TCGRID 3-D Grid Generator for Turbomachinery

User's Manual and Documentation

Version 300, July, 2003

Dr. Rodrick V. Chima
NASA Glenn Research Center, MS 5-10
21000 Brookpark Road
Cleveland, OH 44135 USA

phone: 216-433-5919
fax: 216-433-5802
email: Rodrick.V.Chima@nasa.gov
internet: http://www.grc.nasa.gov/WWW/5810/rvc

Introduction

TCGRID (Turbomachinery C-GRID) is a three-dimensional grid generation code for turbomachinery blades. The code can generate single or multiblock grids that are compatible with various analysis codes including Swift and ADPAC. Single-block grids can be either C-type or H-type, and can be for linear cascades or annular blade rows. Multi-block grids must use a C-type grid around the blade, and can add an H-grid in the inlet region and O-grids in the hub or tip clearance regions.

A brief description of TCGRID and an example of a compressor grid are given in (1). Examples of turbine grids generated with TCGRID can be found in (2). Figures 1 here shows a multiblock H-C grid for a transonic compressor rotor (NASA rotor 37,) and figure 2 shows an H-grid for a transonic fan (NASA rotor 67.) The figures are used to describe the input variables later.

All geometry manipulation in TCGRID is done using parametric cubic splines, so the code can handle axial, mixed, and centrifugal flow machines. The input blade geometry can be translated, rescaled, and flipped tangentially, and full control of spacing along the blade surface is provided. Blade-to-blade grids are generated using an efficient elliptic solver that gives control of spacing and angles at the blade and outer periodic boundary. Grids are reclustered spanwise with control over spacing at the hub, casing, and clearance regions.

TCGRID is written completely in Fortran and runs as a quick batch job on unix or Windows computers. Code input is supplied as an ASCII dataset. Grid parameters are specified using convenient namelist input. Hub and casing geometries are specified as coordinate pairs. Blade shapes may be specified in MERIDL format, Crouse-Tweedt design code format, or specified directly by the user as coordinate triplets. Some printed output is provided. No graphical output is provided, but grid files can be read directly and plotted using the public domain CFD visualization codes PLOT3D and FAST**,** or the commercial codes FIELDVIEW and TECPOT.

This documentation briefly describes how the TCGRID code works. Instructions for dimensioning, compiling, and running TCGRID are given for Silicon Graphics (SGI) workstations and PC's. The namelist input variables and the hub, tip, and blade input are described in detail. Finally, an outline of the code structure given and the output file format is described.

Features of TCGRID

- **New in Version 300**
  Code distributed in tarred format that should compile and run directly on unix machines, or a zipped format that should compile and run directly on Windows PC's.
  Code converted to Fortran 90
  Dynamic memory allocation reduces memory requirements and avoids recompiling for most problems
  New algorithm for the meridional grid works for complicated axial-radial flow paths
  ADPAC compatible grid files
  Improved support for H-grids
  Note: For inlet H-grids, the dummy grid line at j=1 has been deleted for compatibility with Swift version 300. Thus inlet H-grids generated with TCGRID v. 300 are incompatible with earlier versions of Swift.

- **Applications**
  Linear cascades
  Axial compressors and turbines
  Isolated blade rows or multistage machines
  Centrifugal impellers and mixed-flow machines without splitters
  Radial diffusers
  Pumps
- **Grid Types**
  C- or H-grids around blades
  C-grid with upstream H-grid
  O-grids in hub- and tip-clearance regions
  Multistage machines can be modeled by merging C-grids for individual blades using a utility called MULTIX
- **Formulation**
  Blade-to-blade grids generated with a fast elliptic solver (GRAPE)
  Spanwise reclustering done using cubic splines
  H-grids in inlet regions and O-grids in clearance gaps generated algebraically
- **Input**
  Namelist input of grid parameters
  Hub and casing geometries input in (z, r) coordinates
  Blade geometries input in general (z, r, θ) coordinates, or MERIDL format (z, r, θ–upper,θ–lower) or (z, r, Δθ)
- **Printed Output**
  Input parameters
  Convergence information for blade-to-blade grids
  Spanwise output of inlet, leading edge, trailing edge, and exit coordinates
  Index file with grid size information for Swift
- **Grid & Debug Output**
  Grid files are written as binary data in standard PLOT3D format
  Intermediate grid files can be output for debug purposes. Debug files include reclustered blade coordinates, the 2-D throughflow grid, and 2-D blade-to-blade grids which may be useful for other purposes.
- **Computer Requirements**
  Runs as a quick batch process on most unix, linux, or Windows computers
  Fortran 90 compatible compiler
- **Graphical Output**
  No graphical output is provided with TCGRID, but access to some CFD visualization package is absolutely necessary to view and evaluate new grids. Grid files are in standard PLOT3D format and can be read directly and plotted with public-domain CFD visualization tools PLOT3D and FAST, or the commercial tools EnSight, FIELDVIEW, or TecPlot. Check the following web sites for more information.

PLOT3D & FAST:	http://www.nas.nasa.gov/Research/Software/swdescription.html
TecPlot:	http://www.amtec.com/
FIELDVIEW:	http://www.ilight.com/
EnSight:	http://www.mscsoftware.com.au/products/software/cei/ensight/

Getting Started – Unix

Unpacking

For unix systems TCGRID is supplied as a gzipped tar file called tcgrid.tgz. To open:

```
gunzip tcgrid.tgz
tar -xvf tcgrid.tar
```

This will create a directory called tcgrid.300 with subdirectories for the source, documentation, and test cases.

Compiling

Tcgrid must be compiled with a Fortran 90 compatible compiler. Edit modules.f90 and change the maximum array size if desired (see *Parameter Statement below.*) Edit Makefile and change the compiler name (f90) and options (OPTS) as necessary for your compiler.

```
cd tcgrid.300/src; make
```

The executable file tcgrid remains in the src directory.

Important Note: Modules.f90 contains several Fortran 90 modules that are used within other routines. It must be compiled before any other routines are compiled. The Makefile should take care of this automatically. If you ever need to compile manually, be sure to compile modules.f90 first. To delete the object files and executable,

```
make clean
```

Parameter Statement

TCGRID uses dynamic memory allocation for most arrays to avoid redimensioning for most problems. However, for programming convenience the maximum size of many work arrays are set using a Fortran module defined in modules.f90.

```
module param
!  maximum dimensions of any grid, used for work arrays
   integer,parameter::ni=321, nj=63, nk=65, nb=2*ni
end module param
```

This grid size (321 x 63 x 65) is large enough for most problems but can be increased to any size as needed.

Running TCGRID

To run the Goldman turbine vane test case:

```
cd tcgrid.300/gold
../src/tcgrid < gold.int
```

The grid should run in a few seconds. The output can be redirected to a file if desired.

Output Files

The output grid file is written to Fortran unit 1 (fort.1). Debug grid files may be written to fort.11 - fort.19, depending on idbg flags set in the input file. All grid files are written as unformatted binary files. A Swift index file may also be written to fort.10. The index file is in ASCII format and can be edited as desired. Output files should be renamed after running TCGRID,

```
mv fort.1 gold.xyz
mv fort.10 gold.ind
```

Alternative

Setting *iopen=1* in the input file causes all files to be opened explicitly with a default file name. For example, the grid will be named grid.xyz. Other file names are given with the *iopen* options under “&nam3 -Algorithm Parameters” on page 10. By using the default file names you can avoid the file renaming steps, but you may want to rename the files with more descriptive names later.

Binary grid files can be used immediately by Swift. Grid files can be read into PLOT3D using the *read /unformatted* option.

Getting Started – Windows

Unpacking

For Windows, TCGRID is supplied as a zipped file called tcgrid.zip. Use Winzip or PKZIP to unzip it. This will create a directory called tcgrid.300 with subdirectories for the source, documentation, and test cases.

Compiling

Tcgrid must be compiled with a Fortran 90 compatible compiler. Compaq Visual Fortran works well, but I don't have enough experience to give step-by-step instructions. Please let me know if you can adapt my Makefile or have any information about other compilers. In general, using Developer Studio:

Go to tcgrid.300/src
Open a new Fortran console application
Edit modules.f90 and change the maximum array size if desired (see *Parameter Statement below.*)
Set compile options for full optimization and no debug tables
Compile modules.f90 first
Build tcgrid

Important Note: Modules.f90 contains several Fortran 90 modules that are used within other routines. It must be compiled before any other routines are compiled. If you ever need to recompile, be sure to compile modules.f90 first.

Parameter Statement

TCGRID uses dynamic memory allocation for most arrays to avoid redimensioning for most problems. However, for programming convenience the maximum size of many work arrays are set using a Fortran module defined in modules.f90.

```
module param
!  maximum dimensions of any grid, used for work arrays
   integer,parameter::ni=321, nj=63, nk=65, nb=2*ni
end module param
```

This grid size (321 x 63 x 65) is large enough for most problems but can be increased to any size as needed.

