TOWARDS A GENERAL THREE-DIMENSIONAL GRID GENERATION SYSTEM *

L. G. Tysell and S. G. Hedman
The Aeronautical Research Institute of Sweden (FFA)
Stockholm, Sweden

* *This work, which is a part of joint project between FFA and NLR in the Netherlands, was sponsored by the National Swedish Board for Technical Development.*

ICAS-88-4.7.4

TOWARDS A GENERAL THREE-DIMENSIONAL GRID GENERATION SYSTEM *

L. G. Tysell and S. G. Hedman
The Aeronautical Research Institute of Sweden (FFA)
Stockholm, Sweden

Abstract

The first version of a general purpose 3-dimensional grid generation program system has been developed. The program is interactive and very user-friendly. It can be applied to 2- or 3-dimensional grids. The grid can be composed of a number of blocks, both patched and overlaid grids, where each block can have its own topology. The patched grid blocks can be either continuous or discontinuous at the block interfaces. The grid is generated by means of transfinite interpolation. A procedure for smoothing of the normal vectors to a surface is presented, as well as a procedure for smoothing of grids. Grid generation routines developed for special applications can be added to the program. The program can also be used to postprocess grids from, or preprocess grids to, other grid generation programs. An example with a grid around a wing-body configuration is given.

1. Introduction

Numerical grid generation contains many problems. Different approaches can be applied to solve these problems, as can be seen in a review of grid generation methods[1]. To make good grids for finite volume calculations about realistic 3-dimensional geometries is a very difficult task. Therefore, the tendency in grid generation so far has been to develop a new grid generation program for each new type of application. But this is very expensive. In the long run it would be better with grid generation programs of a more general nature. Recently some more general codes have been developed[2,3,4]. A general code should be interactive[5], so the user can build up the grid step by step, inspect it, and gradually improve it. The program should also be able to handle grids composed of a number of blocks, both patched grids[4,6,7,8] and overlaid grids, where each block can have its own topology. To write an interactive program for 2-dimensional grids is relatively easy, since the user in such a case can treat all areas of the grid where difficulties arise one at a time. There is only one grid plane to inspect, and the user can run the program without too much work, even if such a program can do only very basic operations. On the other hand, to write a general purpose grid generation program for 3-dimensional grid generation is a much more difficult problem, since it is impossible for the user to inspect all grid planes and treat them one at a time. That implies that the operations for such a program must be performed on large regions of the grid each time they are used. Thus, the operations must be very robust, they must be reliable and function within a range of geometry variations.

Although it is possible to generate grids for arbitrary geometries with this type of general operations, it is useful to have the possibility to add routines, developed for special applications, to the program. That will reduce the number of operations the user has to do. This type of program will then be a very powerful grid generation tool, since one will have the access both to more general functions and more specific functions. The program can, of course, be used to postprocess grids generated by other grid generation programs, if only the grid file has the proper format.

2. Program Structure

The aim of this work has been to write a user-friendly interactive general grid generation program. In all types of interactive programs the user has to give a lot of input. It is important to reduce the amount of input to a minimum. We have done that by using a form for each command. From the user's point of view the program can be decomposed into five levels, see also Figure 1.

1 - Main Program
2 - Command Interpreter
3 - Form Manager
4 - General Grid Generation Routines
5 - Grid Generation Routines for Some Applications

The Main Program calls the proper routines. It also gives a journal file of the run, so that all input data given on the screen are saved. This journal file can be used for a new run.

* *This work, which is a part of joint project between FFA and NLR in the Netherlands, was sponsored by the National Swedish Board for Technical Development.*

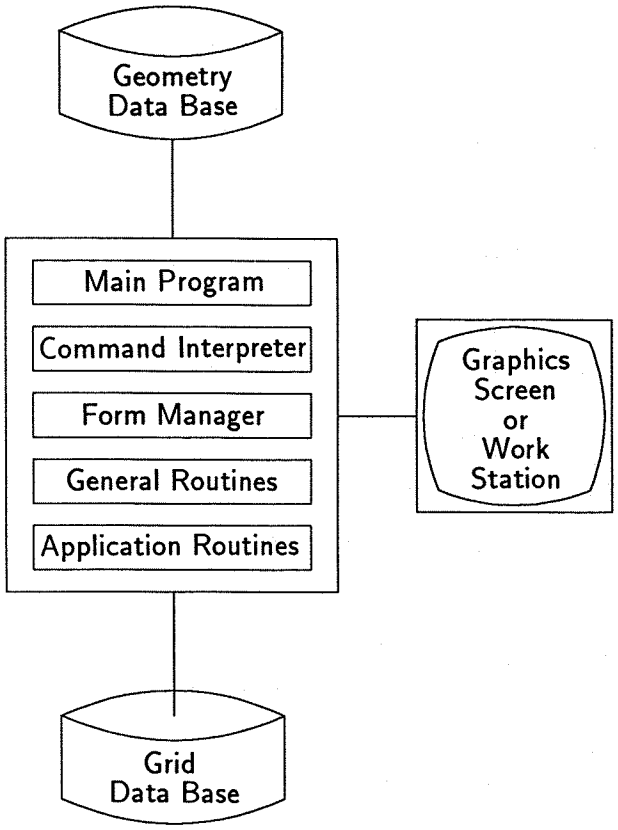

Figure 1. Structure of the grid generation program.

At present the user has access to about 30 commands. A command is executed by typing the command name. The Command Interpreter takes care of the given command and calls the proper form. Abbreviated command names are allowed as long as they are unique. A list of the commands is given in Table 1, but only a few of them will be explained in this paper.

OUT-UNIT	INPUT
SETUP-ARRAY	OUTPUT
FILE-SIZE	ARRAY-SIZE
FIG-INIT	FIG-TEXT
FIG-PLOT	FIG-EXECUTE
COPY	INPOINT
INVECTOR	MULTIPLY
MIXING	TRANSFINITE
SMOOTH	SPLINE
INTERSECTION	NORMAL
DIRECTION-SMOOTH	CIRCLE
VOLUMETRIC	COORDINATE
STATUS-NUMBER	WINGBODYGRID
OLD-FORM	STOP

Table 1. List of interactive commands.

