UNS3D User's Manual
Release 6.2.2

Aerospace Numerical Simulation Laboratory
Department of Aerospace Engineering
Texas A&M University

February 17, 2024

Contents

List of Tables

List of Figures

Nomenclature

ρ – density

u – velocity in x-direction

v – velocity in y-direction

w – velocity in z-direction

p – pressure (absolute)

T – temperature

e – thermodynamic energy (equal to c_vT)

h – thermodynamic enthalpy (equal to c_pT)

γ – ratio of specific heats (c_p/c_v), taken as 1.4 unless otherwise specified

c_p – specific heat under constant pressure conditions

c_v – specific heat under constant volume conditions

$V_m(t)$ – material volume, defined as the volume which encompasses a set portion of material (moves with velocity $\vec{u}$)

$V_g(t)$ – predefined “grid” volume (moves with velocity $\vec{u}_g$)

V_c – volume which coincides with both V_m and V_g at time t

$\vec{u}_r$ – relative velocity between the two volumes ($\vec{u} - \vec{u}_g$) item[q] – state vector, function of time and space: $q = \{\rho, \rho u, \rho v, \rho w, \rho e\}^T$

q_i – average value of state vector over Ω_i, function of time

Ω_i – refers to a single cell, possibly function of time

R – flux function

$\vec{S}_{ij}$ – directional area of face ij

$\dot{V}_{ij}$ – volume swept by a face per unit time

Chapter 1

Introduction

This manual describes how to install and run the `uns3d` software package. This software package consists of the following codes: (i) `uns3d`, the main code that generates both the full-order and reduced-order model results, (ii) `bfg`, a code that generates the basis functions, (iii) `grassman`, a code that generates the basis functions for off-reference conditions, (iv) `prep`, a code that is used in grid generation, (v) `splitmesh`, a code that splits the mesh for parallel runs, and (vi) a graphic user interface for `uns3d` . A prelude of the `uns3d` and `bfg` codes is included herein, while the other software is described later in the manual.

1.1 The `uns3d` code

The `uns3d` name stands for **Uns**teady-**Uns**tructured Three-Dimensional **3D** **P**roper **O**rthogonal **D**ecomposition.

The `uns3d` code models steady and unsteady flows using the Reynolds-averaged Navier-Stokes model. The `uns3d` software generates both full-order and reduced-order model results. The reduced-order model results are generated using the proper orthogonal decomposition (POD) method with static and dynamic basis functions.

The POD dynamic basis functions approach was developed during the GUIde4 project. The POD dynamic basis functions allow the POD method to be applied to flows with moving boundaries, such as the moving blades in a cascade. Prior to this work, the POD method could not be applied to moving surfaces, and therefore, could not be applied to aeroelastic applications where displacements were significant.

The `uns3d` code is written in Fortran95 and uses the MPI library for parallelization. The code was compiled and tested with several Fortran compilers: Absoft, Intel and Gfortran. The code was tested on Unix and Linux operating systems.

1.2 `bfg`

The `bfg` name stands for **B**asis **F**unctions **G**enerator.

The `bfg` software generates the POD static and dynamic basis functions.

The `bfg` software is written in Fortran95. The code was compiled and tested with several Fortran compilers: Absoft, Intel and Gfortran. The code was tested on Unix and Linux operating systems.

1.3 What is new in version 6.0

Version 6.0 added the Spalart-Allmaras turbulence model [1, 12]. This turbulence model is now the default turbulence model. Card4, on page 29, includes two new variable: `neqt` and `nustart`.

Version 6.0 provides an estimate of the run time. This information is saved in the file `estimated_run_time.dat`, described on page 52.

Chapter 2

Flow Model

In this chapter, the physics and discretization of the flow model are discussed, and the reduced-order model is derived.

2.1 Physical Model

The fluid flow is governed by the conservation of mass, momentum, and energy. For three-dimensional viscous flow, in the absence of source terms, these axioms can be expressed in integral form as

$$\frac{\partial}{\partial t}\int_{\Omega(t)}\mathbf{U}d\Omega + \oint_{\partial\Omega(t)}\left(\mathbf{F}_c - \mathbf{F}_v\right)\mathbf{n}\,dS = \mathbf{0}, \tag{2.1}$$

where

$$\mathbf{U} \equiv \begin{Bmatrix} \rho \\ \rho\mathbf{v} \\ \rho E \end{Bmatrix}, \qquad \mathbf{F}_c \equiv \begin{bmatrix} \rho\left(\mathbf{v}-\mathbf{v}_g\right)^T \\ \rho\mathbf{v}\left(\mathbf{v}-\mathbf{v}_g\right)^T + p\mathbf{I} \\ \rho E\left(\mathbf{v}-\mathbf{v}_g\right)^T + p\mathbf{v}^T \end{bmatrix}, \qquad \mathbf{F}_v \equiv \begin{bmatrix} \mathbf{0}^T \\ \mathbf{T} \\ \left(\mathbf{T}\mathbf{v} + k\nabla\Theta\right)^T \end{bmatrix},$$

$$\mathbf{v} \equiv u\mathbf{i} + v\mathbf{j} + w\mathbf{k}, \qquad \mathbf{T} \equiv 2\mu\mathbf{D} - \frac{2}{3}\mu\left(\nabla\cdot\mathbf{v}\right)\mathbf{I},$$

and $\mathbf{v}_g$ is the velocity of the boundary of Ω, which satisfies the geometric conservation law [2, 13, 14]:

$$\mathbf{v}_g^T\mathbf{n} = \frac{\Delta\Omega}{S\Delta t}.$$

The turbulence is modeled using either a one-equation model, the Spalart-Allmaras model [12], or the two-equation eddy viscosity Shear Stress Transport model proposed by Menter [8]. These models can be cast in a form similar to (2.1).

2.1.1 Dimensionless Variables

To reduce the numerical errors, the flow variables are used in dimensionless form, as shown in Table 2.1

Table 2.1: Dimensionless variables denoted with ˜ and derived reference values based on T_{ref}. Note that p_{ref} is also a reference value but it is not used as such, see subroutine `nondimension`. Furthermore, for dimensionless viscosity, the code uses the array `fmu` which contains $\tilde{\mu}/\mathrm{Re}$ as opposed to just $\tilde{\mu}$. Note that Re is the Reynolds number per unit length.

$\tilde{x} = x/L$	$\tilde{u} = u/c_\infty$	$\tilde{t} = tc_\infty/L$	$\tilde{\rho} = \rho/\rho_\infty$	$\tilde{p} = p/(\rho_\infty c_\infty^2)$
$\tilde{E} = E/c_\infty^2$	$\tilde{H} = H/c_\infty^2$	$\tilde{\mu} = \mu/\mu_\infty$	$\tilde{\mu}_t = \mu_t/\mu_\infty$	$\tilde{e} = e/c_\infty^2$
$\tilde{s} = s/s_\infty$	$s_\infty = p_\infty/\rho_\infty^\gamma$	$c_\infty = \sqrt{\gamma R T_{ref}}$	$\tilde{R} = RT_{ref}/V_{ref}^2$	$\tilde{T} = T/T_{ref}$
$V_{ref} = \sqrt{\gamma R T_{ref}}$	$c_\infty = V_{ref}$			

Note that the primitive state variables are stored in the array `q`. This array has dimensions `nnode` and 6, that is, `q(nnode, 6)`. The sixth component of the array is not a component of the state variable, but stores the dimensionless speed of sound squared

$$\tilde{c}^2 = \frac{c^2}{V_{ref}} = \frac{\gamma R T}{\gamma R T_{ref}} = \tilde{T}.$$

Therefore, the sixth component of the array `q` is also equal to the dimensionless temperature, $\tilde{T}$.