Running TCGRID

To run the Goldman turbine vane test case, open a DOS window (Start/Programs/Accessories/Command prompt)

```
cd tcgrid.300\gold
```

Edit the TCGRID input file gold.int and set *iopen=1* in namelist 3. This will cause all files to be opened with the default names given. Now run TCGRID.

```
..\src/tcgrid < gold.int
```

The grid should run in a few seconds. The output can be redirected to a file if desired.

Output Files

The output grid file is written to grid.xyz. Debug grid files may be written to dedugnn.xyz, where nn=11-19, depending on idbg flags set in the input file. All grid files are written as unformatted binary files. A Swift index file may also be written to index.dat. The index file is in ASCII format and can be edited as desired.

Binary grid files can be used immediately by Swift. Grid files can be read into TecPlot using *File/Import/PLOT3D Loader* with the *unformatted* option.

TCGRID Input – Overview

TCGRID input consists of five blocks of namelist input (&nam1 - &nam5), followed by a title, then ASCII hub, tip, and blade coordinates. All grids require the title, hub, tip, and blade coordinate input, (see *Hub, Tip, and Blade Input, page 13*). Many of the namelist variables are initialized in modules.f90. Required input variables that are not initialized are listed below, followed by some comments regarding optional variables. All variables are described in detail in the section entitled *Namelist Input.*

C-grids

To generate the main C-grid around the blade set the following variables:

&nam1 - *im, jm, km, merid, itl, icap*.

To change clustering along the blade surface, along the span, and upstream and downstream of the blade, use *iclus*, *icluss*, and *iclusw*, respectively.

Use *iclus2d*=1 to cluster the meridional grid using the same spanwise clustering as the input blade. For simple, ruled blades use *iclus2d*=0.

For complex flow paths increase the meridional grid size using *i2d* and *k2d*.

&nam2 - *nle, nte, dsle, dste, dswte, dswex, dsmin, dsmax, dshub, dstip*.

Vary the leading- and trailing-edge radii with span using *dsthr.*

Move the location of the leading-edge clustering using *dsra*.

&nam3 - *iterm*.

For a quick check of a new grid, set *iterm*=0. Use PLOT3D to check leading- and trailing-edge spacings, surface clusterings, boundary locations, and outer boundary spacings.

If something goes wrong, use the array *idbg*(9) to generate debug grids to check input coordinates or intermediate grids (see *"TCGRID Code Description" on page 16.*)

Use *aabb* and *ccdd* to move points towards or away from the blade and outer boundary respectively.

&nam4 - Inlet and exit boundary coordinates *zbc* and *rbc* are required.

&nam5 - Most variables can defaulted.

Use *zscale, tscale, rscale, ztrans,* and *tflip* to manipulate the input blade coordinates.

Use *ioble*, *exl*, and *exr* to change the outer boundary shape.

Use *iwakex* and *jwakex* to stretch the wake grid.

H-grids

Most of the parameters required for C-grids are also required for H-grids. In addition, the following indices must be set:

&nam1 - *igch*=1, *ilh, itl*.

&nam2 - Spacing parameters are interpreted as follows:

dsin = spacing at inlet,

dsle = spacing at leading edge,

dste = spacing at trailing edge,

dswex = spacing at exit

dsmax is reset to *dsmin* internally.

&nam3 - Some algorithm control parameters are reset for consistency.

ccdd is reset to *aabb.*

omegpq is reset to *omegrs*.

Linear Cascades

To generate a grid for a linear cascade set the following:

&nam1 - *igeom* = 1,

&nam2 - *gap*.

Inlet H-grid

To generate an inlet H-grid, set the following:

&nam1 - *igin* = 1, *imi*.

&nam2 - dsin.

Since the inlet H-grid overlaps the blade C-grid, the C-grid must be run to convergence for proper H-grid spacings.

Tip Clearance O-grid

To generate a tip clearance grid, set the following:
&nam1 - *igclt* = 1, *jmt, km*t.
&nam2 - *cltip, dsclt*.

The number of points in the i-direction depends on the C-grid size. Since the clearance O-grid overlaps the blade C-grid, the C-grid must be run to convergence for proper O-grid spacings.

Hub Clearance O-grid

To generate a hub clearance grid, set the following:
&nam1 - *igclh* = 1, *jmh, kmh.*
&nam2 - *clhub, dsclh*.

The number of points in the i-direction depends on the C-grid size. Since the clearance O-grid overlaps the blade C-grid, the C-grid must be run to convergence for proper O-grid spacings.

Multistage Grids – Grid requirements

Multistage C-grids are generated one blade row at a time, then are merged using a utility called MULTIX. The individual grids must meet certain requirements:

1. The blades must be in the correct location and orientation. Use *ztrans* to move the blades axially, and *tflip* to flip the θ-coordinates if necessary.
2. All grids must have identical spanwise spacings. Use identical hub and tip inputs, and identical spanwise stretching and spacing parameters for all grids.
3. The grids must match at an interface between the blades. Set the exit boundary coordinates of grid 1 to the inlet boundary coordinates of grid 2. Place the interface midway between the blades, or close enough to blade 2 to get a good C-grid.
4. For Swift, the grids must overlap exactly one cell at the interface. On grid 1 set *dswex* to give a fine spacing near the exit, and set $dslap \approx dswex$. This resets the grid spacing at the exit from approximately *dswex* to exactly *dslap*. On grid 2 set $dsmax_2 = dslap_1$. This will cause the dummy grid line from grid 2 to overlap grid 1 by *dsmax*. (Not necessary for ADPAC.)
5. The relative circumferential spacing between the grids doesn't matter.

Multistage Grids – MULTIX Utility

MULTIX simply merges two PLOT3D files and two index files without much checking. The steps are as follows:

1. Run grid 1. Rename the grid file from fort.1 to name1.xyz. Rename the index file from fort.10 to name1.ind.
2. Run grid 2. Rename the grid file from fort.1 to name2.xyz. Rename the index file from fort.10 to name2.ind.
3. Compile multix.f and run it. It will prompt for name1 and name2.
4. The output files are named out.xyz and out.ind. You can rename them and add more files if necessary.
5. The index file out.ind will have the correct format and grid sizes, but you will have to edit it and change connectivity, boundary condition flags, etc. as described in the Swift documentation.

Namelist Input

Defaults are given in angle brackets, <default=value> or <default.> If no default is given the value MUST be input. Relevant figures are given in parentheses (fig. #.)

***&nam1* - Grid Size Parameters**

Many grid size parameters are illustrated in figures 1 – 4.

merid Flag for type of blade input. See *Hub, Tip, and Blade Input,* page 13, and figures 5 and 6 for complete descriptions of the blade input formats.

= 0 Blade input in stacked sections, (z, r, θ). Completely general, (fig. 5), <default.>

= 1 Blade input in Crouse/Tweedt design code format, (z, r, θ) (fig. 5.) Like *merid*=1 but ordered differently.

= 2 Blade input in MERIDL format, (z, r, θ-upper, θ-lower), (fig. 6).

= 3 Blade input in MERIDL format, (z, r, θ, Δθ)

im Grid size in i- (streamwise) direction, (figs. 1, 2.)

jm Grid size in j- (blade-to-blade) direction, (figs. 1, 2.)

km Grid size in k- (spanwise) direction, (fig. 3.)

itl C-grid: i-index of lower trailing-edge point, (fig. 1.)

H-grid: number of cells downstream of the trailing-edge, (fig. 2.)

icap Number of cells on the inlet part of the C-grid, equally-spaced. Remaining cells are distributed over the periodic boundaries. Increase *icap* to pull points towards inlet, and vice-versa, (fig. 1.)

igeom Flag that tells whether grid will be for a linear cascade or an annular blade row.

= 0 Annular blade row <default.>

= 1 Linear cascade.

iclus Flag for type of clustering along the blade surfaces.

= 1 Hyperbolic tangent clustering - smoothest, but may be sparse at blade center if *im* is small <default.>

= 2 Hermite polynomial clustering - more uniform, but may grow too quickly near leading and trailing edges. Good for large *im*.

icluss Same as *iclus*, but for clustering in the spanwise direction.

iclusw Same as *iclus*, but for streamwise clustering downstream in the wake, and also upstream for an H-grid.

iclus2d Flag that sets spanwise clustering of the meridional grid on which blade-to-blade grids are generated.

= 0 Meridional grid is equally spaced between hub to tip. Good for ruled blades,

= 1 Meridional grid has approximately the same spanwise clustering as the input blade sections. Use this option if the input blade sections resolves spanwise geometry variations, e.g., fillets <default.>

i2d Number of i- (streamwise) points on the coarse meridional grid used to define the passage. Typically 21 for an axial machine, 41 or more for a centrifugal. <default = 21, max=101.>

k2d Number of k- (spanwise) points on the coarse meridional grid used to define the passage. Should be roughly equal to the number of input blade sections *nbs*. <default = 11, max=50.>

Parameters for H-grids Around Blades

igch Flag to set C- or H-grids around blade

= 0 C-grid <default.>

= 1 H-grid.

ilh H-grid: number of cells upstream of the leading edge, (fig. 2.)

itl H-grid: number of cells downstream of the trailing-edge, (fig. 2.)