The Form Manager then shows the form on the screen with filled up default values. With use of the keyboard one can go around with the cursor and change all, or just some values. The Form Manager checks that the user given input values do not exceed bounds. Then the Main Program calls the proper program routine and executes the command. If the program cannot find any obviously erroneous input data and no execution error appears, the data are saved on the journal file. Otherwise the user has to do the command again. Another alternative is to save the input data on the journal file without execution. The journal file can then be executed in a later run, preferably in batch mode. The form for the transfinite interpolation command is shown in Figure 2 as an example of the forms. This form, with exactly these values, has been used to produce the grid in Figure 6. The I, J and K index for the six sides of the computational cube are specified on the third row. The number of boundary conditions on these sides are specified on the fourth row, and the parameters for the stretching function are specified on the fifth row.

The general grid generation routines are, for example, routines for surface spline fitting, calculation of normals to surfaces, grid generation inside a given region, smoothing of irregular grids or plotting of grids. The general routines are independent of geometry, as well as grid topology. They can be applied to any region of any grid block.

It is convenient to have routines, besides the general routines, that make grid generation more easy for special applicatons. Therefore it is easy to add such routines to the program. They can be composed of the general routines. One application routine we have is a routine for grid generation around a wing-fuselage configuration.

3. Program features

The grid generation program is composed of modules, which are as independent of each other as possible. Then it is easy to replace old modules and add new ones. The program, which is written in Fortran 77, consists of almost 10,000 statements, excluding comment lines and library routines. In the Command Interpreter and the Form Manager some statements are dependent of the type of computer and terminal. At present the program runs on VAX computers and VT100/240 compatible terminals.

A code of this type differs very much from a non-interactive code only used for grid generation around some particular type of application. The program can be used irrespective of the number of grid blocks and their sizes, up to a certain level. Consequently there is only one array in the program, and all the grid blocks are put into that ar-

```
# TRANSFINITE
PURPOSE: To do transfinite interpolation of a grid.

GRID TO BE INTERPOLATED _____________ :  WING/BOX-GRID

GRID DEFINING THE BOUNDARY _____________ :  BOUNDARY-GRID

SELECTED REGION _____________________ :  1     57    1     10    1     1

NUMBER OF BOUNDARY CONDITIONS ___<1> :  0     0     2     2     0     0

INTERPOLATION PARAMETERS _____<2> :  1.    1.    1.5   1.5   1.    1.

INTERPOLATION FOR X-Y-Z ? _____________ :  YES   YES   NO

-------------------------------------------------------------------------
<1> 0 = No interpolation.
    1 = Only the boundary points are specified.
    2 = Also the layer next to the boundary is specified.

<2> 1 = Normal value. Large value => the condition at the boundary is
        extended inside the region.
```

Figure 2. The transfinite interpolation form.

ray. The program is also coded in order to prevent all errors that could have been caused by erroneous input. For example, it prevents division by zero. Otherwise there will be a fatal error and several hours of work might be lost.

4. Some Difficulties with General Geometries

There are two dominating types of methods for grid generation, viz. algebraic methods and elliptic methods[9,10]. We have used transfinite interpolation[9,11,12,13], which is an algebraic method, since an interactive program should have short execution times. Transfinite interpolation is much faster than elliptic methods. The main disadvantage with tranfinite interpolation, compared to elliptic methods using a Laplace system, is that there is no inherent guarantee against grid inversion. But this guarantee against grid inversion is based on the maximum principle for the Laplace's equation. The maximum principle assumes that the boundary is smooth, but that is not the case in the discrete formulation[8]. Anyhow, the Laplace grid generation system is not so attractive, since it does not give the desired grid point distribution. Usually a Poisson system is used instead. Then the maximum principle is lost. Nevertheless, these elliptic methods are probably safer than transfinite interpolation.

Using transfinite interpolation, the grid points in the interior of a region are calcululated by interpolation from the grid points on the boundaries. That means, if the boundaries are non-smooth the grid will not be smooth either. Thus the grid must be postprocessed after the transfinite interpolation in order to smooth the grid and remove cross-overs.

The input to the transfinite interpolation is usually both the grid points on the boundary surfaces and the desired out-of-surface derivatives, i.e. the direction and spacing of the outgoing gridlines. For the 2-dimensional case it is usually possible to estimate the out-of-surface derivative. For example, to make an O-grid around a wing profile is easy. The out-of-surface derivatives to the wing can be taken as the normal derivatives to an ellipse with the same chord. But it is almost impossible to do something similar for more complicated 3-dimensional geometries. It may seem natural to take the local normal direction to the surface as the out-of-surface direction. However, that will give rise to a very coarse volume grid where the boundary is strongly convex. This is the reason why the ellipse was used for the wing profile case to get a good grid also at the trailing edge. Furthermore, grid inversion (cross-overs) may arise at concave areas. A way to solve this problem is to take the normals as a first guess, and then smooth them. But the ordinary Laplace smoother is not suited for that. We have formulated a new smoother that works very well for general surfaces.

To generate a 2-dimensional grid on a specified surface is a problem of its own[14], since this can be very difficult for more complicated surfaces. These grids are input to the 3-dimensional transfinite interpolation. If the surface can be represented by a spline formulation the grids can be generated by the program. Otherwise the surface grids has to be provided by a CAD-system[15]. We have done some studies where the grid has been generated on a simpler surface and then projected on the real surface[7]. But we still do not have any final general solution to this problem.

In section 5 through section 9 we will give our solution to the problem of how to calculate the out-of-surface direction and how to remove cross-overs and smooth the grid.

5. Examples Demonstrating the Grid Generation Procedure

All commands used for 3-dimensional grids can be used for 2-dimensional and even 1-dimensional grids, if they make any sense for these dimensions. The algorithms in the next four sections are given for a 3-dimensional grid, but they can easily be modified to lower dimensions. The first demonstration example is a wing profile inside a 2-dimensional box. The geometry for this example is shown in Figure 3. The second example is the grid in Figure 8, which has a severe cross-over. Note that no modifications of the formulae to suit these particular geometries have been made. The formulae can be applied, just as they are, to general geometries much more complex than these examples. The final example is the grid in Figure 10, showing the 3-dimensional grid around a wing-fuselage configuration. All the grids presented are generated and plotted only with use of the interactive commands in table 1.