2.2 Full-Order Model

Letting $\mathbf{F} \equiv (\mathbf{F}_c - \mathbf{F}_v)\mathbf{n}$, (2.1) was discretized using finite volumes [3]:

$$\frac{\Delta\left(\Omega_k \mathbf{U}_k\right)}{\Delta t} = -\sum_{\ell=1}^{\mathrm{faces}(k)} \mathbf{F}_{k\ell} S_\ell \equiv -\mathbf{R}_k. \tag{2.2}$$

Equation (2.2) was solved using a Runge–Kutta method with a Roe–Riemann flux-difference splitting scheme.

Mesh deformation was achieved through radial basis function interpolation within the updated boundaries [4].

Chapter 3

Numerical Method

The numerical method used in the `uns3d` code to solve the Reynolds-averaged Navier-Stokes equations is second-order accurate in time and space. A first-order spatial accurate method is also available for starting up the simulation.

3.1 Spatial Discretization

3.1.1 Integral Formulation

The Navier-Stokes equations are discretized using the finite volume method. Consequently, the governing equations are written in integral form. A generic scalar conservation equation written in the differential form as

$$\frac{Du}{Dt} + F(u)_{i,i} = 0 \tag{3.1}$$

where $(\cdot), i \equiv \partial(\cdot)/\partial x_i$, becomes in integral form

$$\frac{D}{Dt}\int_{\Omega} u d\Omega + \int_{\partial\Omega} F(u)_i n_i d\Omega = 0. \tag{3.2}$$

The weak form of the generic scalar conservation equation is

$$\frac{D}{Dt}\int_{\Omega} \phi u d\Omega - \int_{\Omega} \phi_{,i} F(u)_i d\Omega = \int_{\Omega} \phi F(u)_i n_i d\Omega. \tag{3.3}$$

Eq. (3.3) reduces to Eq. (3.2) if the test function is constant, *i.e.*, $\phi_{,i} = 0$. The integral form can be viewed as a weak formulation of the Finite Element Method with a constant test function.

3.1.2 Fluxes Computation

Several options are available in `uns3d` for the fluxes computation. The Godunov method, which is an upwind method for hyperbolic equations, is employed herein to resolve the inviscid flux. One options is to solve the Riemann problem using the Godunov method with the Roe's approximate Riemann solver [10, 11].

The inviscid flux from Roe's approximate Riemann solver [10, 11] is defined in terms of the two states across a median-dual cell face, $\mathbf{u}_L$ and $\mathbf{u}_R$,

$$\mathbf{f}_c = \frac{1}{2}\left[\mathbf{F}_c(\mathbf{u}_L) + \mathbf{F}_c(\mathbf{u}_R) - |\tilde{\mathbf{A}}|\Delta\mathbf{u}_{R,L}\right] \tag{3.4}$$

where $|\tilde{\mathbf{A}}|$ is the flux Jacobian with respect to the conservative variables and $\Delta(\cdot) = (\cdot)_R - (\cdot)_L$ is the difference between the right and left states. The tilde symbol in $|\tilde{\mathbf{A}}|$ indicates that the Jacobian is evaluated using averaged state variables that are constant and therefore the solution is based on a linearized equation.

Roe's density weighted averages are defined as [10]

$$\begin{array}{rcl} \tilde{\rho} &=& \sqrt{\rho_R \rho_L} \\ \tilde{u} &=& (u_L\sqrt{\rho_L} + u_R\sqrt{\rho_R})/(\sqrt{\rho_L} + \sqrt{\rho_R}) \\ \tilde{v} &=& (v_L\sqrt{\rho_L} + v_R\sqrt{\rho_R})/(\sqrt{\rho_L} + \sqrt{\rho_R}) \\ \tilde{w} &=& (w_L\sqrt{\rho_L} + w_R\sqrt{\rho_R})/(\sqrt{\rho_L} + \sqrt{\rho_R}) \\ \tilde{H} &=& (H_L\sqrt{\rho_L} + H_R\sqrt{\rho_R})/(\sqrt{\rho_L} + \sqrt{\rho_R}) \\ \tilde{V} &=& \tilde{u}n_x + \tilde{u}n_y + \tilde{u}n_z - V_g. \end{array} \tag{3.5}$$

Roe's approximated Riemann solver admits an expansion shock because the vanishing viscosity does not provide enough diffusion. The instability of the Roe Riemann solver in particular cases is well documented [5, 7, 16]. Herein we used Harten's entropy fix [6]. This fix modifies the eigenvalue such that it never reaches 0. Whereas the numerical dissipation provided by Roe's method is proportional to the magnitude of eigenvalue, the Harten's fix adds non-vanishing dissipation to remove the expansion shock and the instability problem at the expense of being more dissipative.

In addition to the Roe with Harten entropy fix, `uns3d` offers the following alternative options for calculating the fluxes: RoeM, AUSM+, AUSMPW+, HLLC, and Modified Steger-Warming.

3.1.3 Gradient Computation

The accuracy of the gradient computation is important for correctly estimating the viscous fluxes, which are defined in terms of the gradients of the velocity components and the temperature. The diffusion terms in the $k - \omega$ turbulence model also require the gradient of the k and ω field. The `uns3d` offers four options for calculating the gradients: (1) the Green-Gauss method, (2) the least squares method, (3) the least squares with QR, and (4) the WENO method.

3.2 Temporal Discretization

There are many possible implementations of integration in time, such as explicit or implicit and single time step or dual time step. `uns3d` can use one of four options: (1) explicit single time stepping, (2) implicit single time stepping, (3) explicit dual time stepping, and (4) implicit dual time stepping.

Explicit and implicit time stepping can be run using either a time-accurate approach or a steady-state approximation. Dual time stepping by nature is time-accurate, though it uses some of the acceleration techniques available to steady-state flows.

Chapter 4

Software Installation

This chapter explains how to install the `uns3d` software on your Linux or Unix computer. The first section presents how to define some useful environment variables and how to create folders for source and data files. The second section presents the installation of the `uns3d` code, the flow solver.

4.1 Preliminary Setup

It is convenient to define first the folder where software is being installed. For the C-shell add the following line to the `.cshrc` file:

```
setenv UNS3DHOME /wherever_you_want_it_to_be
```

For the Bourne, Bash or Korn shell, add the following line to the `.bashrc` file:

```
export UNS3DHOME=/wherever_you_want_it_to_be
```

Create two folders in `$UNS3DHOME`, one for source files and one for the data.

```
cd $UNS3DHOME

mkdir src

mkdir dat
```

In the `src` folder, create a folder for the flow solver :

```
cd $UNS3DHOME/src
```

`mkdir 5.3.1,` this folder is for the version 5.3.1 of the flow solver

4.2 `uns3d` Installation

To install the flow solver, follow these steps:

1. Copy the distribution file `uns3d .tgz` from where you downloaded to `$UNS3DHOME/5.3.1` and untar it:

   ```
   cp uns3d .tgz $UNS3DHOME/5.3.1

   cd $UNS3DHOME/5.3.1

   tar xvfz  uns3d .tgz
   ```

2. Edit the makefile to match the FORTRAN compiler that you have available and then make the makefile to generate the executable `uns3dpod`:

   ```
   make
   ```

3. For convenience, you might want to add the executable `uns3d` to a folder that is in the `$PATH`, for example `/usr/local/bin`. To do this, copy the executable `uns3d` :
   ```
   sudo cp  uns3d /usr/local/bin
   ```
   The `sudo` command requires the superuser password for your computer.