Parameters for Inlet H-grids

igin Flag to generate inlet H-grid ahead of main C-grid. Not used if *igch* = 1.

= 0 No inlet H-grid <default.>

= 1 Generate inlet H-grid.

imi Number of i- (streamwise) points in the inlet H-grid. Only used if *igin* = 1.

Parameters for Clearance Gap O-grids

igclt Flag to generate tip clearance O-grid.

= 0 No tip clearance O-grid <default.>

= 1 Generate tip clearance O-grid.

jmt Number of j- (radial) points in the tip clearance grid, (fig. 4.). Only used if *igclt* = 1.

kmt Number of k- (spanwise) points in the tip clearance grid, (fig. 4.) Only used if *igclt* = 1.

igclh Flag to generate hub clearance O-grid.

= 0 No hub clearance O-grid <default.>

= 1 Generate hub clearance O-grid.

jmh Number of j- (radial) points in the hub clearance grid. Only used if *igclh* = 1.

kmh Number of k- (spanwise) points in the hub clearance grid, (fig. 4.) Only used if *igclh* = 1.

***&nam2* - Grid Spacing Parameters**

All spacing parameters must be input in the units desired for the final grid. All spacing parameters named "*ds*…" refer to spacing along some arc length, and not in a particular coordinate direction. Values suggested as "e.g." should give a good initial guess but may need to be modified after examining the initial grid.

nle Number of points equally-spaced around the blade leading edge, typically 15.

nte Number of points equally-spaced around the blade trailing edge, typically 10.

dsle Spacing around the leading edge at the hub, e.g. $\pi \cdot rle/nle$.

dste Spacing around the trailing edge at the hub, e.g. $\pi \cdot rte/nte$.

dsthr "ds tip-to-hub ratio." *Dsle*, *dste*, and *dswte* are taken as hub values and are varied linearly with span to this factor at the tip. Allows the leading edge radius, etc. to increase or decrease (usually decrease) with span. <default = 1.>

dswte Spacing away from the trailing edge on the wake cut of C-grid, should be ≈ *dste*, (fig. 1.)

dswex Spacing at exit on wake cut of C-grid, hard to estimate in advance. Should be roughly the spacing along the periodic boundary, which is roughly the streamwise distance from the inlet to the exit divided by (*im*-*icap*)/2, (fig. 1.)

dsmin Spacing away from the blade, e.g. chord/10,000 for viscous grids, (fig. 1, 2.) Note: Swift's blade surface output gives y+, the grid spacing at the wall in turbulent wall units. y+ should be $O(1-5)$. If it is too large you may need to rerun your grid with *dsmin*, *dshub*, and *dstip* reduced accordingly.

dsmax Spacing away from the periodic boundary, e.g. midspan-pitch/*jm*, (fig. 1, 2.)

dsin Spacing away from the inlet of inlet H-grid, (fig. 1, 2.) Only used if *igin* = 1 or *igch* = 1.

dsra (Pressure surface arc length)/(total surface arc length). Used to locate the center of the leading edge clustering on the blade. The clustering is centered about *dsra* x (total surface arc length.) Typical values are 0.5 for symmetrical blades, about 0.49 for compressor blades, and about 0.45 for highly-cambered turbine blades. Only used for general blade input, *merid* = 0.

= 0 TCGRID assumes that there are the same number of blade input coordinates on each surface and clusters about the median input point <default.>

gap Blade row pitch for a linear cascade. Only used if *igeom* = 1. If *igeom* = 0 the pitch is set by *nblade*. <default = 1.>

rcorn Radius for the front corner of the C-grid, (fig. 1.) Usually 0., but the inlet may be smoother with $rcorn \approx$ pitch/8 . <default = 0. Reset to 0. if *igin* = 1.>

Spanwise Spacing Parameters

The spanwise grid is clustered in $1-3$ regions as shown in fig. 4. The stretching function in each region is set by *icluss*. A central region is always used between the hub and casing, with wall spacings *dshub* and *dstip*.

If $cltip > 0$ a tip clearance region is added, with wall spacing *dstip*, height = *cltip*, and interface spacing *dsclt*.

If $clhub > 0$ a hub clearance region is added, with wall spacing *dshub*, height = *clhub*, and interface spacing *dsclh*.

Note that these regions are independent of any clearance region O-grids that are added if $igclt = 1$ or $igclh = 1$. In other words, these stretching functions can be used to cluster points near the endwalls without adding O-grids.

dstip Spanwise spacing at the tip, e.g. span/10,000 for viscous grid. Should be ≈ *dshub*, (fig. 4.)

cltip Tip clearance height, (fig. 4.) Used if *igclt* = 1, or to cluster tip grid for a simple periodicity clearance model.

= 0 Grid is stretched continuously away from the casing. <default.>

> 0 Grid is clustered near the casing using *dstip*, *cltip*, and *dsclt*.

dsclt Spanwise spacing at the blade/tip clearance interface, (fig. 4.) Only used if *cltip* > 0. <default = *dstip*.>

dsclh Spanwise spacing at the blade/hub clearance interface, (fig. 4.) Only used if *clhub* > 0. <default = *dshub*.>

clhub Hub clearance, (fig. 4.) Used if *igclh* = 1, or to cluster hub grid for a simple periodicity clearance model.

= 0 Grid is stretched continuously away from the hub <default.>

> 0 Grid is clustered near the hub using *dshub*, *clhub*, and *dsclh*.

dshub Spanwise spacing at the hub, e.g. span/10,000 for viscous grids, (fig. 4.)

Parameters for Blunt Trailing Edges

TCGRID can wrap grids around blades with blunt trailing edges like the blades shown in figure 7. Blade coordinates may be input in one of two ways depending on the value of *merid*.

merid = 0 or 1 (general coordinate input) <default.> Blades must be input with an open trailing edge (fig. 7a,) and *nbase* points are added automatically.

= 2 or 3 (MERIDL input.) Blades are always input with an open trailing edge (fig. 6,) then

if *nbase* = 0 a round trailing edge is added <default.>

if *nbase* > 0 a blunt trailing edge is assumed.

nbase Number of intervals on the blunt trailing edge (fig. 7a.)

ibase Flag controlling the location of the wake cut line with respect to the base of the blade (fig. 7a.) To minimize grid distortion, choose *ibase* such that the cut leaves the corner with the acute angle. If the base is symmetric (fig. 7c, d), use *ibase* = 0.

= +1 Cut line leaves the upper corner of the base.

= 0 Cut line leaves the center of the base <default.>

= -1 Cut line leaves the lower corner of the base.

ibevel Flag for beveling the corner(s) of the base to reduce grid distortion (fig 7c, d.) If *ibase* = 0 both corners are beveled. If *ibase* = 1 the corner opposite the wake cut is beveled.

= 0 No bevel <default.>

= 1 Corner(s) are beveled.

***&nam3* -Algorithm Parameters**

See Sorenson's GRAPE code documentation (5) for more information on algorithm parameters. Most values can be defaulted.

iterm Number of iterations for elliptic solver, usually 50 – 150. Use *iterm*=0 to check initial grid spacings, boundary locations, etc. <default=100.>

idbg(9) Integer flag array with nine elements for writing intermediate debug grids to Fortran units 11-19. Useful for debug, graphics, and possibly for grid generation in itself. For more information see Table 1 on page 18. <default = 9*0.>

omega Relaxation factor for the elliptic solver, rarely changed. Acceptable values from 0. to 2. <default = 1.4.>

omegpq Relaxation factor for the inner boundary forcing functions, rarely changed. Acceptable values from 0. to 2. Set to 0. for a Laplacian inner boundary. <default = 0.1.>

omegrs Like omegpq, but for the outer boundary.

aabb Exponent controlling the distance that angles and spacings at the inner boundary propagate into the interior. Small aabb give large distances but slow convergence, and vice versa. Any value > 0. is acceptable. <default = 0.45> Set $aabb = 0.35$ to cluster more points near the wall, or $aabb = 0.55 - 0.65$ to reduce the clustering.

ccdd Like aabb, but for the outer boundary.

csmoo Smoothing coefficient for the periodic boundary. If $csmoo > 0$, a 4th-difference smoothing operator is applied to the periodic boundary. This allows the boundary points to float a little, which may reduce distortion. Not usually needed, but worth a try if the periodic boundary looks bad. Acceptable values from 0. to 1, typically 0.5. <default=0.>

iopen Flag for opening output files explicitly by name.

= 0 Output files are written to Fortran units without explicitly opening them. <default.>

= 1 Output files are opened by name:

grid.xyz = main grid file (binary)

index.dat = Swift index file (ASCII)

debug*nn*.xyz = debug grid files. *nn* = 11 – 19 refer to the Fortran unit number in Table 1 on page 18.

