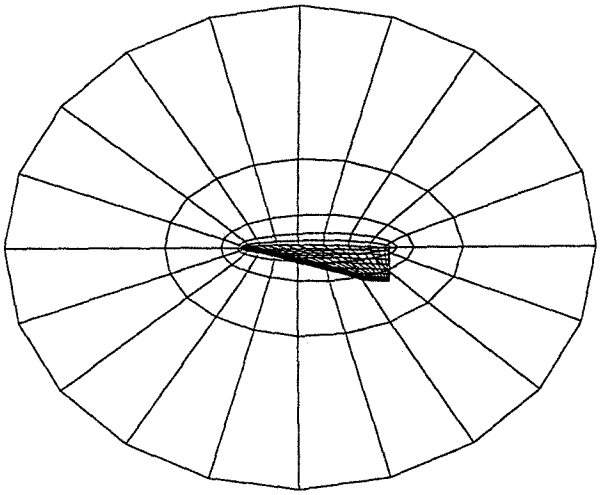

Figure 2: **Geometry for a delta wing.**

3.2) Calculate the ideal position in the normalized space of the top point used to form a new tetrahedron. Calulate also the posititions of some help points. Transform these points back to the physical space.

3.3) Calculate, in the normalized space, the minimum radius of a sphere with centre in the ideal point enclosing the base face. Enlarge this radius with a factor > 1.3.

3.4) Transform this sphere back to the physical space. Calculate a box enclosing this transformed sphere. This box is called the **search region**.

4.1) Collect, from the ADT data structure, the front faces (partly) inside the search region. Transform the nodes for these faces to the normalized space. Collect also, from this list of front faces, the nodes that are inside the search region.

4.2) Order the collected front nodes, the ideal point and the help points in a list depending on how regular tetrahedra they will form.

5.1) Select the best point (the point with the smallest **order factor**) from this list and create the new tetrahedron. If there was no point in the list, store the base face in the list of bad front faces, go back to step 2.1 and selelect a new face.

5.2) Check, in the normalized space, if the new tetrahedron intersects or nearly intersects any of the collected front faces. If it does, go back to step 5.1 and select a new point. We use the technique with covariant and contravariant coordinates to check for intersection between the faces

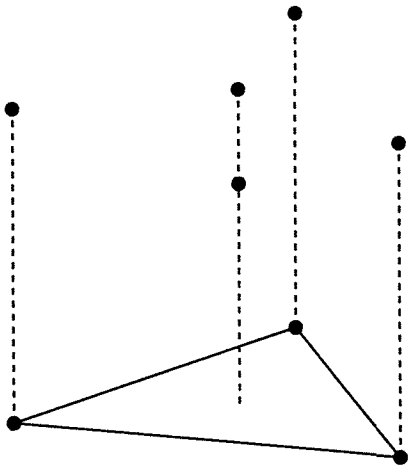

Figure 3: **Ideal point and help points.**

of the tetrahedron and the faces in the front. A detailed description of this is given in[7]. We use a restrictive condition, we consider it as intersection even if they are some distance away (<0.3, input in parameter file) from each other.

6.1 Calculate the size for the new front faces using the current transformation matrix.

6.2 Add the new node, tetrahedron and front faces to their respective lists.

6.3 Delete the faces, that are no longer front faces, from the lists of front faces.

6.4 If there are any face left in the list of front faces, go back to step 2.1

3.1.1 Ideal and Help Points

The ideal point is placed above the centre point (on the normal to the base face passing through the centre point) at the distance δ caclulated by the formula:

$$\delta = 0.83 min(1.8b, max(0.5b, 1.0))$$
$$b = \frac{1}{3}(b_{12} + b_{13} + b_{23})$$

where b_{ij} = distance between nodes i, j of the base face; $i, j = 1, 2, 3$.

Help points are placed at the distance 0.7δ above the centre point and at distance δ above each of the three nodes, see figure 3. We do not have help points closer to the base face, as in[8]. Our experience is that this might cause ever descreasing front faces in difficult regions. This will eventually lead to a hopeless situation, see figure 4 showing all faces around a hole.

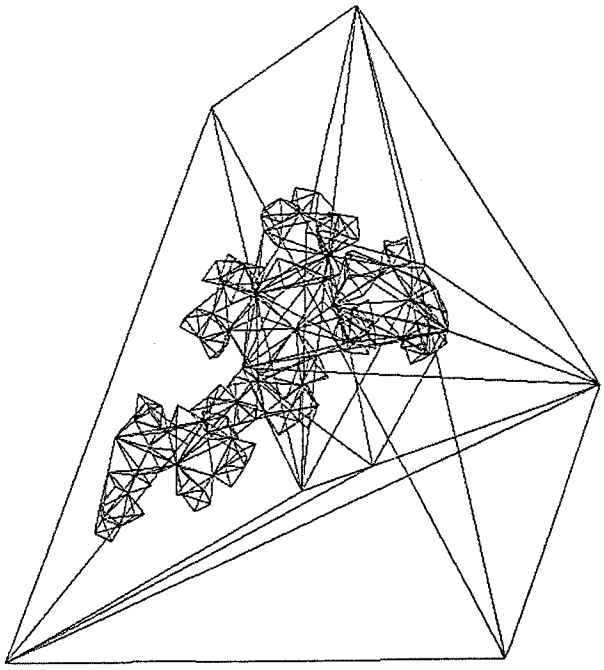

Figure 4: **Decreasing faces around a hole.**

3.1.2 Ordering of Points

The collected front nodes, the ideal point, and the help points are ordered in a list, se step 4.2 above. The order factor for the ideal point and the help point at 0.7δ is 1.0 and >>1.0 respectively. The order factor for the three node help points is 0.99 if $b < 0.7$, otherwise 1.01. Hence, the three node help points are selected before the ideal point if a rapid expansion of the grid cell sizes is required. We believe this will facilitate a rapid expansion of the grid cell size. An order factor is also calculated for the collected old front nodes, depending on how regular tetrahedra they will form. If the tetrahedron is of good quality the order factor is < 1.0. Several criteria have been proposed, see e.g.[8,11]. We use a criteria taking into account both the skewness of the tetrahedron and the distance between the centre point and the top point. As long as the projection of the top point is inside the bace face, the tetrahedron is considered to have no skewness. However wee do not think the criteria is that critical, as long as it is reasonable. It is probably more important how the check in step 5.2 above is done. The faces of the new tetrahedron must not be too close to the old faces in the front. Otherwise this will cause trouble later on. Creation of one tetrahedron of low quality is acceptable as long as the further grid generation is not hampered.