Chapter 5

Starting, Executing, and Stopping `uns3d`

This chapter describes how to run `uns3d` using terminal line commands.

5.1 Running `uns3d`

To generate flow snapshots using `uns3d` , switch to the directory from which you want to run. For example, this can be accomplished with the Unix command:

```
cd $UNS3DHOME/dat/example/FOM.
```

In this example it is assumed the following folder structure:

```
example/basis_da_db_6
example/FOM/out
example/FOM/plt
example/FOM/txt
example/mesh
```

Once in the appropriate directory, the user has two choices for running `uns3d` : (1) interactively or (2) in batch. Before running the `uns3d` code, especially for large cases, it is prudent to check your stack size by typing:

```
ulimit -s
```

If the stack size is too small, you might get the dreaded `Segmentation fault` error message. Therefore, you might want to increase it before running `uns3d` . This can be done using:

```
ulimit -s 65000
```

which sets the stack size to 65,000 kbytes.

To run `uns3d` interactively use the following command statement:

```
uns3d input
```

The above command statement assumes that the `uns3d` executable file is located in the $PATH, as described in Section 4.2. The name of the input file is given by the character argument that follows the name of the executable. In the case of the example, the input file is called `input`. The input file is discussed in Section 6.1.

To run the `uns3d` code in parallel, use the following command:

```
mpirun -np 240 uns3d input
```

when the code was run on 240 processors.

A run file can be used when running `uns3d` in batch. The run file should contain the same commands used to run interactively. An example of this could be

```
uns3d input0
uns3d input1
uns3d input2
```

In this case `uns3d` will be run three consecutive times to produce two intermediate solution files (see next section for details). The run file is then submitted to the queue for execution using the appropriate system commands.[1] The mesh files must be either physically located in or linked to the execution directory for both interactive and batch running options.

`uns3d` writes five main output files during the course of the simulation. The first three files are used to backup the flow fields, turbulence fields, and unsteady flow fields, respectively. These three files can be used to restart the flow solver. A fourth file is used to output the convergence history and the integrated parameters. This file has the same format as the information that is dumped to the screen while `uns3d` is running. The fifth file is used for solution visualization, and is written in either Visual3 or Tecplot format. The names of these five output files can be specified by the user in the input file.

In addition to the five main output files, `uns3d` writes three groups of five files that contain different forms of the variable histories for density, x-component of velocity, y-component of velocity, z-component of velocity, and pressure, respectively. The three different types of variable histories are the average residual history, the maximum residual history, and the variable error. Also written are the iteration (time) histories for the integrated forces in the x-, y-, and z-directions, maximum Mach number, and the ratio of input/output mass flow rates.

Typical Simulation

Rather than specifying too many iterations in one run, it is recommended to use multiple runs with a “reasonable” number of iterations. The value of a “reasonable” number of iterations depends on how many iterations your computer can simulate per hour, and how many hours you would like to wait until an intermediate result is generated. These intermediate results will be used as start-up results for subsequent runs.

To use intermediate results for an inviscid case, the file names `fileinq`, `fileoutq`, and `ireadq` in the input file must be modified. Detailed information about the input and output files is given in Chapter 6. The variable `ireadq` should be set to 0 for the very first run. Afterwards, `ireadq` must be set to 1. When `ireadq` = 1, the variable `fileinq` should be specified. When running a turbulent flow case, the variables `fileinqt`, `fileoutqt`, and `ireadqt` must also be modified in a similar fashion to `fileinq`, `fileoutq`, and `ireadq`. By chaining input and output files between the consecutive runs, an uninterrupted single run can be obtained.

[1]Other commands may be needed in the run file for batch submission. Read your local batch submission instructions for details

5.2 Stopping `uns3d`

`uns3d` can be stopped by adding a file with the name ’stop.dat’ in the folder where the code is run from. Alternative names for this file are ’Stop.dat’ and ’STOP.dat’.

The only information in the ’stop.dat’ file is an integer. If this integer is 0, the code writes the appropriate restart files and Tecplot files, and terminates. If the integer is 1, the code writes the same information before terminating, however, the filenames have the ’.STOP’ appended. If the integer of the ’stop.dat’ file is different from 0 and 1, the code terminates without writing any restart information.

Make sure the ’stop.dat’ file is removed before attempting a new run.

Chapter 6

Input and Output Files for `uns3d`

6.1 `uns3d` Input File

6.1.1 Main input file

The main input file for `uns3d` is written using the Fortran namelist format. Multiple input blocks are used to specify the input parameters. The input file specifies the names of the restart and output files, code control parameters, and the boundary conditions. The input parameters are presented using the same groupings as the actual input file. The first column gives the name of the variable. The second gives the variable type: integer (I), real (R), logical (L), or character string (C). The final column gives the variable description, and in some cases a default value.

&cardf

&cardf – *Output File Names*

title	C	Case description
case_name	C	Specifies the base filename for the y^+ output files
rsdfile	C	Name of the file that stores the convergence history and the integrated parameters, such as mass flow rate, force components, efficiency, and the ratio of outlet static (or total) pressure to a reference pressure
relative_v	L	Flag that specifies whether the output velocity is in a relative or absolute frame of reference
dump_v3	L	Flag that outputs a V3 data file – NOT USED
v3data	C	Name of the output file that stores the flow field and turbulence variables for the V3 visualization package
dump_tecplot	L	Flag that specifies if a TECPLOT format output file is written
tecplot_name	C	Name of the TECPLOT format output file [DEFAULT=tecplot.dat]
dump_yplus	L	Flag that controls the output of y^+ values on walls [DEFAULT=.FALSE.]
bcplot	L	Flag that controls the output of the boundary condition values [DEFAULT=.FALSE.]. If .TRUE., the file is saved as plt/boundary_face_values.plt.
dump_grads	L	Flag that controls the output of gradient values [DEFAULT=.FALSE.]
grad_name	C	Name of the output gradient file [DEFAULT=plt/gradients.plt]
dump_resid	L	Flag that controls output of flow and turbulence equation residual plot files. Files are written to<case_name>**_q_residuals.plt** and <case_name>**_qt_residuals.plt** [DEFAULT=.FALSE.]
mapbc	L	Flag that indicates whether or not a .mapbc boundary index file is present and should be read. This flag dictates how the boundary conditions are specified – see &card4 and &card5 for more details [DEFAULT = .FALSE.]. Note: the grid meshes are different between the cases that run with mapbc .TRUE. and .FALSE. For the .FALSE. option, the mesh file includes the idbcs, that is, the negative -1 to -100+ integers. For the .TRUE. option, the mesh file includes the number of the boundary type, integers from 1 to the max number of boundaries.
mapbcfile	C	Name of the .mapbc file (described in Sec. 6.1.4). File is only read if mapbc = .TRUE. [DEFAULT = vol.mapbc]

ref_io	L	Flag that if set .TRUE. then ref_inlet and ref_outlet are specified in the input file. If .FALSE. then the first inlet boundary is the inlet reference boundary and the first outlet boundary is the outlet reference boundary [DEFAULT=.FALSE.]
ref_inlet	I	Index specifying which boundary condition of the ⋆.mapbc file is the main inlet boundary of the computational domain. The flow conditions of the main inlet boundary are then used when computing reference values for the simulation. Note that if ref_inlet=0 then the first inlet boundary listed in ⋆.mapbc becomes the main inlet boundary, so the order of inlet boundaries in ⋆.mapbc is important. [DEFAULT=0]
ref_outlet	I	Index specifying which boundary condition of the ⋆.mapbc file is the main outlet boundary of the computational domain. The flow conditions of the main outlet boundary are then used when computing reference values for the simulation. Note that if ref_outlet=0 then the first outlet boundary listed in ⋆.mapbc becomes the main outlet boundary, so the order of outlet boundaries in ⋆.mapbc is important. [DEFAULT=0]
bingrid	L	flag for mesh and c2n files; if .TRUE., both files are binary; if .FALSE., both files are ascii [DEFAULT=.FALSE.]
ioflag	I	flag for state variable files [DEFAULT=0] = 0 ascii in / ascii out = 1 ascii in / binary out = 2 binary in / binary out = 3 binary in / ascii out
init_shocktube	L	Flag to generate an initial flow field for a shock tube [DEFAULT=.FALSE.]
multilim	L	Flag to use multiple slope limiters [DEFAULT=.FALSE.]