***&nam4* - Boundary Coordinates**

zbc(3,2) and *rbc*(3,2)

Arrays of (z, r) coordinates that define three line segments that act as the upstream H-grid inlet, the blade C-grid inlet, and the blade C-grid exit. The line segments are intersected with the hub and tip geometry, so the coordinates need not lie exactly on the hub and tip. Anywhere nearby should work. Figure 3 illustrates the locations of the boundary coordinate points. The first index indicates the segment and the second index indicates hub or tip. The six points must be entered in the order shown below. The coordinates of the upstream H-grid inlet may be set to zero if *igin* = 0.

zbc = z-H-hub-in, z-C-hub-in, z-C-hub-ex, z-H-tip-in, z-C-tip-in, z-C-tip-ex

rbc = r-H-hub-in, r-C-hub-in, r-C-hub-ex, r-H-tip-in, r-C-tip-in, r-C-tip-ex

***&nam5* - Miscellaneous Parameters**

Most of these parameters can be defaulted.

iswift Flag to set grid file format for RVC3D, Swift, or ADPAC.

= 0 RVC3D code format – no dummy grid line. Note: RVC3D only uses single block C-grids. <default.>

= 1 Swift code output – one or more blocks with dummy grid lines.

= 2 ADPAC code output – one or more blocks, no dummy grid lines. Note: For ADPAC the j- and k-directions are swapped, i.e., for x(i, j, k), j is the spanwise direction and k is the blade-to-blade direction.

Scaling Parameters

Parameters for rescaling, translating and flipping the input blade coordinates.

zscale Scale factor for blade z-coordinates, <default = 1.>

tscale Scale factor for blade θ-coordinates, <default = 1.>

rscale Scale factor for blade r-coordinates, <default = 1.>

ztrans Translation distance for blade z-coordinates, <default = 0.>

tflip Flag for flipping the blade in the θ-direction and reordering the points.

= 0 Do not flip blade θ-coordinates, <default.>

= 1 Flip blade θ-coordinates.

Outer Boundary Shape Control Parameters

dslap i- (streamwise) spacing at exit of C-grid. If *dslap* > 0, the grid lines at i = 2 and i = im-1 are repositioned exactly *dslap* from the exit, overriding *dswex*. For multistage machines the next blade row should have *dsmax* = *dslap* to give a perfect overlap of the grids.

exl Controls the shape of the left (upstream) periodic boundary of a C-grid. The boundary starts tangent to the mean camber line and curves to axial at a rate determined by *exl*.

> 10 No curvature – the boundary is a linear extension of the mean camber line.

> 1.5 Slow curvature to axial.

= 1.5 Moderate curvature to axial <default.>

< 1.5 Fast curvature to axial.

= 1 Turns the boundary abruptly to axial.

exr Controls the shape of the right (downstream) periodic boundary of a C-grid. The boundary starts tangent to the mean camber line and curves to axial at a rate determined by *exr.*

> 10 No curvature – the boundary is a linear extension of the mean camber line <default.>

> 1.5 Slow curvature to axial.

= 1.5 Moderate curvature to axial.

< 1.5 Fast curvature to axial.

= 1 Turns the boundary abruptly to axial.

fswake Fractional distance along the downstream periodic boundary between the trailing edge and the grid exit, where the j-grid lines from the trailing edge *(i = itl)* intersect the outer boundary, (fig. 1). The default value of <1.> places the outer boundary point directly above and below the trailing edge point. On some blades this can cause the j-grid lines to cross the trailing edge. In this case try setting $fswake < 1.0$ to pull the grid lines towards the downstream boundary.

ioble The periodic outer boundary for a C-grid is made up of three segments, an upstream segment, the mean-camber line between the blades, and a downstream segment. The parameter *ioble* is an index which determines where the upstream segment joins the mean camber line. Values can be <11>, 10, 9 … The default <11> starts the upstream segment at the leading edge. Smaller values move the starting point inside the passage, which can be useful if upstream part of the C-grid becomes distorted due to stagger, (fig. 1).

iwakex Flag for stretching the outer boundary grid spacing along the wake (i-direction).

= 0 Equally-spaced outer boundary along the wake.

= 1 Stretched outer boundary along the wake, <default.>

jwakex Flag for controlling the j-direction spacing along the trailing edge cut.

= 0 j-grid spacing is *dsmin* all along the trailing edge cut, <default.>

= 1 j-grid spacing expands from *dsmin* at the trailing edge to equally-spaced at the exit.

Hub, Tip, and Blade Input

Immediately following the namelist input are unformatted reads for a title, the hub and tip coordinates, and the blade coordinates. Unformatted ASCII reads are used throughout.

Title

A title of ≤ 80 characters or less is read using the following Fortran input statement:

```
read *,ititle
```

ititle An alphanumeric string of 80 characters printed to the output. The character string must be enclosed in single quotes.

Hub and Tip Geometry

Hub and tip coordinate arrays as shown in figure 3 are read in as follows:

```
!     read hub & tip geometry
      read(5,*)nph,npt
      read(5,*)(zhub(i),i=1,nph)
      read(5,*)(rhub(i),i=1,nph)
      read(5,*)(ztip(i),i=1,npt)
      read(5,*)(rtip(i),i=1,npt)
```

nph Number of input hub points, min = 2, max = 321.

npt Number of input tip points, min = 2, max = 321.

zhub, rhub z, r coordinates of the hub.

ztip, rtip z, r coordinates of the tip.

Blade Geometry

The next line of input contains three variables read as follows:

```
!     blade input
      read(5,*)nbs,npb,nblade
```

nbs Number of blade sections, max. = 50.

npb Number of points around the blade, max. = 321.

nblade Number of blades around the wheel, used to determine the pitch.

This is followed by blade coordinates in one of four formats set by the input value of *merid*. The coordinates need not intersect the hub and tip coordinates – they are spline fit if they span the endwalls, or are linearly extrapolated if they do not, and the intersections are calculated by TCGRID. The four input options and their corresponding Fortran reads are as follows:

merid = 0 <default>

Blade input in general stacked sections. Cylindrical coordinates starting at the blade trailing edge, wrapping clockwise around the blade, and repeating the trailing-edge point. Complete definition of the leading- and trailing-edges must be given. Sections are stacked from hub to tip. Figure 5 shows the ordering of points for *merid* = 0.

```
!     merid=0: blade input in stacked sections, cyl. coords.
      if(merid == 0)then
      do k=1,nbs
        read(5,*)(zb(i,k),i=1,npb)
        read(5,*)(yb(i,k),i=1,npb)
        read(5,*)(rb(i,k),i=1,npb)
      enddo
      endif
```

Here (zb, yb, rb) = (z, θ, r)-coordinates of the blade section.

merid = 1

Blade input in Crouse/Tweedt design code format. See Tweedt's writeup on design code options. The point ordering around the blade is the same as for *merid* = 0, as shown in figure 5, but the points are stacked from tip to hub.

```
!     merid=1: blade input in Crouse/Tweedt design code format
      if(merid == 1)then
      read(5,*)zbhub
      do k=nbs,1,-1
        read(5,*)dum
        read(5,*)dum
        read(5,*)(zb(i,k),i=1,npb)
        read(5,*)(yb(i,k),i=1,npb)
        read(5,*)(rb(i,k),i=1,npb)
      enddo
      endif
```

Again (zb, yb, rb) = (z, θ, r)-coordinates of the blade section.

zbhub A z-translation value added to all blade z-coordinates, to shift them to the same reference as the hub and tip. Can also be done using namelist variable *ztrans*.

dum Two dummy records are included before each blade section.

merid = 2 or 3

Blade input in MERIDL format. See Katsanis and McNally's report on MERIDL (3) for more information on MERIDL input. Unlike MERIDL, TCGRID can handle purely radial flows without rotating the coordinate system. MERIDL input has no leading- or trailing-edge definition, but TCGRID will add leading- and trailing-edge circles automatically. Points are input from leading edge to trailing edge, and from hub to tip.

```
!     merid=2 or 3: blade input in MERIDL format
      if(merid > 1)then
      do k=1,nbs
        read(5,*)(  zbl(i,k),i=1,npb)
      enddo
      do k=1,nbs
        read(5,*)(  rbl(i,k),i=1,npb)
      enddo
      do k=1,nbs
        read(5,*)(th1bl(i,k),i=1,npb)
      enddo
      do k=1,nbs
        read(5,*)(th2bl(i,k),i=1,npb)
      enddo
      endif
```

merid = 2 (zbl, rbl, th1bl, th2bl)= (z, r, θ-upper-surface, θ-lower-surface) coordinates of the blade section, as shown in figure 6.

merid = 3 (zbl, rbl, th1bl) = (z, r, θ)-coordinates of the mean-camber-line, ordered like *merid* = 2 (fig. 6),

th2bl = blade tangential thickness (θ-upper-surface – θ-lower-surface).