3.2 Program RESTRT

In some regions of the grid it may not be possible to generate grid elements (tetrahedra/triangles) of good quality. This is particularly true for 3-

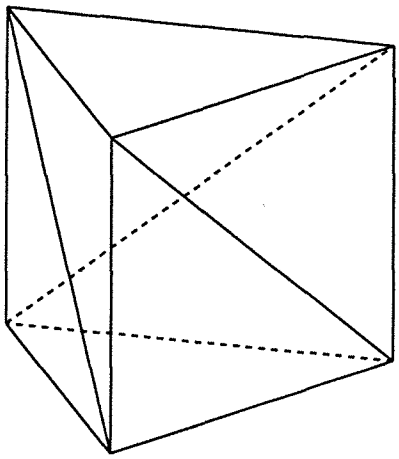

Figure 5: **Generic hole in the grid.**

dimensional grids. Hence, holes are left in the grid, see step 5.1 in the description of program ADVFRO. Figure 5 shows a situation where the hole can not be filled with terahedra without adding a new node inside the hole. This is because the diagonals on the three vertical sides are unsuitably oriented. Note that this is not caused by a poor algorithm. The problem is inherent in the Advancing Front grid generation concept. It is worth to stress that this situation has no counterpart in a 2-dimensional space. When problem arise in the grid generation process it is often a variation of this theme. As explained in section 3.1.1, adding a new node may be the solution, but it may also cause more problem later on.

Another way to handle this problem is to delete the layer of tetrahedra around the hole, and then restart the grid generation from this new front. The restart program RESTRT locates the holes and deletes the layer of elements around the holes, in order to create a new front.
The procedure consists of the following steps:

1) Mark the faces that are still in the front.

2) Mark the nodes located on these front faces.

3) Mark the faces that have 1, 2 or 3 front nodes, (user defined).

4) Delete the tetrahedra having a face marked in step 3.

5) Renumber the nodes, faces and tetrahedra.

The grid generator is then used once again with this new front as input. A random parameter can be set by the user effecting the order in which the faces are selected and/or the order factor of the points. Thus,since the grid generation process is not exactly repeated this time the same situation will hopefully

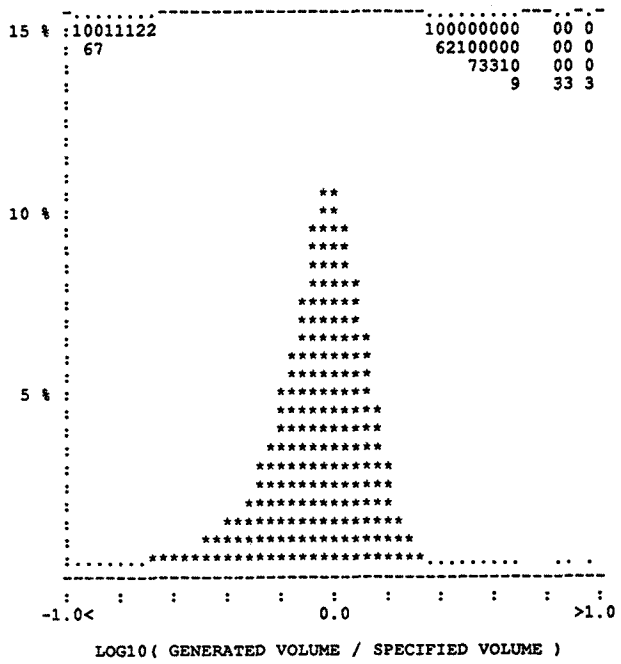

Figure 6: **Grid quality statistics.**

not occur and the holes will be filled. If some holes still remain in the grid the restart program must be used once again. Usually this has to be repeated a limited number of times.

A more radical solution may be to swap some of the edges around the holes. Then it must be checked that the new faces do not intersect the other faces. We will write a program for this purpose.

Figure 2 shows the geometry and symmetry plane for a delta wing. The grid generated around this wing consits of 322.000 tetrahedra and 650.000 faces. After the first run 4.600 faces still remain in the front. This corresponds to approximately 460 holes. Grid quality statistics for this grid is shown in figure 6.

3.3 Program SMOOTH

This program is used in order to smooth the grid generated by the grid generator. Each node is updated iteratively. The well-known Laplace smoother defined by the formula:

$$\vec{X}^{n+1} = \vec{X}^n + \frac{C_{rel}}{N} \sum_{k=1}^{N} (\vec{X}_k^n - \vec{X}^n)$$

is used, where N is the number of nodes surrounding the node $\vec{X}$ to be smoothed. C_{rel} is a relaxation factor. The node is updated from step n to step $n+1$ only if:

$$min(\Delta_1^{n+1}, ..., \Delta_M^{n+1}) > min(\Delta_1^n, ..., \Delta_M^n)$$

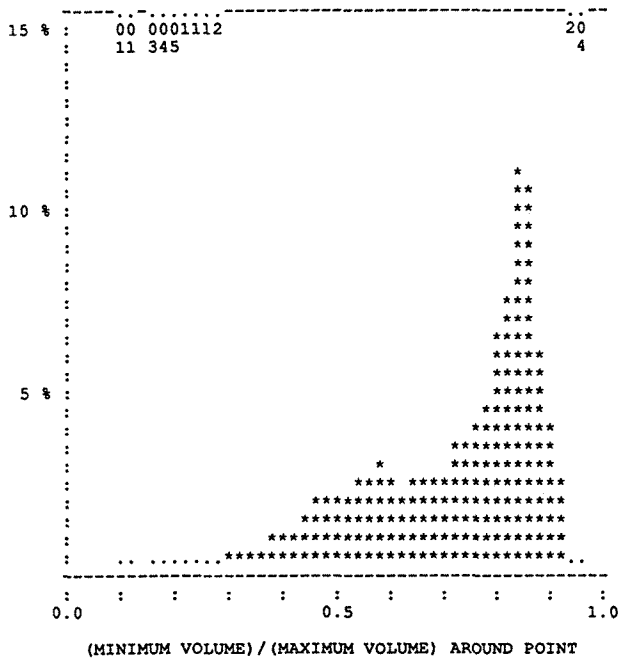

Figure 7: **Grid quality before smoothing.**

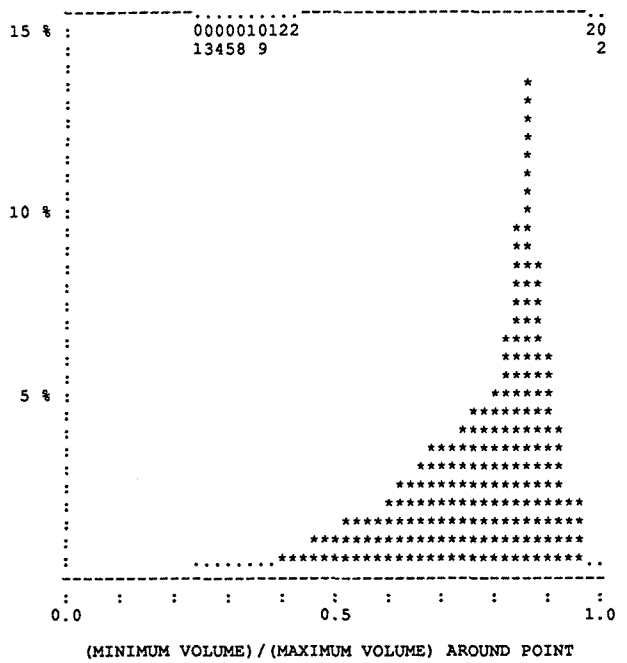

Figure 8: **Grid quality after smoothing.**

where Δ_j is the volume of tetrahedron j, and M is the number of tetrahedra surrounding the node $\vec{X}$. Hence, the grid can not be inverted. Since each node must be updated immeditely it is not possible to vectorize the algorithm. The measure we use here is the ratio:

$$\text{Quality} = \frac{min(\Delta_1, ..., \Delta_M)}{max(\Delta_1, ..., \Delta_M)}$$

By using this measure we do not need the background grid, since it is not necessary to normalize the size of the tetrahedra. Figure 7 and figure 8 show the quality statistics for the wing profile grid shown in figure 14, before and after smoothing respectively.

3.4 Program COLOUR

Our finite volym flow solver[12] use the colouring techique in order to vectorize/parallelize the algorithm. The program described here is used in order to compute the colouring of the grid. That means that the edges and faces of the tetrahedra (edges of the triangles for 2-dimensional grids) in the grid are collected into different groups. Two different colourings are computed, one for the convective terms and one for the dissipative terms. The colouring pattern for the convective terms is computed by the following simple procedure:

1) For each face store the top nodes in the two adjacent tetrahedra.

2) Invert the list above. For each node store the surrounding faces.

3) For each node:
   Walk through the surrounding faces. For each face check if a colour already has been selected, if not - select a colour that not has been used for any face surrounding this node.

Despite the simple scheme, the number of colours is very close to the minimum possible value (the maximum number of faces surrounding any point). The colouring pattern for the dissipative terms is computed by a similar procedure.

3.5 Program NUMTET

This program is used in order to estimate the number of tetrahedra (triangles) generated by the Advancing Front grid generator. The input to this program is the backround grid only. It is assumed that the background grid completly but not excessively covers the region where the grid has to be generated. The transformation matrix at the centre of each tetrahedron is interpolated from the values at the nodes. The size of the grid tetrahedra to be generated inside each background tetrahedron is estimated from this transformation matrix. The number of tetrahedra inside the background tetrahedron is taken to be the volume of the background tetrahedron divided by the estimated tetrahedron size. The number of tetrahedra in every background tetrahedron is then summed up to get the total number of tetrahedra.

3.6 Program EXTEDG

So far all programs described can be used in both two and three dimensions. However, the first step in the generation of a 3-dimensional grid is to generate the grid on the boundary surfaces.

This program reads all surface patches from the files and extract the boundary curves. The boundary curves are represented by a set of points. Since the surface patches may not match exactly, the program accepts a tolerance specified by the user. Only one patch on each object need to have the outward normal pointing in the right direction. The other patches are reversed automatically if necessary. Figure 9 shows a simple surface composed of 7 surface patches. All corners of the patches do not match. One edge of a patch may be degenerated to a point.

The boundary curves are extracted by the following procedure:

1) Extract all boundary curves. Store them in a list. The curves are represented by the points on the edges of the networks defining the surface patches

2) If an endpoint of one boundary curve is, within a certain tolerance, the same as an endpoint in another (or the same) boundary curve - make them identical.

3) Check if the endpoint of one curve is, within a certain tolerance, located on another curve. If it is, divide the latter curve into two curves. The new endpoints for the two curves are copied from the endpoint of the "intersecting" curve. Continue this process until no endpoint is located on any other curve.

4) a) If more than two curves have the same two endpoints - divide one of the curves into two curves and go back to step 3.
b) If two curves have the same two endpoints - make the curves identical. Thus, copy the points defining the curve from one of the curves into the other one in the reversed order.

5) Store the new boundary curves on files, one file for each patch.