&cardg

&cardg – *Grid File Name*

`gridfile`	C	Name of the pre-processed grid data. This file is generated by the grid pre-processing code (PREP)
`c2nfile`	C	Name of the cell-to-node grid data. This file is generated by the grid pre-processing code (PREP)
`l2gfile`	C	Name of file that gives the relationship between the number of a node on its local processor and the number of the same node on entire (global) grid. [DEFAULT = loc2glob.dat]
`XleXteRte`	C	Name of the file that is used to define the initial flow field. For an annular cascade of an axial compressor (igeom=0) the file defines the velocities at every node. For an annular cascade of a radial compressor (igeom=-1) the file specifies the impeller face location (xle), the impeller backplate (xte), and the outer radius of the impeller (rte).
`ijkfile`	C	
`surfint_file`	C	
`multilimfile`	C	Name of `.dat` file (described in Sec. 6.1.5). File is only read if multilim = `.TRUE.` [DEFAULT = typlim_ids.dat]

&cardh

&cardh – *Flow Field Restart Files*

`fileinq`	C	Name of the input file that contains the state variables $\{\rho, u, v, w, p\}$ at time t_{last} of the previous run; this file should have been written in the previous run; this file is read at the beginning of the current run unless `ireadq` = 0.
`fileoutq`	C	Name of the output file that contains the state variables $\{\rho, u, v, w, p\}$ at time t_{last}, the last time of current run; this file is written at the end of the current run.
`fileinq12`	C	Name of input file that stores the state variable $\{\rho, u, v, w, p\}$ at times t_{last-1} and t_{last-2} of the previous run; this file should have been written in the previous run; this file is read at the beginning of the current run unless `readq12` = F.
`fileoutq12`	C	Name of output file that is written at the end of the current run, file that stores the state variable $\{\rho, u, v, w, p\}$ at times t_{last-1} and t_{last-2} of the current run; written at the end of the run for unsteady problem.

&cardi

&cardi – *Turbulent Field Restart Files*

`fileinqt`	C	Name of input file that contains the turbulence variables at time t_{last} of the previous run; this file should have been written in the previous run; this file is read at the beginning of the current run unless `ireadqt` = 0
`fileoutqt`	C	Name of the output file that contains the turbulence variables at time t_{last}, the last time of current run; this file is written at the end of the current run.
`fileinqt12`	C	Name of the input file that contains turbulent variables at times t_{last-1} and t_{last-2} of the previous run; this file should have been written in the previous run; this file is read at the beginning of the current run unless `readqt12` = F.
`fileoutqt12`	C	Name of the output file that contains turbulent variables at times t_{last-1} and t_{last-2} of the current run; written at the end of the run for unsteady problem.

&cardk

&cardk – *Multi-Block Mesh Restart Files*

`filembin`	C	Name of the multi-block mesh position input restart file
`filembout`	C	Name of the multi-block mesh position output restart file

&card0

&card0 – *Boundary Condition Type Controls*

`noblade`	I	number of compressor/turbine blades, used to define periodicity; For `noblade` >0, `igeom` must be 0 or -1. If `noblade` = 0, `igeom` must be 1. For annular cascades, `noblade` = 0 is used to initialize the solution and then we switch to the correct `noblade`. = `1` 360 deg = `2` 180 deg … … = `n` 360/n deg = `0` linear cascade; `igeom` must be 1
`igeom`	I	Flag indicating the geometry type: = `-1` Annular cascade row: **centrifugal** = `0` Annular cascade row: **axial** = `1` Linear cascade
`inbc(i)`	I	Inlet boundary condition flag ***array*** whose array index corresponds to the boundary face index found in `*.mapbc` file. $1 \leq i \leq$ itmax. Omit the array index if mapbc = `.FALSE.` When multiple inlet boundary conditions are specified, the first inlet boundary condition is the main one. The main inlet boundary condition is used for nondimesionalization. = `-2` Subsonic inlet boundary conditions that prevent reversed flow at inlet; uses stagnation pressure, two velocity angles, and Riemann 1 from upstream, and Riemann 2 from the interior = `-1` = `0` Uniform inlet boundary conditions [DEFAULT] = `1` = `2` = `3` = `4` Entropy, stagnation enthalpy = `5` Supersonic flow; uniform inlet velocity, pressure and temperature = `6` See Blazek Section 8.4 = `7` Supersonic flow; uniform inlet velocity, pressure and temperature = `8` Test case for supersonic free vortex flow
`ioutbc(i)`	I	Outlet boundary condition flag ***array*** whose array index corresponds to the boundary face index found in `*.mapbc` file. Omit the array index if mapbc = `.FALSE.` When multiple outlet boundary conditions are specified, the first outlet boundary condition is the main one.

		= `0` “leak outlet” condition (secondary back pressure outlet)
		= `1` Checks if there is back-flow, and if there is, switches boundary condition pressure from outside the domain to inside the domain
		= `2` Impose static back pressure unless supersonic [DEFAULT]
		= `6` Extrapolation
`ipex(i)`	I	Outlet static pressure radial variation flag ***array*** whose array index corresponds to the boundary face index found in `*.mapbc` file. Omit the array index if mapbc = `.FALSE.`
		= `0` Specify uniform pressure [DEFAULT]
		= `1` Enforce radial equilibrium with pressure defined at “hub”
		= `2` Enforce radial equilibrium with pressure defined at “tip”
`isymbc`	I	Symmetry boundary condition flag. Refer to Sec. A.1 for detailed option descriptions of each option
		= `0` “Pseudo-ghost cell”
		= `1` “Inviscid wall” [DEFAULT]
`ifarbc`	I	Far-field boundary condition flag. Refer to Sec. A.2 for detailed descriptions of each option.
		= `0` Slip-wall [DEFAULT]
		= `1` Extrapolate
		= `2` Riemann Invariant
`ispet`	I	Thermal boundary condition type
		= `0` Adiabatic wall [DEFAULT]
		= `1` Isothermal wall. Wall temperature given in `&card5`
`leak_outlet`	L	Leak outlet flag
		=`.TRUE.` Leak outlet exists
		=`.FALSE.` No leak outlet [DEFAULT]
`iwall`	I	Type of wall boundary condition for inviscid flow
		= `1` Ghost cell and interior cell impose no-penetration
		= `2` No longer used
		= `3` Ghost cell imposes no-penetration [DEFAULT]
		= `4` No longer used
`deltafhat`	R	Entropy correction [DEFAULT=0.001] (see AERO615 class notes, p.20:7)