Grid Output *XYZ*-File

Grids are stored using standard PLOT3D xyz-file structure. Single block grids can be read with the following code:

```
      read(1)im,jm,km                          !grid dimensions
!     read grid coordinates
      read(1)
     &(((x(i,j,k),i=1,im),j=1,jm),k=1,km),
     &(((y(i,j,k),i=1,im),j=1,jm),k=1,km),
     &(((z(i,j,k),i=1,im),j=1,jm),k=1,km)
```

Multiblock grids can be read with the following code:

```
      integer,dimension(3,10)::idx

      read(1)ngrid                             !number of grids
      read(1)((idx(l,ng),l=1,3),ng=1,ngrid)    !grid dimensions
!     loop over the grids
      do ng=1,ngrid
        im=idx(1,ng)
        jm=idx(2,ng)
        km=idx(3,ng)

        read(1)
     &  (((x(i,j,k),i=1,im),j=1,jm),k=1,km),
     &  (((y(i,j,k),i=1,im),j=1,jm),k=1,km),
     &  (((z(i,j,k),i=1,im),j=1,jm),k=1,km)
      enddo
```

TCGRID Code Description

TCGRID is based on an old, 2-D version of the Steger/Sorenson GRAPE code (5, 6). The GRAPE code is used to generate blade-to-blade grids on several surfaces of revolution between the hub and casing. The individual blade-to-blade grids are stacked and reclustered spanwise to form the 3-D grid. Additional inlet or clearance blocks are then generated algebraically.

Grids are generated in several sequential steps. Most steps are coded as separate subroutines, many of which can generate PLOT3D compatible grid files for debugging purposes. The outline below lists the main subroutines of TCGRID in the order that they are called, describes their function, and describes the debug flag and debug output generated. Indentation implies subroutine nesting.

Debug files are requested using the namelist array *idbg*(9) described in Table 1 on page 18. In general, if $idbg(n) = 1$, a debug grid file will be generated on Fortran unit $n + 10$. Grids are in PLOT3D format, may be 2-D or 3-D, and may be in multigrid format. If there is a problem with TCGRID, set *idbg* = 9*1 and plot fort.11 – fort.19 in turn. (Some files may not be generated depending on input options.) Refer to the outline below to determine at which step the problem occurred.

tcgrid.f

Main calling program.

1. setup.f
   Reads the namelist, hub, tip, and blade input. Hub and casing geometries are input as arrays of (z,r) coordinates. Blade geometries are input as arrays of (z,r,θ) coordinates. Several blade input formats are supported. The blade coordinates can be scaled and translated if desired.
   Output file: *idbg*(1) = 1, unit 11, 3-D blade as input (*merid* = 0 or 1.)
   Output file: *idbg*(1) = 1, unit 11, 3-D blade after addition of leading- and trailing-edge circles (*merid* = 2 or 3.)
   openfile.f - Opens files by name if $iopen = 1$.

2. inner.f (*merid* = 0 or 1)
   Reclusters points around the blade sections. The leading and trailing edges are evenly-spaced. The trailing-edge spacing is centered around the first blade input point (which is repeated as the last input point unless $nbase > 0$.) The leading-edge spacing is centered around some fraction of arc length specified using *dsra*. If $dsra = 0$, the leading-edge spacing is centered on the median input point. The blade surfaces are reclustered between the leading and trailing edges using Hermite polynomials or hyperbolic tangent clustering. Inner.f also adds points on the base if $nbase > 0$.

3. merfix.f (*merid* = 2 or 3)
   Converts MERIDL blade sections to GRAPE-type sections. Adds leading- and trailing-edge circles. Adds points on the base if $nbase > 0$. Reclusters points around the blade sections.
   Output file: *idbg*(2) = 1, unit 12, 3-D multigrid MERIDL blade as input (*merid* = 2 or 3).
   lete.f - Computes leading- and trailing-edge circles for the MERIDL blades using the technique described in (4).

4. addht.f
   Adds inlet and exit points to hub and tip arrays.

5. meridg.f
   Generates a coarse meridional grid between the hub and casing. The grid is generated algebraically by connecting corresponding points on the hub and casing. The number of spanwise points should be about the same as the number of input blade sections. If $iclus2d = 0$ the meridional grid is equally-spaced spanwise. This option is good for simple ruled blades or blades defined on equally-spaced stream surfaces. If $iclus2d = 1$ the meridional grid is generated with approximately the same spanwise clustering as the input blade sections.
   Output file: *idbg*(3) = 1, unit 13, 2-D meridional grid between hub and tip.

6. blades.f
   Intersects spanwise lines between blade sections with the streamwise lines of the meridional grid. This gives new blade sections on the stream surfaces defined by the meridional grid.
   Output file: *idbg*(4) = 1, unit 14, 3-D blade after interpolation onto meridional grid.

7. grid2d.f
   Generates 2-D blade-to-blade grids along the meridional grid lines using an old version of the Steger/Sorenson GRAPE code (refs. 5 and 6). GRAPE is strictly 2-D, so the 3-D blade sections are mapped to a 2-D $(m, \bar{r} \times \theta)$ system, where m is the meridional coordinate defined by $dm^2 = dz^2 + dr^2$ and r is the mean radius.

   GRAPE requires the specification of an inner and outer boundary. For a C-grid the inner boundary is defined by the mapped blade coordinates plus a polynomial wake cut. A periodic outer boundary is defined using the mean camber line inside the passage, with polynomial extensions up- and downstream. For an H-grid the boundaries are defined by the blade surfaces and polynomial extensions up- and downstream. Angles and spacings are specified on the boundaries. The angles are set to give normal grid lines at the blade and inlet, and vertical grid lines at the periodic boundaries. The spacings are input using *dsmin* and *dsmax*.
   outc.f - Generates the periodic boundary for a C-grid
   outh.f - Generates the periodic boundary for a H-grid

   An initial grid is generated algebraically. GRAPE uses an SLOR scheme to solve the Poisson equations, and is typically iterated 100 iterations. Dummy grid lines may be added algebraically.
   grelax.f - Solves the elliptic grid equations.
   dumgl.f - Adds dummy grid lines if *iswift* = 1.
   Output file: *idbg*(5) = k, unit 15, 2-D blade-to-blade grid on surface k of the meridional grid.

   The 2-D, $(m, r \times \theta)$ grid is transformed back to (z, r, θ).
   Output file: *idbg*(6) = 1, unit 16, 3-D grid before spanwise clustering.

8. fill3d.f
   Reclusters the 2-D grids spanwise using either Hermite polynomials or hyperbolic tangent clustering to make a full 3-D grid. Separate clustering functions are used in the hub clearance, blade, and tip clearance regions.

9. ginlet.f
   Generates an algebraic H-grid block upstream of the blade if $igin = 1$. The boundaries are defined by the hub and casing, and the H-grid and C-grid inlet boundaries. Transfinite interpolation (7) is used to fill in the interior points. The resulting meridional grid is swept tangentially.
   Output file: *idbg*(7) = 1, unit 17, 3-D inlet H-grid.

10. gtip.f
    Generates an algebraic O-grid block in the tip clearance region if $igclt = 1$.
    Output file: i*dbg*(8) = 1, unit 18, 3-D tip clearance O-grid.
    Generates an algebraic O-grid block in the hub clearance region if $igclh = 1$.
    Output file: i*dbg*(9) = 1, unit 19, 3-D hub clearance O-grid.

11. ospan.f
    Prints the inlet, leading edge, trailing edge, and exit coordinates.

12. outmg.f
    Transforms (z, r, θ) to (*x*,*y*,*z*) and writes the grid file on unit 1 in PLOT3D format.
    Output file: unit 1, 3-D grid.
    Output file: unit 10, ASCII index file for use by the Swift code. Contains grid sizes and indices that may be needed for other codes.

i	Value	Unit	Subroutine	Grid Size	Plot3d Format	Grid Description
1	1	11	Input	npb, nbs, 1	3d/unf	General blade as input (*merid* = 0, 1) MERIDL blade after l.e. & t.e. addition (*merid* = 2, 3)
2	1	12	Merfix	nbs, npb, 1	3d/unf/mg	MERIDL blade as input (*merid* = 2, 3)
3	1	13	Meridg	i2d, k2d	2d/unf	2-D meridional grid between hub and tip
4	1	14	Blades	npb, k2d, 1	3d/unf	3-D blade after interpolation onto 2-D grid
5	k	15	Grid2d	im, jm, 1	2d/unf	2-D blade-to-blade grid on section k
6	1	16	Grid2d	im, km,k2d	3d/unf	3-D rid before spanwise clustering
7	1	17	Ginlet	imi, jmi, kmi	3d/unf	3-D inlet H-grid
8	1	18	Gtip	imt, jmt, kmt	3d/unf	3-D hub clearance O-grid
9	1	19	Gtip	imh,jmh,kmh	3d/unf	3-D tip clearance O-grid

Table 1 — Debug grid files available with *idbg* options.

Acknowledgments

Joe Steger and Reece Sorenson developed and supplied the original GRAPE code that serves as the basic grid solver in TCGRID. Kevin Kirtley developed an early version of TCGRID called GRD3D. Larry Schuman provided the algorithm for adding leading edge circles to MERIDL blades. Dan Tweedt developed the spline routines used in TCGRID and provided many helpful comments and suggestions. Dave Miller provided the transfinite interpolation routine used for the upstream grid block.