3.7 Program DIVEDG

This program reads the boundary curve file for each surface patch. By use of the background grid the boundary curves are divided into line segments of proper size. In the first step the boundary curve is approximated by straight lines through the points. There are two copies of each boundary curve, belonging to two adjacent patches. Since the points in these two boundary curves are running in the opposite direction, the division of the boundary curve into line segments must be independent of this direction, otherwise the grid points in the two adjacent surface patches will not match. This is accomplished by taken as the start point the endpoint with the minimum x/y/z-value. If the endpoints are the same, check the points next to the endpoints. Finally, in a second step, this distribution is applied to a cubic spline through the original boundary curve points. The line segments are written to a new boundary curve file.

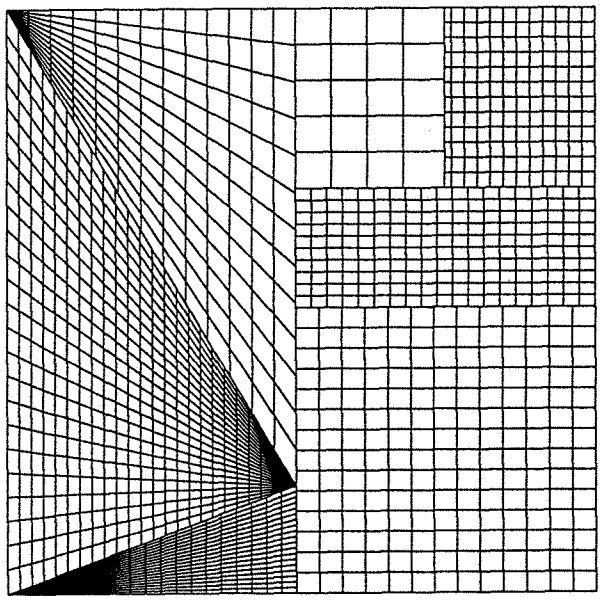

Figure 9: **Surface composed of 7 patches.**

3.8 Program MAPSRF

This program maps a surface patch and its boundary curves to a 2-dimensional quadrilateral with straight edges. The shape of the patch and the length relations between the four/three edges of the patch are kept as much as possible. The reason is that the size and stretching of the triangles in the mapped space will then not be so distorded. This will make it easier to produce a grid of good quality. The mapping is good if it is possible to flatten the surface patch without excessive strecthing. The techique used is similar to the mapping in[7].

The steps involved in the mapping of the surface patch are:

1) Calculate the arc-length of the four edges in the 3-dimensional space.

2) Rotate the patch so that the shortest edge will be the top edge in the 2-dimensional $(x' - y')$

space. The other edges are called bottom, left and right.

3) Calculate the arc length distribution along the edges in the 3-dimensional space.

4) Place the lower-left corner at (0,0) in the 2-dimensional space. Use the arc length distribution calculated in step 3 to place the points on the bottom edge along the xl-axis. The top edge is made parallell to the xl-axis. The yl-coordinate is taken to be a weighted mean of the lengths of the left and right edges. The centre of the top edge is placed so the shape of the patch is kept as much as possible. Use the arc length distribution calculated in step 3 to distribute the points along the top edge. The points on the left and right edges are distributed along straight lines using the arc length distribution calculated in step 3.

5) The interior points are calculated as the intersection of straight lines between lower-upper edge and left-right edge.

This mapped 2-dimensional surface network will also be the new 2-dimensional background grid after diagonalization. The 3-dimensional tranformation matrix M_3 (3×3) is interpolated from the original background grid (the tetrahedra are stored in an ADT data structure) and translated to a 2-dimensional transformation matrix M_2 (2×2) applied at the points of the 2-dimensional surface network. Since a 2-dimensional background grid is used, and not the original 3-dimensional background grid, exactly the same grid generation program (ADVFRO) can be used both for surface grids and ordinary 2-dimensional grids. The drawback is that the surface network must be fine enough to resolve the characteristics of the background grid. However, this should be no problem since the surface network always can be refined. This boundary and new background grid will then be used by the 2-dimensional grid generator.

3.9 Program SPLSRF

This program maps the 2-dimensional unstructured grid, generated in the mapped space inside a surface patch, back to the physical space. The surface is approximated by a bicubic spline surface through the network of points. The 2-dimensional unstructured grid is read from a file, mapped and written back to the same file.
The steps involved in the mapping back to the physical space are:

1) Represent the surface patch by a bicubic spline surface through the network of points. I- and J-index are used as the u- and v-parameter.

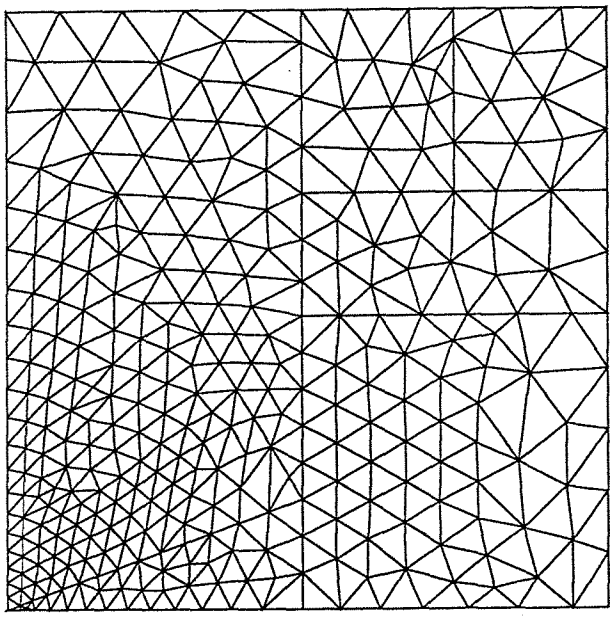

Figure 10: **Surface grid on the same surface**

2) Calculate the I- and J-index (reals) for every node in the 2-dimensional unstructured grid. Bilinear interpolation on the diagonalized surface patch is used. The ADT data structure is employed for the searching of the proper network triangle.

3) Use the spline representation to map the nodes back to the 3-dimensional space.