&card1

&card1 – *Code Controls*

ireadq	I	Flag that tells whether the state variables $\{\rho, u, v, w, p\}$ are either read from the file fileinq (specified in &cardh) or initiated using a uniform field = 0 Start from uniform field [DEFAULT] = 1 Read flow field from fileinq
readq12	L	Flag that tells whether to read in and use the flow fields q1 and q2 at the previous two time steps, which are stored in file fileinq12; [DEFAULT=.FALSE.]
ireadqt	I	Flag that tells whether the turbulence variables are either read from the file fileinqt (specified in &cardi) or initiated using a uniform turbulence field. = 0 Start from uniform field [DEFAULT] = 1 Read field from fileinqt
readqt12	L	Flag that tells whether to read in and use the turbulence variables qt1 and qt2 at the previous two time steps, which are stored in file fileinqt12; [DEFAULT=.FALSE.]
npseudotimesteps	I	Number of pseudo-time marching steps. For steady flows this is the number of iterations. [DEFAULT=20]
mtime	I	Number of real time marching steps, [DEFAULT=1]. = 1 Steady flow or unsteady single time-stepping $>$ 1 Dual time-stepping
iramp	I	Number of pseudo-time steps over which the boundary conditions are ramped up. Use 1 to apply boundary conditions suddenly, if there is no convergence problem; [DEFAULT=1]
iramp0	I	Number of pseudo-time steps after which the ramping up of the boundary conditions begins (i.e., in the first iramp0 pseudo-time steps the boundary conditions are not updated). Use 0 if there is no convergence problem; [DEFAULT=0] Example: iramp = 10, iramp0 = 5 $\rightarrow$ for the first 5 pseudo-time steps, the boundary conditions are not changed (there are exceptions, such as wall boundary conditions); then, for the next 10 pseudo-time steps, the boundary conditions are ramped up.
iramp_leak	I	Iterations to ramp boundary conditions for leak outlet used in the compressor with leakage version; currently not used in versions 4.4 and later; similar to iramp except that applied to compressor leakage; [DEFAULT=1].

iramp0_leak	I	Used if leak_outlet=.true.; for stability purposes, leak outlet can be treated initially as a solid wall. The boundary opens after iramp0_leak iterations; [DEFAULT=0]
timestep	I	Flag specifying time step computation method
		= 0 the most accurate version; takes into account both convective and viscous spectral radii (the viscous spectral radius does not include turbulence effects) - currently produces smallest time step because convective spectral radii are larger than viscous (laminar) spectral radii [DEFAULT=0]
		= 1 convective spectral radii are ignored - currently produces largest time step
		= 2 uses a variant definition of convective spectral radii
		= 3 identical to 1
ttime	R	End run time for unsteady simulations, in seconds [DEFAULT=1000.0]

&card2

&card2 – *Code Controls*

`iorder`	I	Spatial accuracy order = `1` First-order accurate [DEFAULT] = `2` Second-order accurate
`use_limiter`	L	Flag to indicate whether to use the limiter, [DEFAULT= `.TRUE.`]
`invis`	L	Flow model flag = `.TRUE.` Inviscid flow (Euler equations) = `.FALSE.` Viscous flow (Reynolds-averaged Navier-Stokes equations) [DEFAULT]
`lamin`	L	Flag that specifies whether the flow is laminar or turbulent = `.TRUE.` Laminar flow [DEFAULT] = `.FALSE.` Turbulent flow modeled with either the Spalart-Allmaras or the $\kappa - \omega$ Shear Stress Transport model
`itrans`	I	*No longer used*
`mstg`	I	Number of stages in Runge-Kutta = from 1 to 5 for first order [DEFAULT=4] = from 1 to 5 for second order [DEFAULT=3]
`irhsm`	I	Number of Jacobi iterations for implicit residual smoothing, [DEFAULT=0]
`lsgg`	I	Gradient computation method = `0` Green-Gauss [DEFAULT] = `1` Inverse distance weighted least squares = `2` Weighted least squares with QR (WLSQR). Weight specified by `wlsqr_pwr` = `3` WENO = `4` Hybrid Green-Gauss/WLSQR (experimental stage)
`iramp_lim0`	I	The first iramp_lim0 iterations the limiter is set 0; after iteration iramp_lim0 the limiter is gradually increased to its full values (during iramp_lim iterations); [DEFAULT=0]
`iramp_lim`	I	number of iterations after iramp_lim0 iterations during which the limiter is multiplied by a ramp function that varies smoothly between 0 and 1; [DEFAULT=1]
`steady`	L	Flag for steady flow = `.TRUE.` steady flow [DEFAULT] = `.FALSE.` unsteady flow
`typlim`	I	Limiter type for second-order computations

		= `0` No limiter
		= `1` Barth
		= `2` Venkatakrishnan [DEFAULT]
		= `3` van Albada
		= `4` MLP - Venkatakrishnan
		= `5` Flux limiting (Roe - Harten flux only)
`typlim_opt`	I	Additional options for limiters [DEFAULT=0]
if `typlim = 2`		
= `0`		Venka-e1 (original Venkatakrishnan)
= `1`		Venka-Wang
= `2`		Venka-e2 (Forrest's modification)
if `typlim = 4`		
= `0`		MLP-e1
= `1`		MLP-Wang
= `2`		MLP-e2
if `typlim = 5`		
= `0`		Slope limiter fct. van PHI=1.0
= `1`		Slope limiter fct. van Albada
= `2`		Slope limiter fct. van Leer
= `3`		Slope limiter fct. ospre
= `4`		Slope limiter fct. HQUICK
= `5`		Slope limiter fct. superbee
= `6`		Slope limiter fct. minmod
`fluxtype`	I	Flux function selection (highly recommended are options 1 and 3)
		= `1` Roe with Harten entropy fix [DEFAULT]
		= `2` AUSMPW+
		= `3` RoeM (4.9d and later) or Roe-EC(4.9c)
		= `4` AUSM+
		= `5` Steger-Warming (4.9d and later)
		= `6` HLLC (4.9d and later)
`venka_c`	R	Coefficient in Venkatakrisnan's limiter; [DEFAULT=5.0]
`hardwall`	L	Flag for hard wall boundary condition; [DEFAULT=`.TRUE.`]
`preconditioned`	L	Flag for preconditioning; [DEFAULT=`.FALSE.`]
`twod`	L	Flag for 2D or 3D; currently inactive [DEFAULT=`.FALSE.`]
`wlsqr_pwr`	R	Value of the exponent applied to the WLSQR weighting function, $w_i = d_i^{-p}$ where d_i is the distance along edge i
		= `0.0` Un-weighted
		= `1.0` Inverse distance weighted [DEFAULT]
		= `2.0` Inverse distance squared weighted

&card2a

&card2a – *Implicit Solver Controls*

imp	L	Flag that specifies whether to use implicit solver or not = .TRUE. - implicit [DEFAULT] = .FALSE. - explicit
lag	R	Amount the implicit terms are lagged, lag $\in$ (0,1), [DEFAULT=0.5d0]
imp_tol	R	Implicit sub-iteration tolerance, [DEFAULT=5d-6]
max_imp_sub	I	Maximum number of implicit sub-iterations, [DEFAULT=5]
int_cfl	R	Initial CFL number for CFL ramping, [DEFAULT=2.0]
cfl_ramp_n	I	Number of iterations in which the CFL number is ramped from int_cfl to cfl, [DEFAULT=50]
cfl_back	R	Percentage the CFL number is lowered at the start of each real time step in dual time stepping [DEFAULT=0.2]
imp_o_store	L	Flag for computation/storing of imp_o matrix =.TRUE. Compute once & store (faster, more memory) [DEFAULT] =.FALSE. Compute when needed (slower, less memory)
visc_jac	I	Flag used for implicit solver, which sets type of viscous flux Jacobian = 1 numeric viscous flux Jacobian = 2 diagonalized viscous flux Jacobian = 3 direct viscous flux Jacobian [DEFAULT] = 4 no viscous flux Jacobian