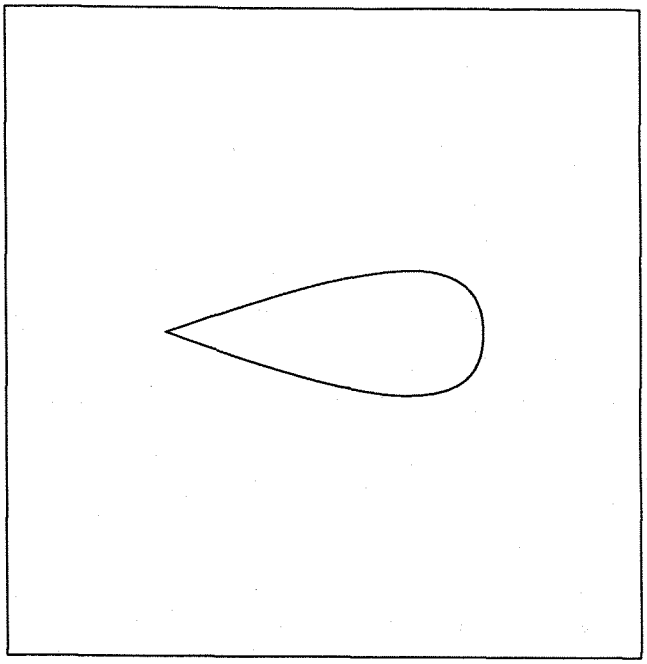

Figure 3. Geometry for the 2-dimensional example.

6. Calculation of Normals to Surfaces

Assuming the surface grid on the boundary already exists, the next step is to calculate the out-of-surface derivative at each surface grid point. The magnitude can be specified, but the direction must be computed. A natural first guess is to take the local normal direction to the surface. The surface grid might have discontinuities, so a spline formulation for the surface is not suitable. We want a formula that gives a "normal" also in corners. The surface grid might also be very irregular. That means that the formulae for the normal calculation must be very robust. The simple scheme below satisfies these conditions. The normal is calculated by just summing the cross-products of the vectors drawn from the current surface grid point to the surrounding grid points.

Let the vector $\vec{X}_B$ denote the prescribed value of the grid

$$\vec{X}(i,j,k) = (x(i,j,k), y(i,j,k), z(i,j,k))$$

on the sides of the computational cube

$$\begin{aligned} i_a \le i \le i_b \\ j_a \le j \le j_b \\ k_a \le k \le k_b \end{aligned}$$

The normal vector $\vec{N}(j,k)$ to the surface $\vec{X}_B(i_a,j,k)$ is calculated by the simple formulae (1)-(3) below.

$$\begin{aligned} \vec{R}_1(j,k) =& (\vec{X}_B(i_a,j+1,k) - \vec{X}_B(i_a,j,k)) \times \\ & (\vec{X}_B(i_a,j,k+1) - \vec{X}_B(i_a,j,k)) \\ \vec{R}_2(j,k) =& (\vec{X}_B(i_a,j,k+1) - \vec{X}_B(i_a,j,k)) \times \\ & (\vec{X}_B(i_a,j-1,k) - \vec{X}_B(i_a,j,k)) \\ \vec{R}_3(j,k) =& (\vec{X}_B(i_a,j-1,k) - \vec{X}_B(i_a,j,k)) \times \\ & (\vec{X}_B(i_a,j,k-1) - \vec{X}_B(i_a,j,k)) \\ \vec{R}_4(j,k) =& (\vec{X}_B(i_a,j,k-1) - \vec{X}_B(i_a,j,k)) \times \\ & (\vec{X}_B(i_a,j+1,k) - \vec{X}_B(i_a,j,k)) \end{aligned} \tag{1}$$

Normalize each vector in order to give them the same magnitude, independently of the surface grid spacing. Then they are summed to

$$\vec{R}(j,k) = \sum_{n=1}^{4} C_n \frac{\vec{R}_n(j,k)}{|\vec{R}_n(j,k)|} \tag{2}$$

and to achieve the desired magnitude

$$\vec{N}(j,k) = \pm \frac{N_{spec}(j,k)}{|\vec{R}(j,k)|} \vec{R}(j,k) \qquad (3)$$

where $N_{spec}(j,k)$ is the specified magnitude.

The coefficient C_n is given by

$$C_1 = \begin{cases} 0 & \text{if } j = j_b \text{ or } k = k_b \\ 1 & \text{otherwise} \end{cases}$$

C_2 , C_3 and C_4 are defined in the same way.

When having a multi-block grid generation program, it is not suitable to store both the surface grid and the out-of-surface derivatives. The logic of the program would then be more complicated than necessary. Thus, calculate the next layer of points instead. Then only points need to be stored.

$$\vec{X}_B(i_a+1,j,k) = \vec{X}_B(i_a,j,k) + \vec{N}(j,k)$$

The other sides of the computational cube are treated in the same way.

Figure 4 shows the grid points on the boundaries and the grid points next to the boundaries calculated by this method. It can be seen that the method did calculate a "normal" at the trailing edge of the wing and at the corners on the outer bondary. The branch cut is located at the leading edge. It should always be placed where the boundary is smooth, since the formulae in this section and in section 7 and section 9 work well only if there are no discontinuities at the branch cut.

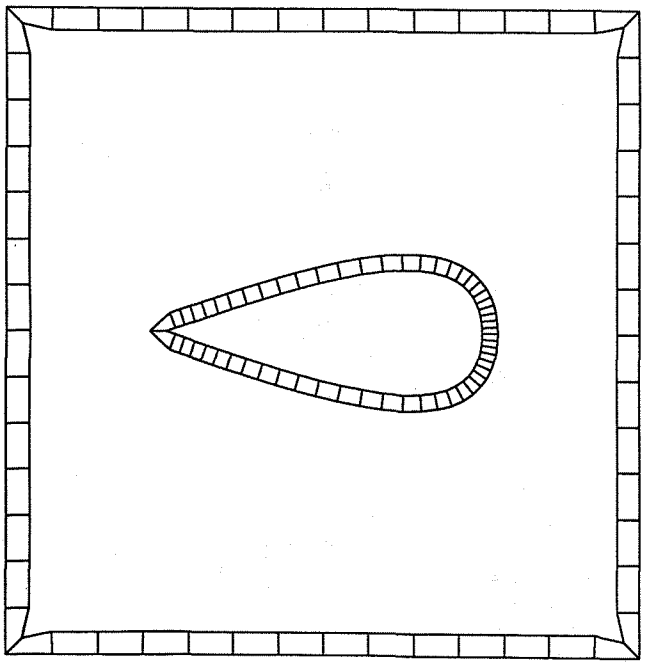

Figure 4. Out-of-surface derivatives.