References

1. Chima, R. V., “Viscous Three-Dimensional Calculations of Transonic Fan Performance,” in *CFD Techniques for Propulsion Applications*, AGARD Conference Proceedings No. CP-510, AGARD, Neuilly-Sur-Seine, France, Feb. 1992, pp 21-1 to 21-19. Also NASA TM-103800.

2. Chima, R. V., Giel, P. W., and Boyle, R. J., “An Algebraic Turbulence Model for Three-Dimensional Viscous Flows,” in *Engineering Turbulence Modelling and Experiments 2*, Rodi, W. and Martelli, F. editors, Elsevier pub. N. Y., 1993, pp. 775-784. Also NASA TM-105931.

3. Katsanis, T., McNally, W. D., “Revised Fortran Program for Calculating Velocities and Streamlines on the Hub-Shroud Midchannel Stream Surface of an Axial-, Radial-, or Mixed-Flow Turbomachine or Annular Duct,” NASA TN D-8430, Mar. 1977.

4. Schumann, L. F., “Fortran Program for Calculating Leading-and Trailing-Edge Geometry of Turbomachine Blades,” NASA TM X-73679, June, 1977.

5. Sorenson, R. L., “A Computer Program to Generate Two-Dimensional Grids About Airfoils and Other Shapes by Use of Poisson’s Equation,” NASA TM-81198, 1980.

6. Steger, J. L., and Sorenson, R. L. “Automatic Mesh Point Clustering Near A Boundary in Grid Generation with Elliptic Partial Differential Equations,” Journal of Computational Physics, Vol.33, No. 3, Dec. 1979, pp.405-410.

7. Thompson, J. F., Warsi, Z. U. A., Mastin, C. W., *Numerical Grid Generation Foundations and Applications*, North-Holland, N. Y., 1985.

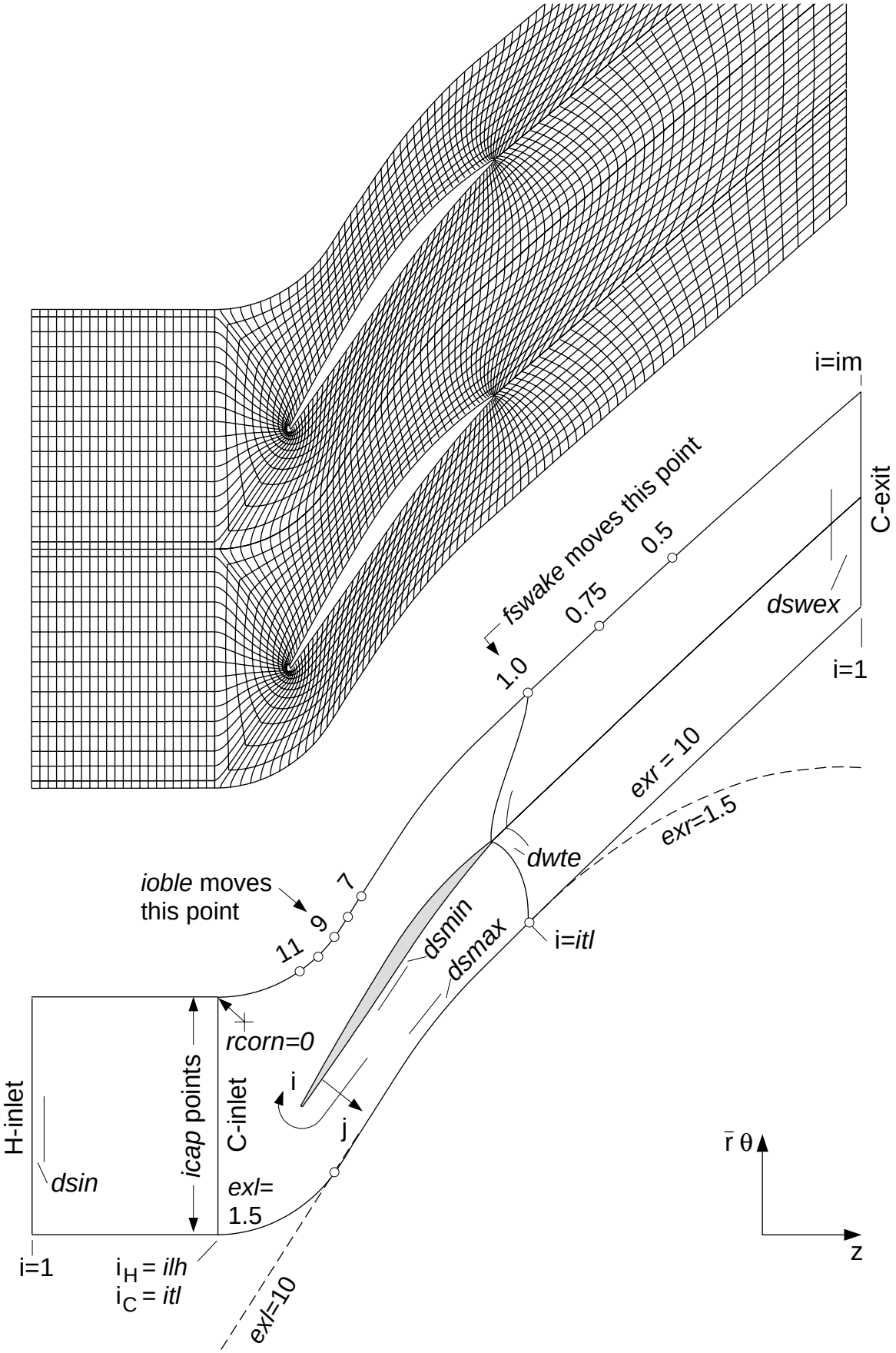


Figure 1 — Top: Blade-to-blade H–C grid for a transonic compressor rotor.
Bottom: TCGRID nomenclature and input variables for blade-to-blade grid.

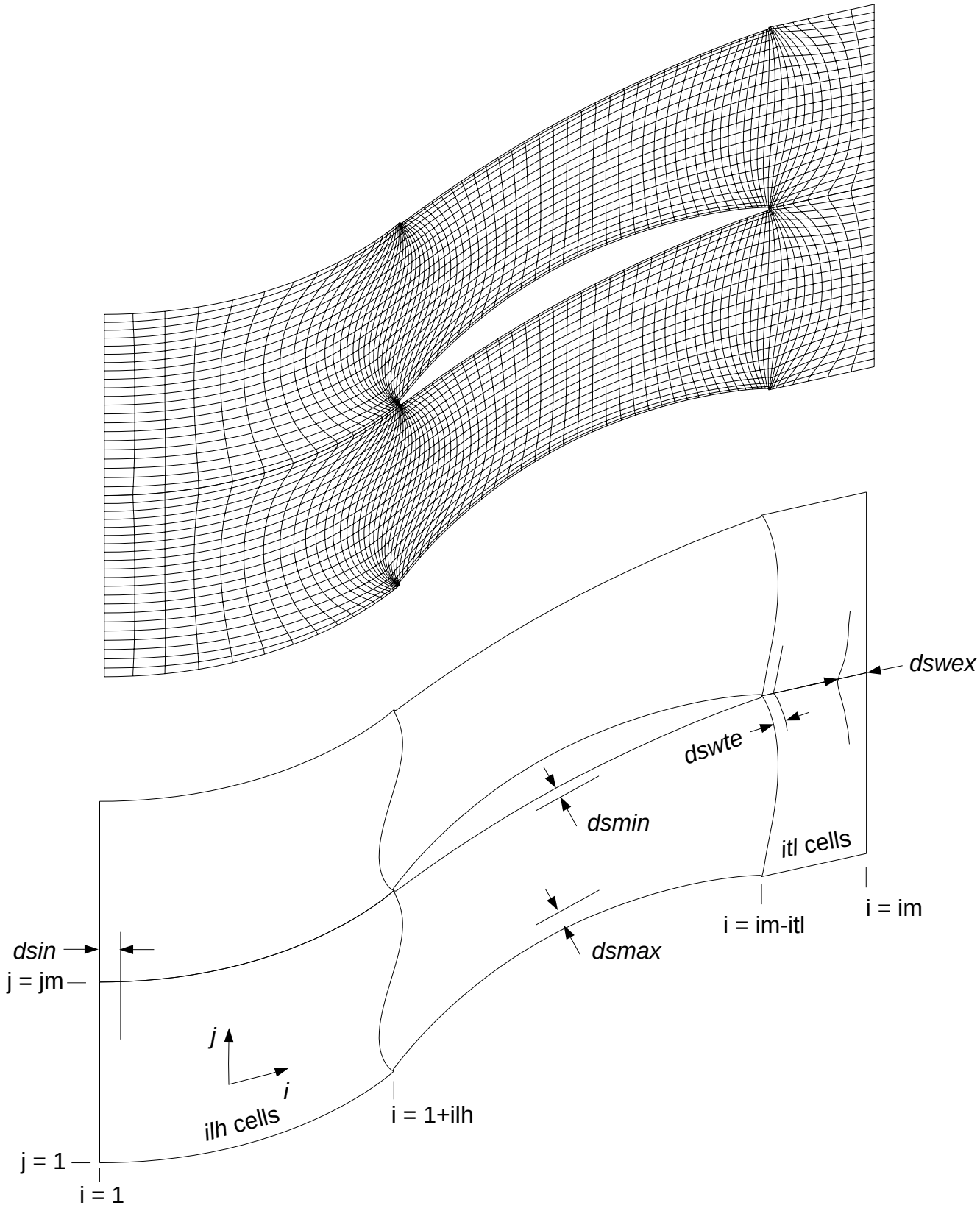


Figure 2 — Top: Blade-to-blade H-grid for a compressor rotor hub section.
Bottom: TCGRID nomenclature and input variables for H-grids.