&card3

&card3 – *Time Step Definition*

`cfl`	R	Courant-Friedrichs-Lewy (CFL) number. Maximum CFL number depends on the order of the scheme and the number of stages in the Runge-Kutta scheme; [DEFAULT=0.5]
`dtimedim`	R	Size of the real time step for *dual-time* stepping scheme; [DEFAULT=1.0d-4] Note: the size of the real time step for the *single-time* stepping scheme is calculated using the CFL number.
`epss`	R	implicit residual smoothing factor; [DEFAULT=0.5]

&card4

&card4 – *Initial Flow Field Parameters*

Some variables that define the initial flow field parameters depend on the value of mapbc. This approach was taken to ensure backwards compatibility with older input files.

hrough	R	The equivalent sand grain roughness height in turbulent wall units; [DEFAULT=5.0] - NO LONGER USED
vel0_scale	R	Scaling factor that can be used to scale the uniform velocity field during solution initialization; [DEFAULT=1.0d0]
neqt	I	Number of equations for the turbulent model. If neqt = 1, the Spalart-Allmaras turbulence model is used. If neqt = 2, the Shear Stress Transport (SST) turbulence model is used; [DEFAULT=2]
nustart	R	Initial value of eddy viscosity-like variable, $\tilde{\nu}$. The eddy viscosity-like variable is needed for the Spalart-Allmaras turbulence model, that is, when neqt = 1; [DEFAULT=3.0d0]

mapbc = .FALSE. specific variables

u0	R	Freestream Mach number component in x-direction; [DEFAULT=0.0]
v0	R	Freestream Mach number component in y-direction; [DEFAULT=0.0]
w0	R	Freestream Mach number component in z-direction; [DEFAULT=0.0]
alfax	R	Flow angle between x-axis and the projection of the velocity on the $x-z$ plane, in degrees; [DEFAULT=0.0]
alfaz	R	flow angle between velocity vector and its projection on the $x-z$ plane, in degrees; [DEFAULT=0.0]
tintens	R	Turbulent intensity; [DEFAULT=0.01]
tlength	R	Turbulent length scale; typically 5% of the channel height at the inlet in a turbine; could be calculated as $\ell = C_\mu \kappa^{1.5}/\epsilon$, where κ is the kinetic energy (per unit mass) of the turbulent fluctuations, $\kappa = 0.5\overline{u_i' u_i'}$, ϵ is the dissipation per unit mass, $\epsilon = \nu \overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_i'}{\partial x_k}}$ and C_μ=0.09. [DEFAULT=0.01]

mapbc = .TRUE. specific variables

ax(i)	R	Array equivalent of alfax (see above), where i specifies the *.mapbc index of the target inlet boundary face
az(i)	R	Array equivalent of alfaz (see above), where i specifies the *.mapbc index of the target inlet boundary face

`tin(i)`	R	Array equivalent of `tintens` (see above), where `i` specifies the `*.mapbc` index of the target inlet boundary face
`tlen(i)`	R	Array equivalent of `tlength` (see above), where `i` specifies the `*.mapbc` index of the target inlet boundary face

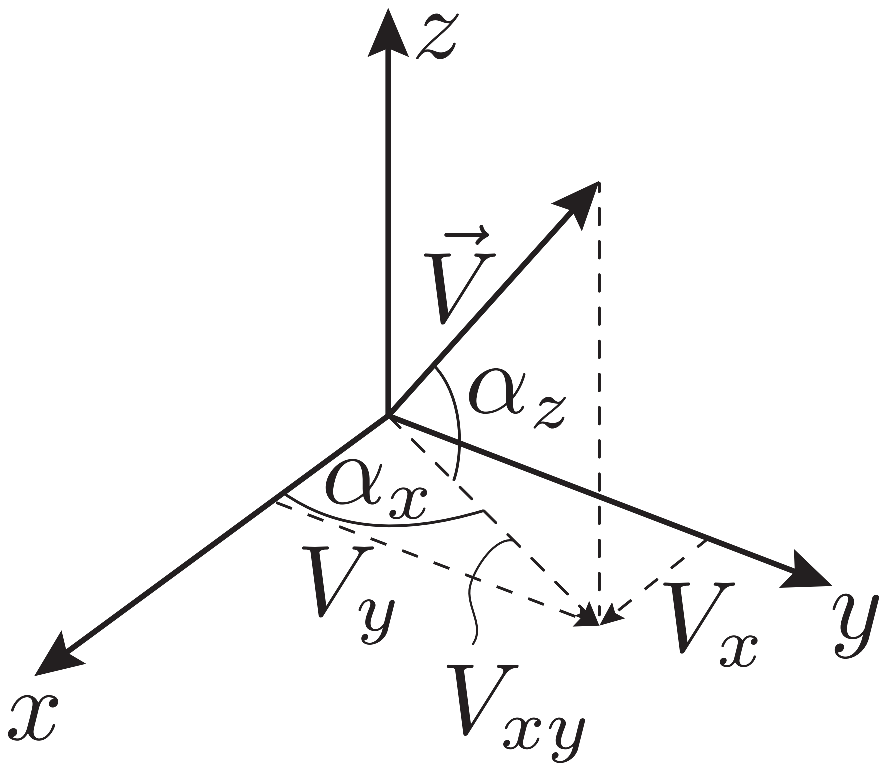


Figure 6.1: Inlet velocity and its angles.

&card5

&card5 – *Flow Conditions*

Some variables used to define the flow conditions depend on the value of mapbc. This approach was taken to ensure backwards compatibility with older input files.

twall	R	Wall temperature, in K; [DEFAULT=288.15]

mapbc = .FALSE. specific variables

ptot	R	Inlet total pressure, in Pa; [DEFAULT=101325.0]
ttot	R	Inlet total temperature, in K; [DEFAULT=288.15]
pback	R	Outlet static pressure, in Pa; [DEFAULT=ptot]
pback_ratio	R	Ratio of outlet static pressure to inlet total pressure. Used in place of pback if pback_ratio $\neq 0$; [DEFAULT=0.0d0]
pback_leak_rat	R	Ratio of leak outlet static pressure to inlet total pressure; [DEFAULT=1.0]

mapbc = .TRUE. specific variables

ubc(i,k)	R	Array equivalent of u0,v0, and w0 where i is the *.mapbc index of the target *inlet* boundary, and k corresponds to either 1 (u0), 2 (v0), or 3 (w0) [DEFAULT=0.]
p0_in(i)	R	Array equivalent of ptot (see above) where i is the *.mapbc index of the target *inlet* boundary [DEFAULT = -1.]
t0_in(i)	R	Array equivalent of ttot (see above) where i is the *.mapbc index of the target *inlet* boundary [DEFAULT = -1.]
ps_ex(i)	R	Array equivalent of pback (see above) where i is the *.mapbc index of the target *outlet* boundary [DEFAULT = -1.]