7. Smoothing of Normals

The layer of points next to the boundary can usually not be used as input to the transfinite interpolation, unless it is modified. Otherwise the grid may become inverted (cross-overs) where the boundary is strongly concave. The grid would also become very coarse where the boundary is strongly convex, see the sharp corners in Figure 4. The points next to the boundary must be smoothed. The ordinary Laplace smoother is not suited for that, but the smoothing scheme defined below works very well.

Define the 2-dimensional Laplace operator $\Lambda_{JK}(\vec{X})$ operating on the surface i =constant

$$\begin{aligned} \Lambda_{JK}(\vec{X}(i,j,k)) = {} & \vec{X}(i,j+1,k) + \vec{X}(i,j-1,k) \\ & + \vec{X}(i,j,k+1) + \vec{X}(i,j,k-1) \\ & - 4\vec{X}(i,j,k) \end{aligned}$$

Define also the unit normal vector

$$\vec{n}(j,k) = \frac{\vec{N}(j,k)}{|\vec{N}(j,k)|}$$

where $\vec{N}(j,k)$ is the original normal vector calculated by the formulae in section 6.

$$\vec{N}(j,k) = \vec{X}_B(i_a+1,j,k) - \vec{X}_B(i_a,j,k)$$

Define the smoothing operator $\Gamma(\vec{X}_B)$ operating on the surface $i = i_a + 1$

$$\begin{aligned} \Gamma(\vec{X}_B(i_a+1,j,k)) = {} & \Lambda_{JK}(\vec{X}_B(i_a+1,j,k)) \\ & - \Lambda_{JK}(\vec{X}_B(i_a,j,k)) \\ & + |\Lambda_{JK}(\vec{X}_B(i_a,j,k)) \cdot \vec{n}(j,k)| \vec{n}(j,k) \end{aligned}$$

The first term is the ordinary Laplace smoother. By adding the second term the smoother can be used also for surface grids having discontinuous grid point distribution. Otherwise the smoother would work only for equally spaced surface grids without any discontinuities. The third term is added in order to take care of convex corners.

The new normal vector is then given by

$$\vec{N}_{new}(j,k) = \vec{X}_B(i_a+1,j,k) - \vec{X}_B(i_a,j,k) + C_{rel}\Gamma(\vec{X}_B(i_a+1,j,k)) \quad (4)$$

where C_{rel} is a relaxation factor. Then

$$\vec{X}_B(i_a+1,j,k) = \vec{X}_B(i_a,j,k) + \frac{|\vec{N}(j,k)|}{|\vec{N}_{new}(j,k)|}\vec{N}_{new}(j,k) \quad (5)$$

Now go back to (4) and do as many iterations as necessary. Note that $\vec{N}$ is unchanged throughout the iterations. The other sides of the computational cube are treated in the same way.

Figure 5 shows the grid after the smoothing. The grid is much smoother than in Figure 4, but it is still mainly orthogonal. The smoother keeps the grid as orthogonal as possible. The grid can now be used as input to the transfinite interpolation.

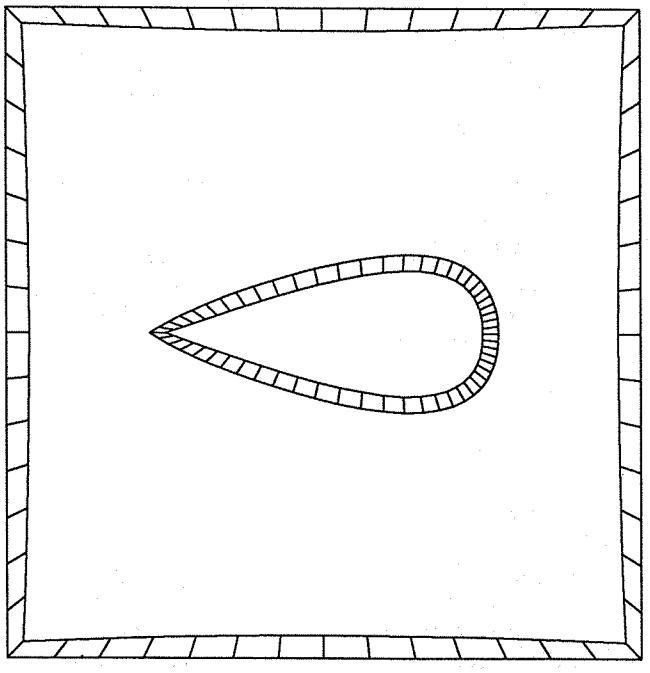

Figure 5. Smoothed out-of-surface derivatives.

8. Transfinite Interpolation

Transfinite interpolation is an algebraic grid generation method. Thus, it is very fast and suitable for interactive use. It also offers good control of the grid generation procedure. The method used below is a standard transfinite interpolation scheme. The input to the grid generator is two, one or no layers of points at each boundary. That makes it easy to use for multiblock grid generation or regeneration of subregions.

The grid $\vec{X}$ in the interior of the cube

$$\begin{aligned} i_a \le i \le i_b \\ j_a \le j \le j_b \\ k_a \le k \le k_b \end{aligned}$$

is obtained by use of the transfinite interpolation

$$\vec{X} = \Upsilon(\vec{X}_B)$$

defined by the recursive transfinite interpolation scheme

$$\begin{aligned} \vec{X}_I &= \Omega_I(\vec{X}_B) \\ \vec{X}_J &= \vec{X}_I + \Omega_J(\vec{X}_B - \vec{X}_I) \\ \vec{X}_K &= \vec{X}_J + \Omega_K(\vec{X}_B - \vec{X}_J) \\ \vec{X} &= \vec{X}_K \end{aligned} \quad (6)$$

where the operator Ω_I is the interpolation operator given by

$$\Omega_I(\vec{X}) = \vec{X}(i_a,j,k)\psi_a(\xi) + \vec{X}(i_a+1,j,k)\psi_{a+1}(\xi) + \vec{X}(i_b-1,j,k)\psi_{b-1}(\xi) + \vec{X}(i_b,j,k)\psi_b(\xi) \quad (7)$$

The operators Ω_J and Ω_K are defined in the same way. The functions $\psi(\xi)$ are called blending functions.

The stretching function $\xi(i)$, which can be any monotonic function, is used to control the grid point distribution.

$$0 \le \xi(i) \le 1 \quad , \quad i_a \le i \le i_b$$

Now define the two parameters N_a and N_b .