3.10 Program ADDSRF

This program reads all surface grid files and merges them to one surface. Figure 10 shows a surface grid generated on the surface given in figure 9. The boundary of each patch is kept, but the grid is smooth through the boundaries. The triangles are smaller in the lower left corner, as specified.

3.11 Program DELPNT

This program deletes doublets of nodes in a 2-dimensional boundary grid file. The nodes at the end of each curve in the boundary grid file exist twice. These redundant nodes are removed and the remaining nodes are renumbered.

This program does also delete doublets (triplets, etc.) of nodes in a 3-dimensional surface grid file. The nodes at the boundary of each patch in the surface grid file exist more than once, since they also exist in the adjacent patches. These redundant nodes are removed and the remaining nodes are renumbered. The ADT data structure is used for the searching involved in the algorithm.

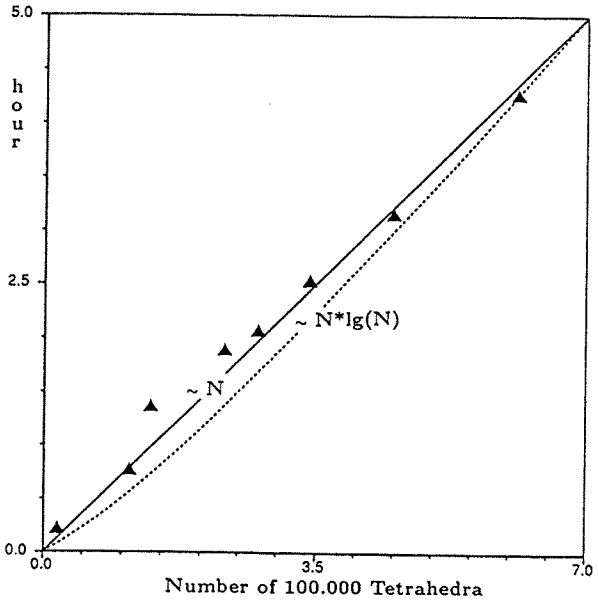

Figure 11: CPU time of grid generator, (▲).

4. CPU Times and Memory Requirements

The programs with significant execution times and memory requirements are those used for 3-dimensional grid generation. All of them have execution times less than propotional to O(Nlog(N)), where N is the number of tetrahedra. The execution times in the table below are for a 3.5 Mflops computer.

Program	Number of words/point	Minutes/ 100.000 tetrahedra
ADVFRO	25	45
RESTRT	60	1
SMOOTH	80	5
COLOUR	80	2

Figure 11 shows execution times for the grid generator ADVFRO on a 3.5 Mflops computer. We see that the execution time depends almost linear upon the number of tetrahedra. Theoretically the execution time should follow the asymptotic Nlog(N) behaviour of the ADT algorithm. The program ADVFRO requires less memory than the other programs. The reason is that only the faces in the front is stored in the memory.

5. The Grid Generation Package

The 3-dimensional grid generator has been modified for use also for the generation of 2-dimensional grids, since there are so much in common. Thus, there is only one grid generator used for all purposes. The grid generation package requires a background grid file and files defining the surface patches. There is also a parameter file common for all programs. The parameter file is given below. The amount of input given by the user, except for the background grid, is very moderate. Before the grid generation starts it is suitable to check that the grid will have the desired number of elements. This is done by use of the program NUMTET. The background grid (or the density parameter) is modified if the estimated number of elements differs to much from the desired number of elements.

```
$SURFACE PATCHES
---- number of patches ---------------------
7
---- test distance for corners -------------
0.01

$ADVANCING FRONT
---- type of run : 1-4 ---( Restart=2-4 ) --
1
---- threshold parameter : 1-6 -------------
2
---- parameter for random number : 1-50 ----
14
---- grid density parameter : > 0.0 --------
1.0

$RESTART
---- minimum number of points on front face
3

$SMOOTHING
--- number of iterations -------------------
50
--- relaxation factor ----------------------
0.5
```

5.1 Surface Grid Generation Script

As explained in the previous sections many programs are involved in the process of generation of a surface grid. For the sake of simplicity and to make it easy to exchange modules in the future, each step in the grid generation process is represented by a separate program. These programs are linked together by use of a UNIX-script. The script below shows how the programs are linked together.

```
#!/bin/csh
#--- grand loop for all surface patches ----
#copy parameter/background file to tmp files
cp PARGRD.DAT ZZZ111.TMP
cp BACGRD.DAT ZZZ222.TMP
# read number of surface files
set LINE = `cat PARGRD.DAT | \\
awk '/\$SURFACE[ ]PATCH/ {print NR+2}'`
set NFIL = `cat PARGRD.DAT | \\
awk "NR == $LINE"`
```