&card6

&card6 – *Reference Conditions*

`pref`	R	Pressure, in Pa. Can be either $\mathtt{p}_\infty$ or `ptot`; [DEFAULT=96173.42]
`tref`	R	Temperature, in K. Can be either $\mathtt{T}_\infty$ or `ttot`; [DEFAULT=302.77]
`scale`	R	Scale factor for grid units; [DEFAULT=1.0]
`pinf`	R	Static pressure corresponding to `ptot` and inlet Mach number; if not specified or set to -1, the values is calculated by the code [DEFAULT=-1.0]
`tinf`	R	Static temperature corresponding to `ttot` and inlet Mach number; if not specified or set to -1, the values is calculated by the code [DEFAULT=-1.0]

&card7

&card7 – *Gas Properties*

`fmu0`	R	Gas dynamic viscosity, in Pa·sec; [DEFAULT=1.716d-5]
`vispwr`	R	Power of viscosity variation; [DEFAULT=2/3] - no longer used
`gcp`	R	Specific heat capacity at constant pressure, in J/(kg K); [DEFAULT=1004.5]
`gamma`	R	Ratio of specific heat capacities; [DEFAULT=1.4]
`rgas`	R	Gas constant, in J/(kg K); [DEFAULT=287.16]
`prl`	R	Prandtl number; [DEFAULT=0.72]
`prt`	R	Turbulent Prandtl number; [DEFAULT=0.9]

&card8

&card8 – *Rotating Wheel Speed*

`omegax`	R	Rotational frequency, n, about the x-axis, in RPM. n is then used to calculate an angular velocity, ω_x divided by a reference speed, V_{ref}: $\omega_x/V_{ref} = 2\pi n/(60V_{ref})$ = `0.0` No rotation [DEFAULT] $>$ `0.0` Angular velocity vector points along x-axis $\left(\vec{\omega}_x \cdot \hat{i} > 0\right)$ $<$ `0.0` Angular velocity vector points opposite x-axis $\left(\vec{\omega}_x \cdot \hat{i} < 0\right)$
`omegax2`	R	Rotational frequency
`xrle`	R	Axial location at which hub rotation begins; [DEFAULT=-1.d32]
`xrte`	R	Axial location at which hub rotation ends[1]; [DEFAULT=1.d32]
`rte`	R	Maximum radius of the rotating wheel; [DEFAULT=1.d32]

[1]Rotational boundary conditions are only applied for $xrle \leq x \leq xrte$.

&card9

&card9 – *Postprocessing Options*

echo	L	Show and generate extra debugging information; [DEFAULT=.TRUE.]
debug	L	Show debug information, e.g. current subroutine location; [DEFAULT=.FALSE.]
debug2	L	Show additional debug information by providing residuals after each iteration, as opposed to when the buffer is full. Note: slows down the code significantly; [DEFAULT=.FALSE.]
intev_freq	I	Integrate and display physical parameters (such as, forces, etc.) every intev_freq iterations; [DEFAULT=100]
intev_freq_pt	I	Integrate and display physical parameters (such as, forces, etc.) every intev_freq_pt pseudo-time iterations; [DEFAULT=10]
res_freq	I	Frequency of writing residuals; [DEFAULT=1]
res_freq_pt	I	Frequency of writing residuals in pseudo-time; [DEFAULT=10]
itersave	I	Save solution every itersave iterations; [DEFAULT=100]
ires	I	Chooses which residual to monitor: = 1 Density = 2 x-momentum = 3 y-momentum = 4 z-momentum = 5 Energy
q_corrctn_limit	R	Stop Runge-Kutta iterations if q_correction is less than q_corrctn_limit. The value of q_corrctn_limit includes only mass, momentum, and energy. The turbulence model equations, if used, are not included. [DEFAULT=1.0d-14]
q_corrctn_ratio	R	Stop Runke-Kutta iterations if residuals drop by q_corrctn_ratio orders. The value of q_corrctn_ratio includes only mass, momentum, and energy. The turbulence model equations, if used, are not included. [DEFAULT=5.0d0]
MonitorMaxMach	L	Flag to indicate whether to monitor maximum Mach number; [DEFAULT=.TRUE.]
MaxMachThreshold	R	Display a warning if Mach number is higher than MaxMachThreshold; [DEFAULT=2.0d0]
MonitorMaxTemp	L	Flag to indicate whether to monitor maximum temperature; [DEFAULT=.TRUE.]

MaxTempThreshold	R	Display a warning if temperature is higher than MaxTempThreshold; [DEFAULT=1000.0d0]
reset_iter_counter	L	Reset iteration number to 0 if reset_iter_counter=.TRUE.; [DEFAULT=.FALSE.]
iflux_type	I	No longer used
Force_l	L	Flag for computing forces on the body; [DEFAULT=.TRUE.]
force_itype	I	Flag to determine which surfaces and how they are grouped when computing body forces = 0 All wall surfaces are grouped together [DEFAULT] = 1 Surfaces "extracted" using (i,j,k) information taken from ijkfile. Assumption of turbomachinery grids with surfaces defined on $j = 1$ layer of the multi-block domain's O-Grids. = 2 Surfaces defined by an external surface definitions file, surfint_file.
gradtest	L	Flag used to test gradients; [DEFAULT=.FALSE.]
mom_o	R	Point of (x,y,z) coordinates that define the origin about which moments are computed
flow_type	C	Flow type flag. Not used in general code
makemovie	L	Flag that allows to generate movies; [DEFAULT=.FALSE.]
mov_start_num	I	Initial movie file id number; [DEFAULT=1]
movie_freq	I	Frequency of movie output; [DEFAULT=1]
multirow_search_debug	L	Generate debug files for multi-row mesh
dumplimiter	L	Flag to generate Tecplot file to visualize the stencil-based solution limiter values. Use with caution, this will generate a Tecplot file at every stage of the Runge-Kutta, for every iteration – VERY SLOW! [DEFAULT=.FALSE.]

&card10

&card10 – *Relaxation Factors*

`resrlx`	R	Relaxation factors that multiply the variations that update state variables; vector of dimension 7

&cardprecon

&cardprecon – *Preconditioning*[2]

eps_minur	R	minimum reference velocity; [DEFAULT=0.0 for viscous flow and 0.1($u0^2$+$v0^2$+$w0^2$) for inviscid flow]
eps_deltap	R	pressure velocity coefficient; [DEFAULT=0.001]
eps_vis	R	viscous velocity coefficient; [DEFAULT=1.0]

[2]read only if `preconditioned=.true.` and `imp=.false.`

&cardforced

&cardforced – *Forced Vibration*

amp	R	Amplitude value, nondimensionlized by chord; [DEFAULT=0.0d0]
		for def_wave=cos, $y(t) = \texttt{amp} \cdot [1 - \cos(t)]/2$
		for def_wave=sin, $y(t) = \texttt{amp} \cdot \sin(t)$