$N_a = 0 \Leftrightarrow \vec{X}_B(i_a,j,k)$, $\vec{X}_B(i_a+1,j,k)$ are not specified
$N_a = 1 \Leftrightarrow$ Only $\vec{X}_B(i_a,j,k)$ is specified
$N_a = 2 \Leftrightarrow \vec{X}_B(i_a,j,k)$, $\vec{X}_B(i_a+1,j,k)$ are specified

N_b is defined in the same way.

Let the operator Ω_I be the Lagrange interpolation operator. Then the functions $\psi(\xi)$ are the Lagrange polynomials given by

$$N_a = 1, N_b = 0 \quad \Rightarrow \begin{cases} \psi_a(\xi) = p_a^b(\xi) p_a^b(\xi) \\ \psi_{a+1}(\xi) = 0 \\ \psi_{b-1}(\xi) = 0 \\ \psi_b(\xi) = 0 \end{cases}$$

$$N_a = 1, N_b = 1 \quad \Rightarrow \begin{cases} \psi_a(\xi) = p_a^b(\xi) \\ \psi_{a+1}(\xi) = 0 \\ \psi_{b-1}(\xi) = 0 \\ \psi_b(\xi) = p_b^a(\xi) \end{cases}$$

$$N_a = 2, N_b = 0 \quad \Rightarrow \begin{cases} \psi_a(\xi) = p_a^{a+1}(\xi) p_a^b(\xi) p_a^b(\xi) \\ \psi_{a+1}(\xi) = p_{a+1}^a(\xi) p_{a+1}^b(\xi) p_{a+1}^b(\xi) \\ \psi_{b-1}(\xi) = 0 \\ \psi_b(\xi) = 0 \end{cases}$$

$$N_a = 2, N_b = 1 \quad \Rightarrow \begin{cases} \psi_a(\xi) = p_a^{a+1}(\xi) p_a^b(\xi) \\ \psi_{a+1}(\xi) = p_{a+1}^a(\xi) p_{a+1}^b(\xi) \\ \psi_{b-1}(\xi) = 0 \\ \psi_b(\xi) = p_b^a(\xi) p_b^{a+1}(\xi) \end{cases}$$

$$N_a = 2, N_b = 2 \quad \Rightarrow \begin{cases} \psi_a(\xi) = p_a^{a+1}(\xi) p_a^{b-1}(\xi) p_a^b(\xi) \\ \psi_{a+1}(\xi) = p_{a+1}^a(\xi) p_{a+1}^{b-1}(\xi) p_{a+1}^b(\xi) \\ \psi_{b-1}(\xi) = p_{b-1}^a(\xi) p_{b-1}^{a+1}(\xi) p_{b-1}^b(\xi) \\ \psi_b(\xi) = p_b^a(\xi) p_b^{a+1}(\xi) p_b^{b-1}(\xi) \end{cases}$$

where

$$p_n^m(\xi) = \frac{\xi(i) - \xi(i_m)}{\xi(i_n) - \xi(i_m)} \quad , \quad m, n = a, a+1, b-1, b$$

The blending functions $\psi(\xi)$ for the other three possible combinations of N_a and N_b can be derived from the cases given above.

In the recursive transfinite interpolation scheme (6) the I-direction is the first interpolation direction. But it can be shown that with the interpolation operator (7) and the Lagrange polynomials, the interpolation scheme is independent of the order in which the successive interpolation steps are taken. There is only one exception, the directions where N_a or N_b is zero must be the last ones.

Figure 6 shows the grid after the transfinite interpolation.

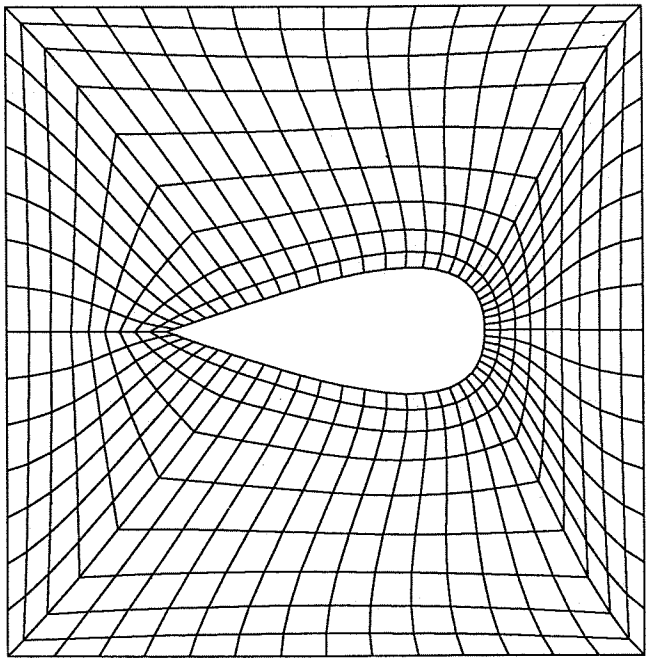

Figure 6. Grid obtained by transfinite interpolation.

9. Smoothing of Grids

In transfinite interpolation all types of discontiuities on the boundaries will propagate into the volume grid. If the grid line slopes or grid spacing are discontinuous on the boundaries, this will also be the case in the volume grid. Thus, the volume grid must be smoothed after the transfinite interpolation. But that is not an easy task, since it is almost impossible to formulate, in mathematical terms, what the grid should look like. If the grid is improved in some region, it might be less good in some other region. Has the overall grid become better? The only way to finally judge if one grid is better than another is to run the flow solver on both grids and see which grid that gives the best convergence. One suggestion to obtain a better grid is to minimize the overall discontinuities[16,17]. But does that give the optimum grid? Anyhow, this method is slow for 3-dimensional grids[18]. Another

way is to use the well-known Laplace smoother. This smoother is very fast, but not reliable. It tends to give an equally spaced grid, whatever the surface grids are. Another problem with the Laplace smoother is that it easily gives grid inversion where the boundary is concave and too large cells where the boundary is convex. We have developed a new smoother. The smoother is a type of Laplace smoother. But it is reliable and works very well for general geometries.