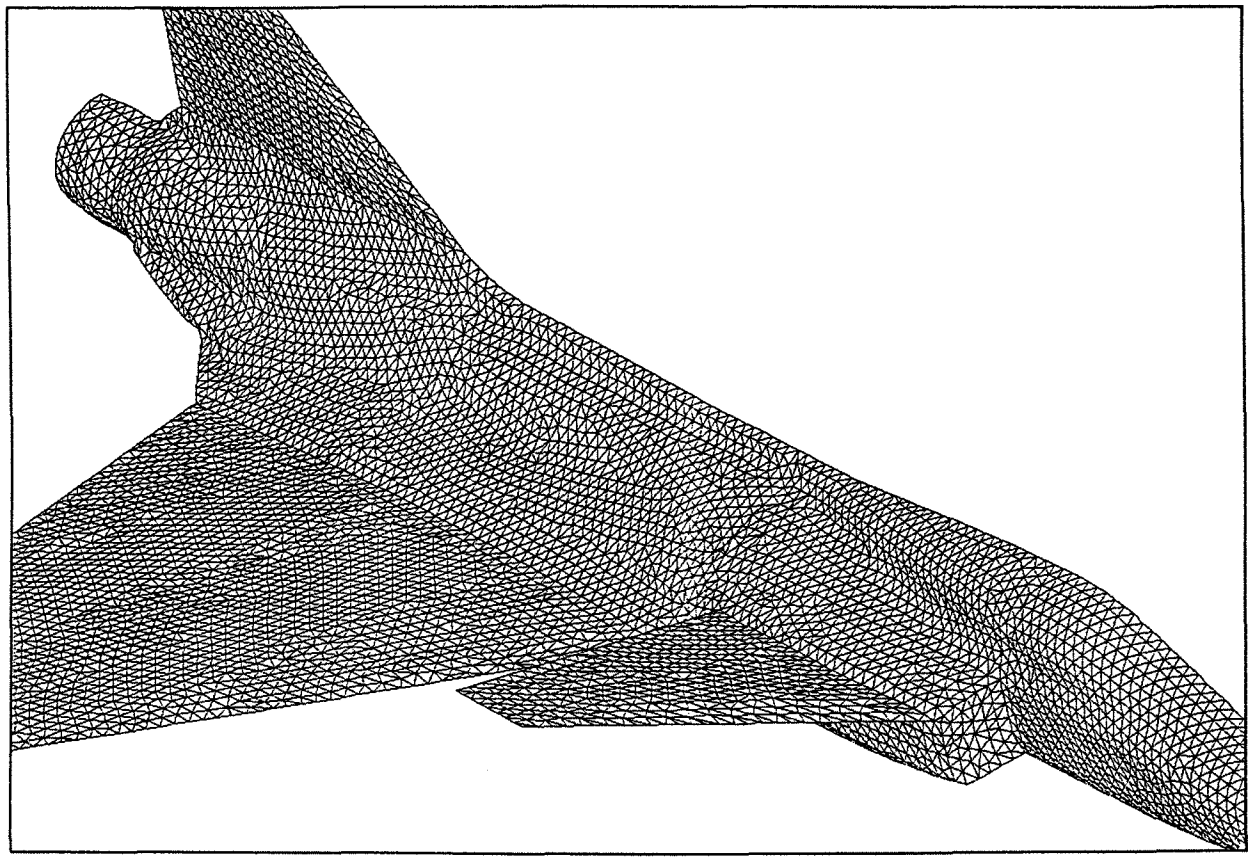

Figure 12: **Surface grid on a fighter configuration.**

```
# extract boundary curves from all patches
extedg.x
#--- grand loop for all surface patches ----
@ N = 0
while ($N < $NFIL)
@ N = $N + 1
# copy into local files
cp ZZZ111.TMP    PARGRD.DAT
cp ZZZ222.TMP    BACGRD.DAT
cp PATGRD.DAT_$N PATGRD.DAT
cp CURGRD.DAT_$N CURGRD.DAT
# divide each curve into line segments
divedg.x
# map each surface patch and boundary to 2D
mapsrf.x
# delete identical points on boundary curves
delpnt.x
# generate the unstructured grid
L1:
advfro.x > ZZZERR.TMP
# read the message file
set ERR = 'cat ZZZERR.TMP | grep -c 'ERR''
if( $ERR == 1 ) then
  cat STAGRD.LIS
  echo "Edit Parameter file"
  L2:
  echo "Then write continue."
  set dummy = $<
  if( $dummy != continue ) goto L2
  smooth.x
  restrt.x
  goto L1
endif
# smooth the grid
smooth.x
# map the 2D surface grid back
splsrf.x
cp TETGRD.NET BNDGRD.DAT_$N
end
#----------- end of grand loop -------------
# add all boundary grids into one file
addsrf.x
# delete identical points
delpnt.x
# copy parameter/background files back
cp ZZZ111.TMP PARGRD.DAT
cp ZZZ222.TMP BACGRD.DAT
# delete local files
rm ZZZ*.TMP BNDGRD.DAT_* CURCRD.DAT_*
```

Figure 12 shows a surface grid on a fighter configuration. The surface is composed of 20 surface patches. We see that the grid is smooth and regular, also at the patch boundaries.