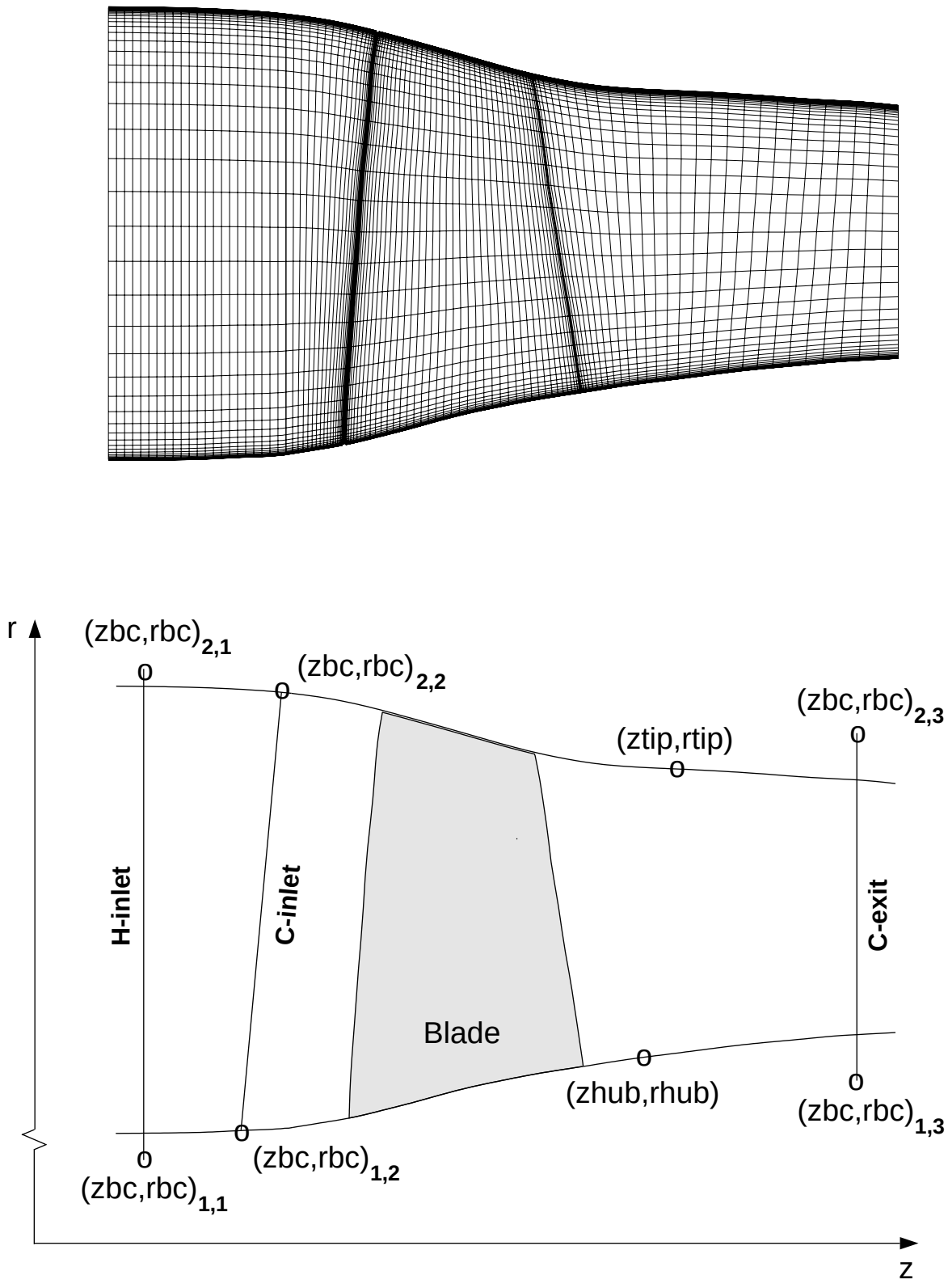
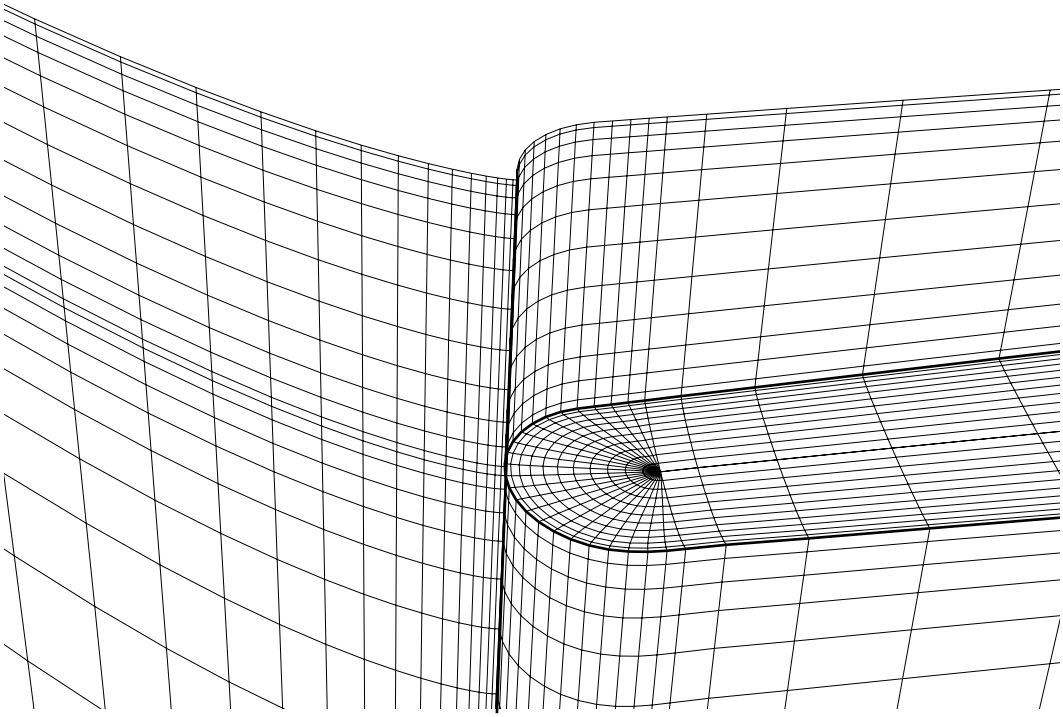


Figure 3 — Top: Meridional H–C grid for a transonic compressor rotor.
Bottom: TCGRID nomenclature and input variables for meridional grid.



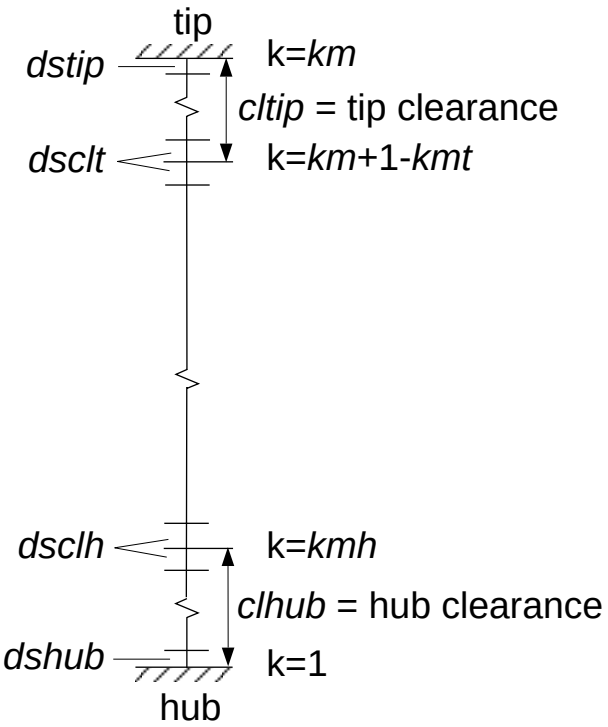


Figure 4 — Top: Tip clearance O–grid for a transonic compressor rotor.
Bottom: TCGRID nomenclature and input variables for spanwise grid.