Define the 3-dimensional Laplace smoother $\Lambda(\vec{X})$

$$\begin{aligned}\Lambda(\vec{X}(i,j,k)) = \vec{X}(i+1,j,k) + \vec{X}(i-1,j,k) \\ + \vec{X}(i,j+1,k) + \vec{X}(i,j-1,k) \\ + \vec{X}(i,j,k+1) + \vec{X}(i,j,k-1) \\ - 6\vec{X}(i,j,k)\end{aligned}$$

and the 1-dimensional Laplace smoother $\Lambda_I(\vec{X})$ operating along the line i = constant

$$\begin{aligned}\Lambda_I(\vec{X}(i,j,k)) = \vec{X}(i+1,j,k) + \vec{X}(i-1,j,k) \\ - 2\vec{X}(i,j,k)\end{aligned}$$

$\Lambda_J(\vec{X})$ and $\Lambda_K(\vec{X})$ are defined in the same way.

Now we can define

$$\vec{X}_2 = \Lambda(\vec{X})$$

on the computational cube

$$\begin{aligned}i_a + 1 \le i \le i_b - 1 \\ j_a + 1 \le j \le j_b - 1 \\ k_a + 1 \le k \le k_b - 1\end{aligned}$$

and

$$\begin{aligned}\vec{X}_{4I} = \Lambda_I(\vec{X}_2) \\ \vec{X}_{4J} = \Lambda_J(\vec{X}_2) \\ \vec{X}_{4K} = \Lambda_K(\vec{X}_2)\end{aligned} \qquad (8)$$

on the six sides of the computational cube

$$\begin{aligned}i_a + 2 \le i \le i_b - 2 \\ j_a + 2 \le j \le j_b - 2 \\ k_a + 2 \le k \le k_b - 2\end{aligned}$$

Interpolate the values of $\vec{X}_{4I}$, $\vec{X}_{4J}$ and $\vec{X}_{4K}$ into the interior of the cube with the transfinite interpolation formula

$$\vec{X}_4^{int} = \Upsilon(\vec{X}_{4I}) + \Upsilon(\vec{X}_{4J}) + \Upsilon(\vec{X}_{4K}) \qquad (9)$$

where the blending functions for the transfinite operation $\Upsilon(\vec{X}_{4I})$ are

$$I - \text{Direction} \Rightarrow \begin{cases}\psi_a(\xi) = 1 - \xi \\ \psi_b(\xi) = \xi \\ \psi_{a+1}(\xi) = \psi_{b-1}(\xi) = 0\end{cases}$$

$$J - \text{and } K - \text{Direction} \Rightarrow \begin{cases}\psi_a(\xi) = (1-\xi)^\nu \\ \psi_b(\xi) = \xi^\nu \\ \psi_{a+1}(\xi) = \psi_{b-1}(\xi) = 0\end{cases}$$

where ν is a smoothing factor , $\nu \ge 1$. $\vec{X}_{4I}$ is smooth on boundary surfaces i=constant, but can be non-smooth on boundary surfaces j=constant and k=constant, since these surface grids themselves can be non-smooth. With the blending functions above the influence from these irregularities will be small. The blending functions for $\Upsilon(\vec{X}_{4J})$ and $\Upsilon(\vec{X}_{4K})$ are given in the same way. Now smooth $\vec{X}_2(i,j,k)$ with the scheme

$$\vec{X}_2^{new} = \vec{X}_2 + C_{rel}(\Lambda(\vec{X}_2) - \vec{X}_4^{int}) \qquad (10)$$

$$\vec{X}_2 = \vec{X}_2^{new} \qquad (11)$$

where C_{rel} is a relaxation factor. Then, go back to (10) and do as many iterations as necessary.

The last step is to smooth $\vec{X}(i,j,k)$ on the computational cube

$$\begin{aligned}i_a + 1 \le i \le i_b - 1 \\ j_a + 1 \le j \le j_b - 1 \\ k_a + 1 \le k \le k_b - 1\end{aligned}$$

with the scheme

$$\vec{X}^{new} = \vec{X} + C_{rel}(\Lambda(\vec{X}) - \vec{X}_2) \qquad (12)$$

$$\vec{X} = \vec{X}^{new} \qquad (13)$$

Go back to (12) and do as many iterations as necessary.

A simpler alternative is to skip equation (10) and (11) and apply the 1-dimensional Laplace operator in (8) on $\vec{X}$ instead of on $\vec{X}_2$. This will give a stronger but not as safe smoothing.

Figure 7 shows the grid after the smooting. The smoothing effect is strongest in the interior, and the grid close to the boundaries is almost unaffected, compare Figure 6. The grid line slopes in the interior are smoothed but the spacing is almost unaffected, since it already was smooth. Figure 8 shows another grid with discontinuous grid line slopes and cross-overs. This grid was generated in order to get a really poor grid. It is not generated with transfinite interpolation. As seen in Figure 9 the cross-overs and kinks can be removed by the smoother. Note that the smoother works well even if the grid is not equally spaced on the boundaries.

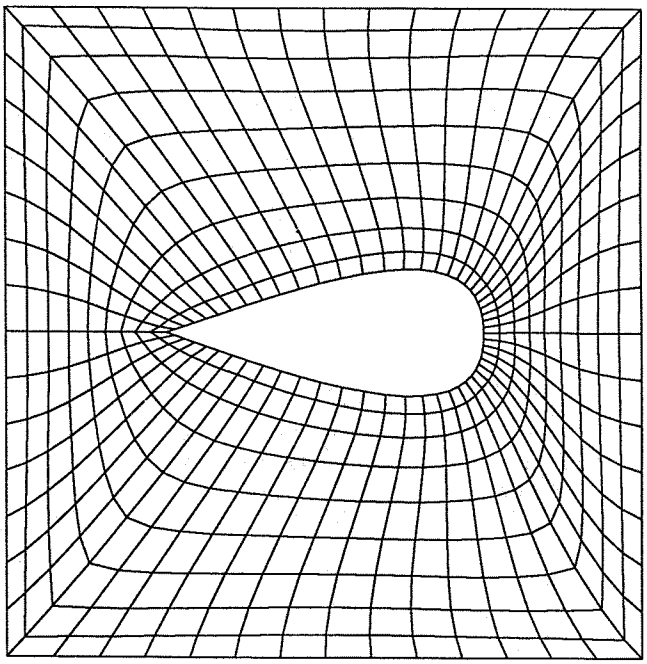

Figure 7. Smoothed grid.