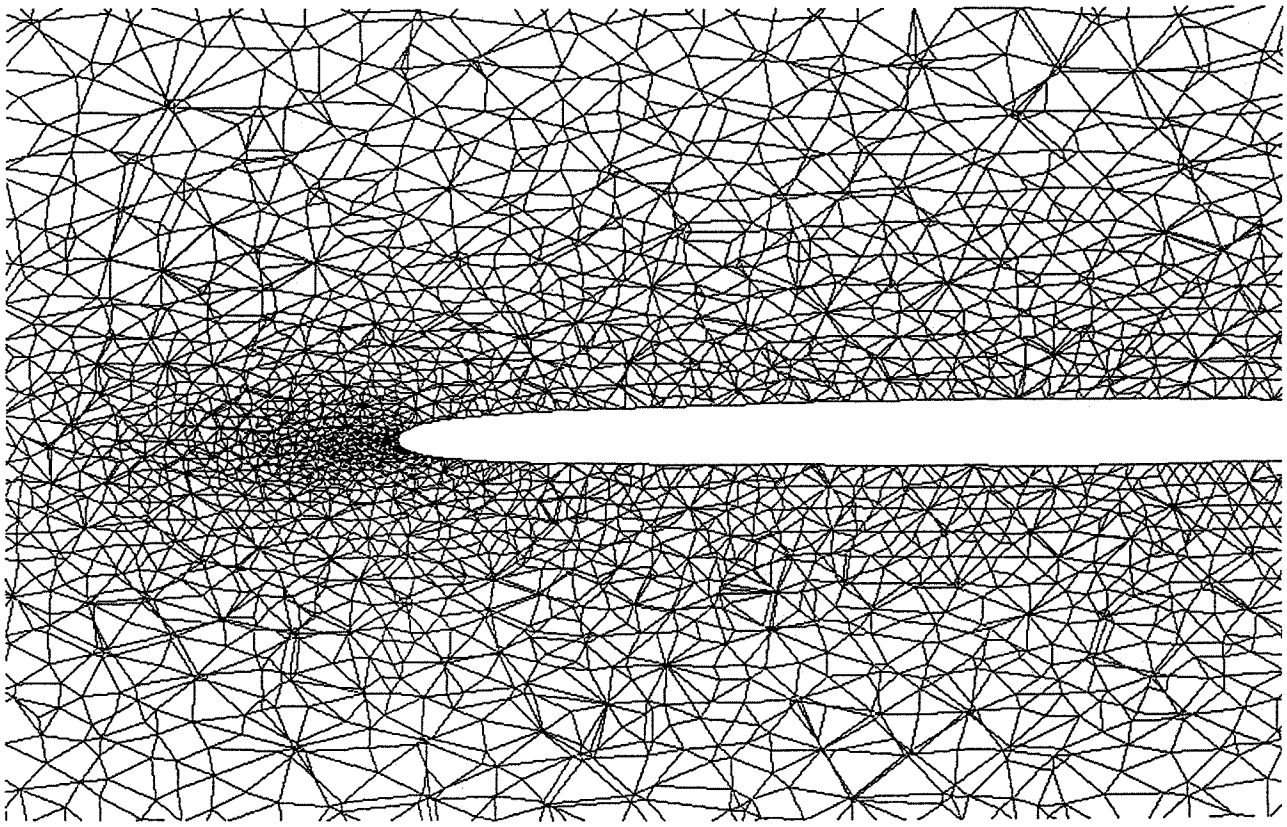

Figure 13: **Cut through a 3-dimensional grid around a delta wing.**

5.2 Volume Grid Generation Script

When the surface grid has been generated it is time for the generation of the volume grid. The programs involved are: ADVFRO, RESTRT, SMOOTH and COLOUR. They are used one after the other. The script below shows how the programs are linked together. We see that smoothing is done before the layers around the holes are deleted. This gives a modified (smoothed) grid around the holes, which will facilitate the grid generation.

```
#!/bin/csh
# copy the parameter file to a tmp file
cp PARGRD.DAT ZZZ111.TMP
# generate the unstructured grid
L1:
advfro.x > ZZZERR.TMP
# read the message file
set ERR = 'cat ZZZERR.TMP | grep -c 'ERR''
if( $ERR == 1 ) then
  cat STAGRD.LIS
  echo "Edit Parameter file"
  L2:
  echo "Then write continue."
  set dummy = $<
  if( $dummy != continue ) goto L2
  smooth.x
  restrt.x
  goto L1
endif
# smooth the grid
smooth.x
# colour the grid
colour.x
# copy the parameter file back
cp ZZZ111.TMP PARGRD.DAT
# delete local files
rm ZZZ*.TMP
```

Figure 13 shows a cut through the 3-dimensional grid around the delta wing. The geometry was shown in figure 2. 3-dimensional unstructured grids are very difficult to visualize. A cut through a 3-dimensional grid gives a very peculiar pattern, since the plane randomly cuts the tetrahedra close to the top or the base. The problem can be solved by showing the cut as an iso-surface, where the scalar can be e.g. the volume or the skewness of the tetrahedron. Then the quality of the grid is shown by the colour pattern, not the shape and size of the polygons. Our flow visualization program FloView has been extended with this feature for unstructured grids. Unfortunately that kind of colour pictures are too difficult to reproduce in black and white. Anyhow, the figure indicates that there are small tetrahedra at the leading edge, larger tetrahedra mid-chord and away from the wing profile.