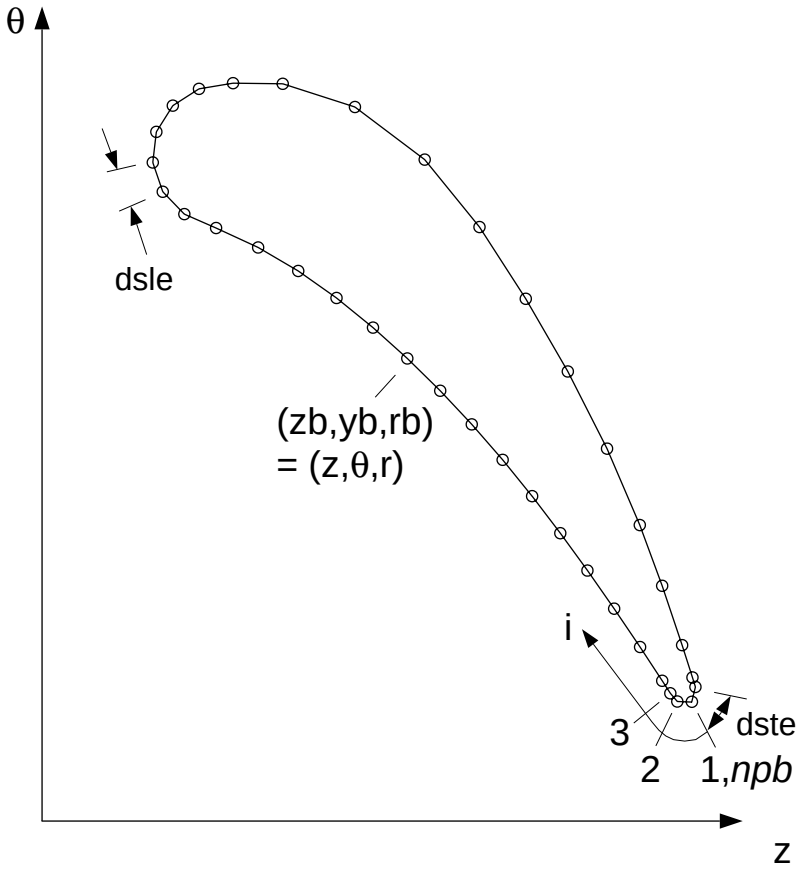


Figure 5 — Blade coordinate input variables for *merid*=0 or 1
Leading and trailing edge spacing parameters.

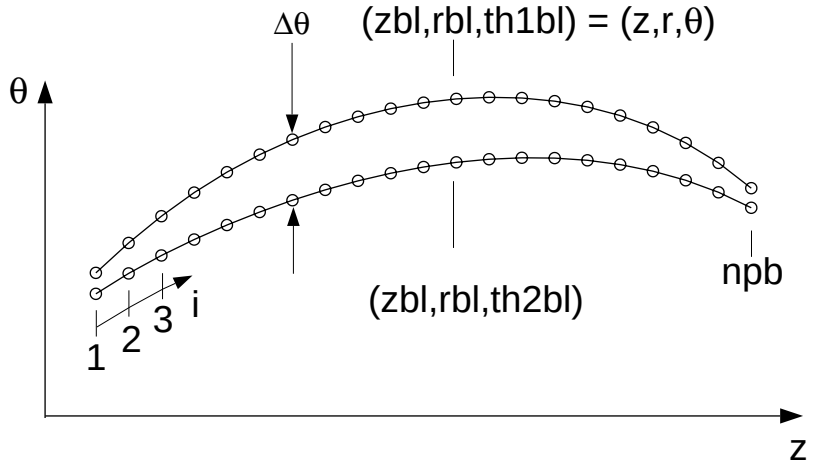
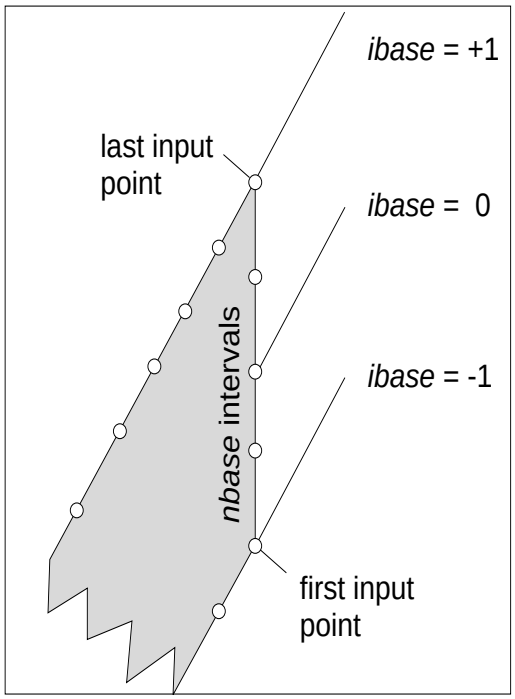
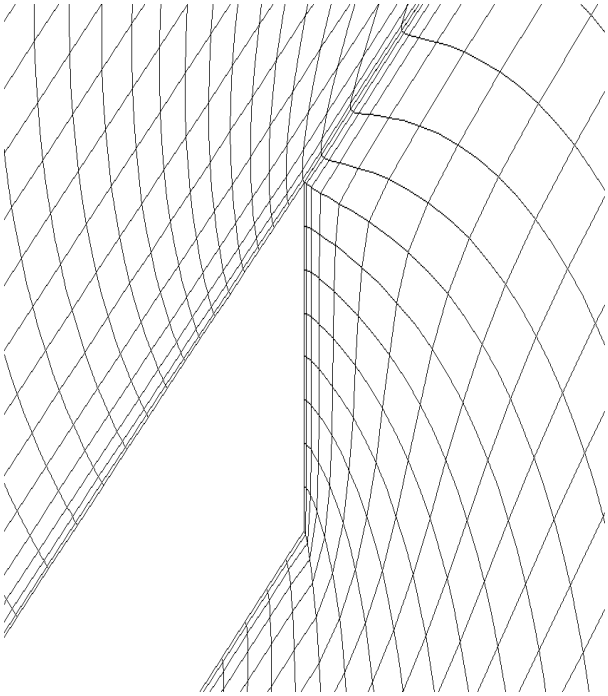


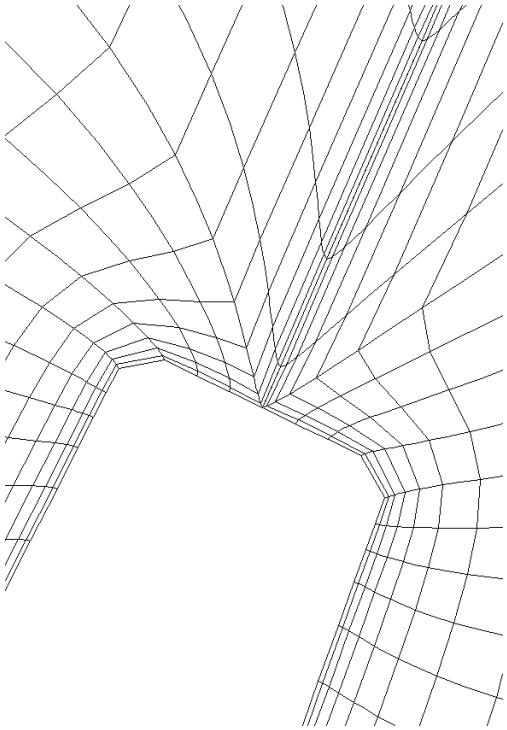
Figure 6 — Blade coordinate input variables for *merid*=2, 3.



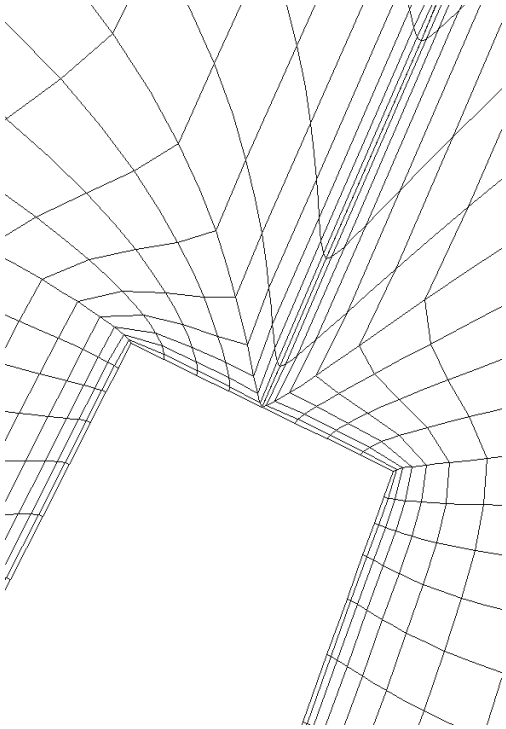
a. Effect of *ibase* on location of wake cut.



b. Centrifugal impeller trailing edge. *ibase* = 1, *nbase* = 8, *ibevel* = 0.



c. Inlet guide vane trailing edge. *ibase* = 0, *nbase* = 8, *ibevel* = 1.



d. Inlet guide vane trailing edge. *ibase* = 0, *nbase* = 8, *ibevel* = 0.

Figure 7 — Options for blunt trailing edges.