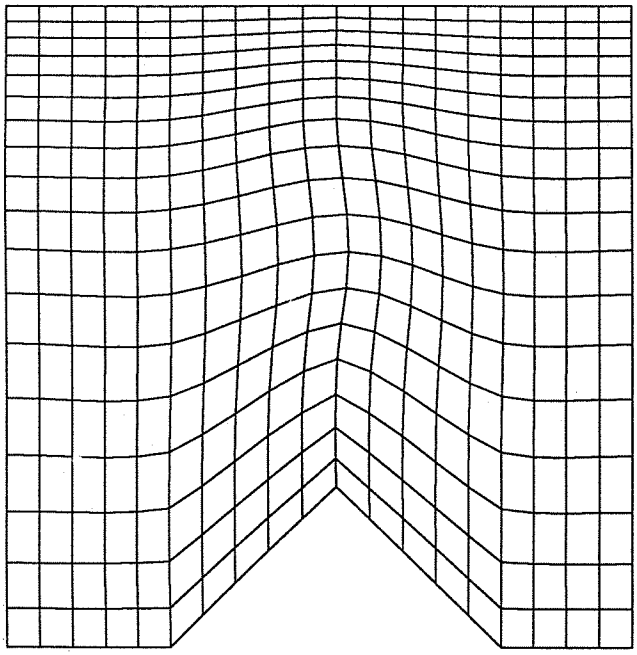

Figure 9. Smoothed grid.

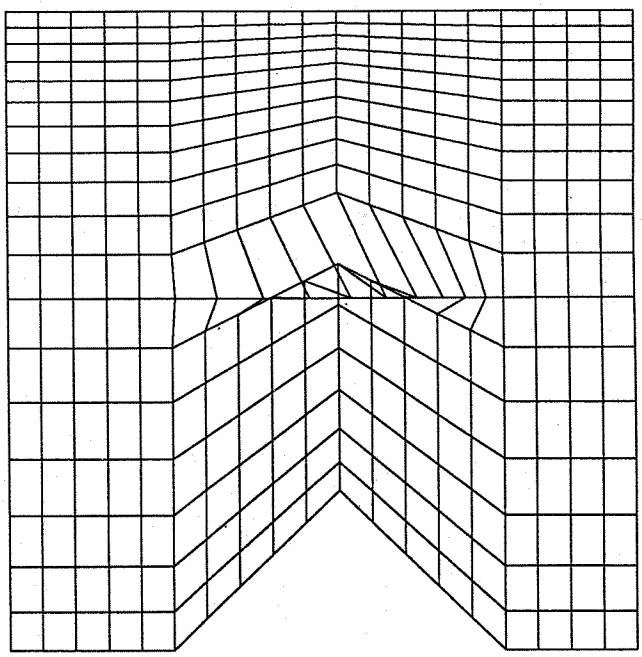

Figure 8. Grid with cross-overs and discontinuous grid line slopes.

10. Application Routines

The first application routine added to the program is a routine for generation of a single block grid around a wing alone or wing-fuselage configuration. This routine has been developed to match a full potential code developed by NLR[19]. The grid can be either of C-H or C-O type. Conformal mapping is used to transform the fuselage to the symmetry plane. The grid is generated in the transformed space by means of transfinite interpolation. The size of the fuselage contour line conformity region, as well as the location of the wing wake, are input parameters. The grid can even be completely contour line conforming or not contour line conforming at all. The grid can be characterized as a wing grid for accurate calculation of wings with reasonable consideration of the presence of the fuselage. The grid is generated with the WINGBODYGRID command, where the user has access to a large set of input parameters by which the grid can be tailored. A grid generated by this routine is shown in Figure 10 and Figure 11.

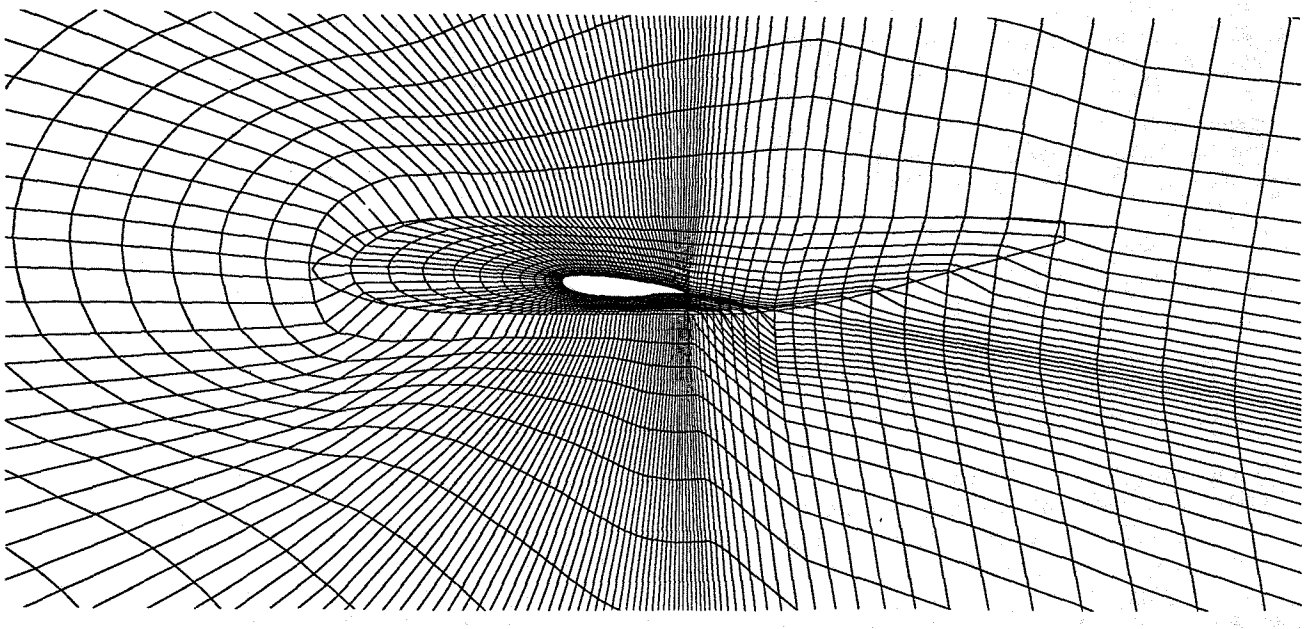

Figure 10. Grid on fuselage and symmetry plane. The fuselage contour line is also shown.