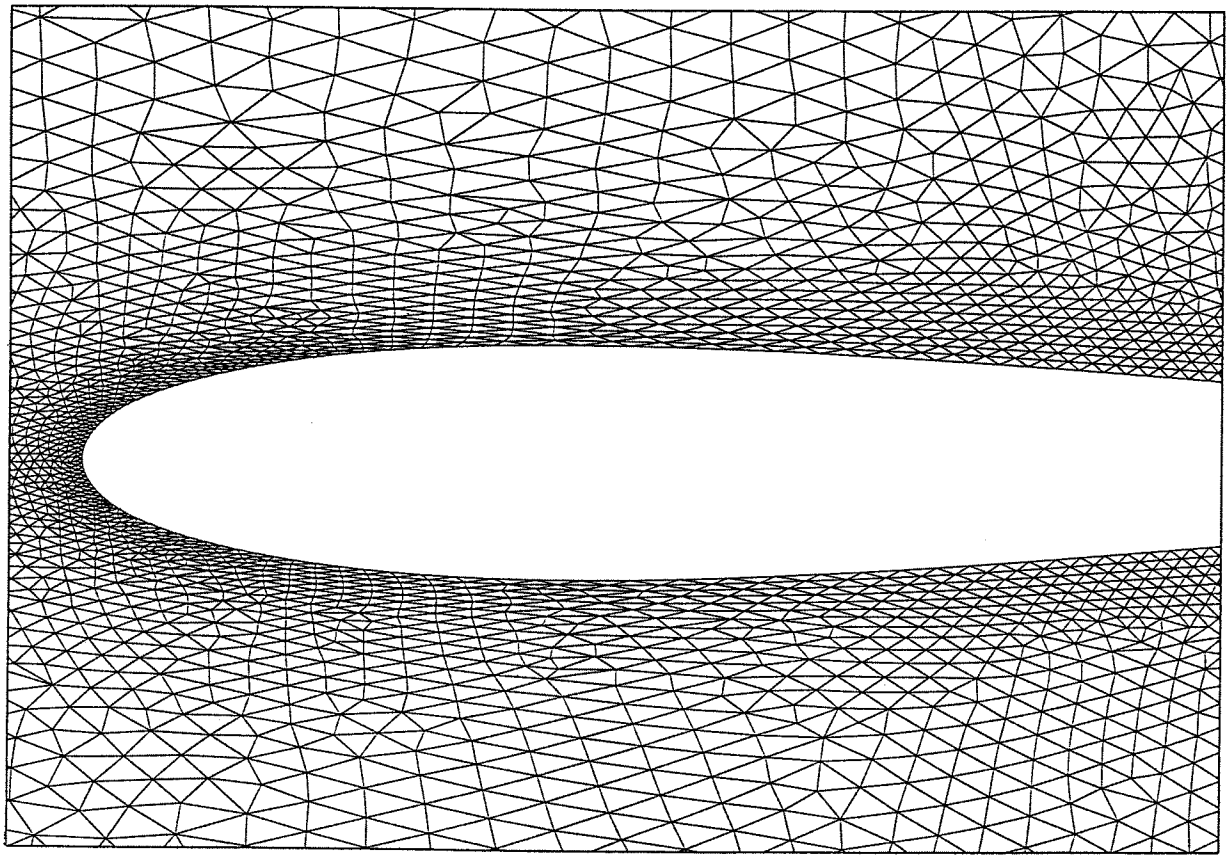

Figure 14: **Grid around a wing profile.**

5.3 2-Dimensional Grid Generation Script

Generation of 2-dimensional grids involves the programs DIVEDG, DELPNT, ADVFRO, RESTRT, SMOOTH and COLOUR only. In this case the boundary grid is computed by use of program DIVEDG only. The script below shows how the programs are linked together.

```
#!/bin/csh
# copy the parameter file to a tmp file
cp PARGRD.DAT ZZZ111.TMP
# divide each curve into line segments
divedg.x
# delete identical points on boundary
delpnt.x
# generate the unstructured grid
L1:
advfro.x > ZZZERR.TMP
# read the message file
set ERR = 'cat ZZZERR.TMP | grep -c 'ERR''
if( $ERR == 1 ) then
  cat STAGRD.LIS
  echo "Edit Parameter file"
  L2:
  echo "Then write continue."
  set dummy = $<
  if( $dummy != continue ) goto L2
  smooth.x
  restrt.x
  goto L1
endif
# smooth the grid
smooth.x
# colour the grid
colour.x
# copy the parameter file back
cp ZZZ111.TMP PARGRD.DAT
# delete local files
rm ZZZ*.TMP
```

Figure 14 shows the grid around a wing profile. The grid has a variable stretching decreasing away from the wing profile. The stretching is approximately 20/1 in the wake region (not shown here).

6. Future Work

We will continue the work and develop a program for the generation of the background grid. Then this program package will be complete. The main feature of this module will be:
The background grid nodes are specified in an interactive session on a graphics workstation using the mouse. The size and directional stretching of the grid

tetrahedra are specified at each node. Then, the background grid is automatically generated by a Delaunay algorithm. This will reduce the time spent by the user giving the input. There will also be a possibility for the user to connect the nodes to a grid by use of the mouse. In order to generate highly stretched grids for viscous calculations we will implement the Advancing Layer concept[13]. We will also implement adaption by use of remeshing and a gradient sensor as the main method. This will just be another module added to the program package.

7. Flow solver

The grids will be used by the unstructured flow solver for 2- and 3-dimensional geometries we are developing at FFA. The flow solver[12] is based on finite volume discretization using the trapezoidal rule of integration. For dissipative terms, a blend of Laplacian and biharmonic operators is employed. A cell-vertex scheme was preferred over a cell-centered scheme due to substantially smaller memory requirements and better accuracy. Multi-colouring is employed to enable vectorization and parallellization of the code. Convergence is accelerated by means of local time stepping, enthalpy damping and implicit residual smoothing.

8. Conclutions

A program package for the generation of three-dimensional unstructured grids around complex geometries has been developed. The grid generator is based on the Advancing Front algorithm. Tetrahedra of variable size, as well as directionally stretched tetrahedra can be generated by specification of a proper background grid. The geometry is defined by a set of surface patches. Each patch is represented by a quadrilateral network of points. The highly modularized program package is easy to maintain and extend. Surface grids and volume grids generated show that the program package works well.

9. References

[1] R. L. Sorensen, K. M. Mccann, A Method for Interactive Specification of Multiple-Block Topologies, Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Arcilla A. S. , Hauser J. (Eds.), Elsevier Sience Publishers B. V. 1991.

[2] B. K. Soni, J. F. Thompson, GENIE++, EAGLEView and TIGER: General and Special Purpose Graphically Interactive Grid Systems. AIAA Paper 92-0071, 1992.

[3] Ch. Hirsch, C. Dener, IGG - An Interactive 3D Surface Modelling and Grid Generation Systems. AIAA Paper 92-0073, 1992.

[4] J. P. Steinbrenner, J. R. Chawner, Incorporation of a Hierachical Grid Component Structure into GRIDGEN. AIAA Paper 93-0429, 1993.

[5] J. M. de la Viuda, J. Diet, G. Ranoux, Patch-Independent Structured Multiblock Grids for CFD Computations, Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Arcilla A. S. , Hauser J. (Eds.), Elsevier Sience Publishers B. V. 1991.

[6] N. P. Weatherill, O. Hassan, D. L. Marcum, Calculation of Steady Compressible Flowfields with the Finite Element Method. AIAA Paper AIAA-93-0341, 1993.

[7] R. Löhner, P. Parikh, Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method. AIAA Paper AIAA-88-0515, 1988.

[8] J. Peraire, K. Morgan, J. Peiro, Unstructured Mesh Methods for CFD. VKI Lecture Series 1990-06, 1990.

[9] J. Bonet, J. Peraire, An Alternating Digital Tree (ADT) Algorithm for 3D Geometric Searching and Intersection Problems. International Journal for Numerical Methods in Engineering, vol 31, 1991.

[10] R. Löhner, Some Useful Data Structures for the Generation of Unstructured Grids. Communication in Applied Numerical Methods, vol 4, 1988.

[11] P. Parikh, S. Pirzadeh, R. Löhner, A Package for 3-D Unstructured Grid Generation, Finite-Element Flow Solution and Flow Field Visualization. NASA CR-182090, 1990.

[12] T. Berglind, Finite Volume Solutions of 2D Euler Equations on Triangular Grids. FFA TN 1994-17, Stockholm, 1994.

[13] S. Pirzadeh, Unstructured Viscous Grid Generation by the Advancing-Front Method. NASA CR-191449, 1993.