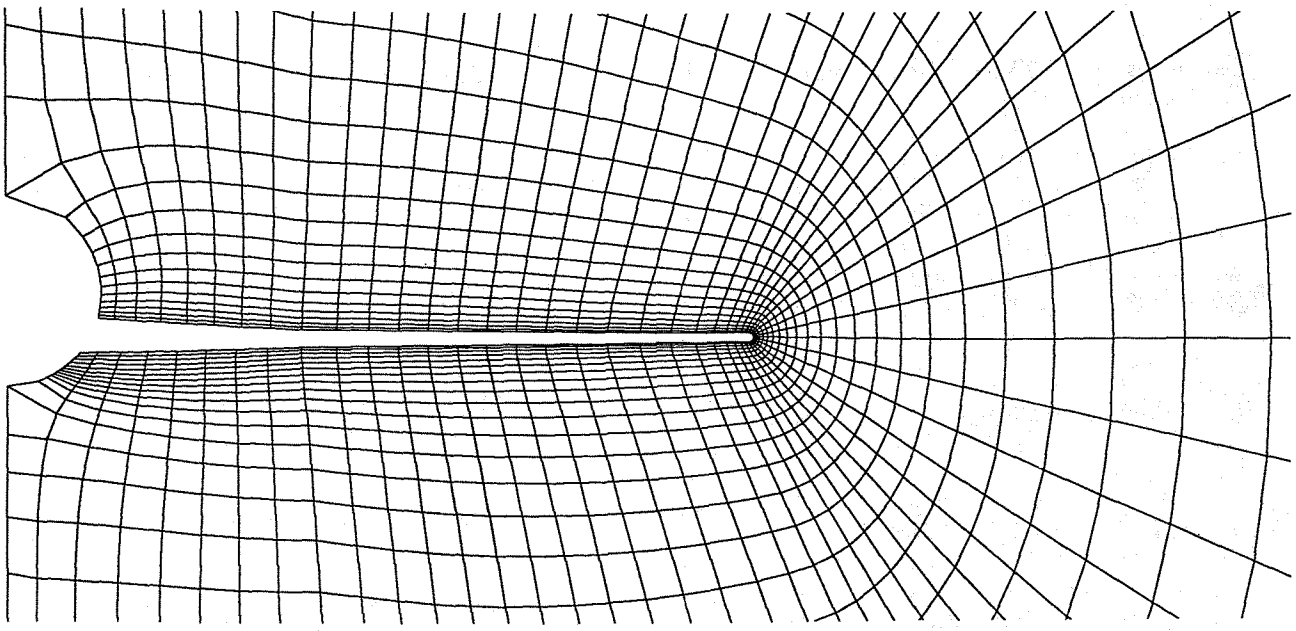

Figure 11. Spanwise section of C-O grid.

11. Future Work

The program, in its present state, can be used for grid generation of overlaid grids around complex configurations. It can also be used for generation of patched grids, with a reasonable number of grid blocks. What might be needed is some routines to set up the overall block stucture in order to make it easier to handle grids composed of a large number of blocks. The work will be directed into that area. We will also study the problem of surface grid generation for general surfaces.

12. References

1. Thompson, J. F., "Grid Generation Techniques in Computational Fluid dynamics", AIAA Journal, Vol. 22, No. 11, 1984.

2. Thompson, J. F., "A Composite Grid Generation Code for General 3-D Regions", AIAA paper 87-0275, AIAA 25th Aerospace Sciences Meeting, 1987.

3. Holcomb, J. E., "Development of a Grid Generator to Support 3-D Multizone Navier-Stokes Analysis", AIAA paper 87-0203, AIAA 25th Aerospace Sciences Meeting, 1987.

4. Woan, C. J., "Three-Dimensional Elliptic Grid Generations Using a Multi-Block Method", AIAA paper 87-0278, AIAA 25th Aerospace Sciences Meeting, 1987.

5. Smith, R. E., "Three-Dimensional Algebraic Grid Generation", AIAA paper 83-1904, AIAA 6th Computational Fluid Dynamics Conference, 1983.

6. Eriksson, L. E., "Flow Solution on a Dual-Block Grid Around an Airplane", Computer Methods in Applied Mechanics and Engineering, Vol. 64, 1987.

7. Rubbert, P. E. and Lee, K. D., "Patched Coordinate Systems", Numerical Grid Generation, Thompson, J. F. (Ed.), Elsevier Science Publishing Company, 1982.

8. Shaw, J., Forsey, C. R., Weatherhill, N. P. and Rose, K. E., "A Block Structured Mesh Generation Technique for Aircraft Geometries", Numerical Grid Generation in Computational Fluid Dynamics, Häuser, J. and Taylor, C. (Eds.), Pineridge Press Limited, 1986.

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

10. Sorenson, R. L., "Three-Dimensional Elliptic Grid Generation About Fighter Aircraft for Zonal Finite-Difference Computations", AIAA paper 86-0429, AIAA 24th Aerospace Sciences Meeting, 1986.

11. Eriksson, L. E., "Practical Three-Dimensional Mesh Generation Using Transfinite Interpolation", SIAM Journal on Scientific and Statistical Computing, Vol. 6, No. 3, 1985.

12. Eriksson, L. E., "Generation of Boundary-Conforming Grids Around Wing-Body Configurations Using Transfinite Interpolation", AIAA Journal, Vol. 20, No. 10, 1982.

13. Smith, R. E., "Algebraic Grid Generation", Numerical Grid Generation, Thompson, J. F. (Ed.), Elsevier Science Publishing Company, 1982.

14. Snepp, D. K. and Pomeroy, R. C., "A Geometry System for Aerodynamic Design", AIAA paper 87-2902, AIAA/AHS/ASSE Aircraft Design, Systems and Operation Meeting, 1987.

15. Edwards, T. A., "Definition and Verification of a Complex Aircraft for Aerodynamic Calculations", AIAA paper 86-0431, AIAA 24th Aerospace Sciences Meeting, 1986.

16. Kennon, S. R. and Dulikravich, G. S., "A Posteriori Optimization of Computational Grids" AIAA paper 85-0483, AIAA 23rd Aerospace Sciences Meeting, 1985.

17. Carcailett, R., "Optimization of Three-Dimensional Computational Grids and Generation of Flow Adaptive Computational Grids" AIAA paper 86-0156, AIAA 24th Aerospace Sciences Meeting, 1986.

18. Mastin, C. W., Soni, B. K. and McClure, M. D. "Experience in Grid Optimization" AIAA paper 87-0201 AIAA 25th Aerospace Sciences Meeting, 1987.

19. van der Vooren, J., van der Wees, A. J. and Meelker, J H., "MATRICS Preliminary Design Document", National Aerospace Laboratory NLR, The Netherlands, NLR TR 84076 L, 1985.