forced_freq	R	Forcing frequency, in Hz; [DEFAULT=1.0d0]
def_type	C	Type of deformation: none, random, piston, fromfile, pitch, rbf_pitch, rbf_plunge, rbf_modal_for [DEFAULT='none']
def_wave	C	Type of deformation wave: cos or sin; [DEFAULT=cos]
forced_file	C	File name if def_type is fromfile; [DEFAULT='forced.def']
ea(1)	R	x-component of elastic axis location for pitching only [DEFAULT=0.0d0]
ea(2)	R	y-component of elastic axis location for pitching only [DEFAULT=0.0d0]
axisdir	I	Direction of elastic axis (assumed z) [DEFAULT=3]
pitch_rrat	R	Ratio of max and min radius for pitching, valid for def_type pitch, rbf_pitch and rbf_plunge [DEFAULT=3.0d0]
cascade	L	Flag that indicates linear cascade; [DEFAULT=.FALSE.]
stagger_deg	R	Stagger angle, in degrees - this must be in agreement with the mesh information; [DEFAULT=0.0d0]
mode_file	C	Name of the externally supplied file that contains bending/-torsion mode shapes for use with the rbf_modal_for deformation type, see Sec. 6.1.6 for a description of the file
nmode_vibe	I	Specify the number of modes to use. *Only valid* if def_type=rbf_modal_for [DEFAULT=1]
modal_freq_for(i)	R	Specify the deformation frequency, in Hertz, for each nmode_vibe vibration mode. *Only valid* if def_type=rbf_modal_for
modal_amp_for(i)	R	Specify the deformation amplitude for each nmode_vibe vibration mode. *Only valid* if def_type=rbf_modal_for
mode_select(i)	I	Select the desired mode from the mode shape definitions file. Negative values may also be specified to include: $-1 \rightarrow$ pitching about ea, $-2 \rightarrow$ plunging motion in direction of axisdir, and $-3 \rightarrow$ plunging motion normal to the stagger angle
ibpa_deg	R	Specify the inter-blade phase angle, in degrees, for turbomachinery forced vibration cases [DEFAULT=0.0]

`j0free_frac`	R	Specify fraction of the j-layers on the "tip" layer of a turbomachinery blade O-grid that are allowed to slide to permit three-dimensional deformations of a turbomachine blade [DEFAULT=0.5]
`sparse_rbf`	L	Radial basis function type flag, set to `.TRUE.` to use a sparse matrix implementation. **Not currently implemented, placeholder only** [DEFAULT=`.FALSE.`]

&cardrom

&cardrom – *Reduced-order Model*

rom	L	Run FOM or ROM; = .TRUE. Run reduced-order model (ROM) = .FALSE. Run full-order model (FOM) [DEFAULT]
nmodes	I	Number of POD modes; [DEFAULT=20]
restart_rom	L	Restart flag; [DEFAULT=.FALSE.]
check_ROM_jacobian	L	Check Jacobian of ROM; [DEFAULT=.FALSE.]
write_snapshots	L	Write snapshots flag; [DEFAULT=.FALSE.]
snapshot_freq	I	Number of iterations between writing snapshots; [DEFAULT=100]
vary_pback	L	Specifies whether back pressure oscillates; [DEFAULT=.FALSE.]
POF	R	Back pressure frequency$\times 2\pi$; [DEFAULT=100.d0]
POA	R	Back pressure amplitude; [DEFAULT=0.01d0]
dyn_avg	L	Specifies whether uses dynamic average or static average; [DEFAULT=.FALSE.]
dyn_phi	L	Specifies whether uses dynamic basis function or static basis functions; [DEFAULT=.FALSE.]
mode_dir	C	Folder where the modes.dat file is; [DEFAULT='modes/']
time_dir	C	Folder where the r?txxxx.ascii files are; [DEFAULT='time_coe/']
pback_file	C	File name for the back pressure vs. time; [DEFAULT='pback.dat']

&vortex

&vortex – *Initialize flow vortex*[3]

init_vortex	L	Flag for seeding a vortex [DEFAULT=.FALSE.]
init_pressure	L	Flag for re-initializing the pressure field. if .FALSE., the local pressure field is not modified [DEFAULT=.FALSE.]
v_xc	R	x-location of vortex [DEFAULT=24.5]
v_yc	R	y-location of vortex [DEFAULT=0.0]
v_strength	R	Vortex strength [DEFAULT=1.0]
max_radius	R	Maximum radius of initial vortex [Default=10^{25}]
vinode	I	Node number where the vortex's strength is to be measured and reported. Note: This feature is not currently parallelized [DEFAULT=1]

[3]Used for debugging the code

&patchbox

&patchbox – *Patch Initial Flowfield Using Hexagonal Regions*

`patch_q_box`	L	Flag to patch the initial flowfield using up-to five different hexagonal regions with different flow values [DEFAULT=`.FALSE.`]
`nbox`	I	Number of hexagonal patches to include, max of five allowed
`dbx_ratio(i)`	R	Density ratio, ρ_i/ρ_∞, used to set the density inside the i^{th} hexagonal region
`pbx_ratio(i)`	R	Pressure ratio, p_i/p_∞, used to set the pressure inside the i^{th} hexagonal region
`ubx(i)`	R	Normalized u-velocity, similar to `u0`, inside the i^{th} hexagonal region
`vbx(i)`	R	Normalized v-velocity, similar to `v0`, inside the i^{th} hexagonal region
`wbx(i)`	R	Normalized w-velocity, similar to `w0`, inside the i^{th} hexagonal region
`xbx(i,k)`	R	Specifies the upper ($k = 1$) and lower ($k = 2$) boundaries for the i^{th} hexagonal region in the x-direction
`ybx(i,k)`	R	Specifies the upper ($k = 1$) and lower ($k = 2$) boundaries for the i^{th} hexagonal region in the y-direction
`zbx(i,k)`	R	Specifies the upper ($k = 1$) and lower ($k = 2$) boundaries for the i^{th} hexagonal region in the z-direction

&shocktube

&shocktube – *Initialize Flow for Shock Tube Geometries*

`x_diaphragm`	R	x-coordinate of the diaphragm [DEFAULT=0.5]
`qst0_L(i)`	R	Dimensional (SI) state vector of the fluid upstream of the diaphragm [DEFAULT=$\{1, 0, 0, 0, 10^5\}$]
`qst0_r(i)`	R	Dimensional (SI) state vector of the fluid downstream of the diaphragm [DEFAULT=$\{1/8, 0, 0, 0, 10^4\}$]

6.1.2 vol.mesh File

In addition to the main input file, grid information is also needed. This information is provided through two files: vol.mesh and c2n.def (or vol.mesh_xxx and c2n.def_xxx for the parallel run, where xxx is the processor number). A vol.mesh file that was used on a sequential run can be used by the parallel version of `uns3d` by first copying vol.mesh into vol.mesh_000 and then appending the line '0 1 0' at the end of vol.mesh_000.

The Fortran commands that read the information from the vol.mesh file are:

```
open (np, file = gridfile, status = 'old', form = 'formatted')
read (np, *) ncell, nnode, nedge, nface, nbface

! read in grid point coordinates
do i = 1, nnode
   read (np, *) xnd(i), ynd(i), znd(i)
end do

! bface-point entries and boundary face condition idbcs
do i = 1, nbface
   read (np, *) np_bface(i), (ip_bface(i, n), n = 1, np_bface(i)), &
        idbcs(i)                                   ! bface condition
end do

! cell-face entries
do i = 1,ncell
   read (np, *) nf_cell(i)                          ! number of faces
   read (np, *) (if_cell(i, n), n = 1, nf_cell(i)) ! face number
   do n = 1, nf_cell(i)
      if (if_cell(i, n) == 0) then
         write (*,*) 'zero face number at cell', i, ' face', n
         stop 'in readstruc'
      end if
   end do
end do

! face-edge entries
do i = 1, nface
   read (np, *) ne_face(i)                          ! number of edges
   read (np, *) (ie_face(i, n), n = 1, ne_face(i)) ! edge number
end do
!
! edge-point entries
do i = 1, nedge
   read (np, *) (ij_edge(i, n), n = 1, 2) ! 2 end points for each edge
end do
```

6.1.3 c2n.def File

The Fortran commands that read the information from the c2n.def file are:

```
open (134, file = trim(c2nfile), iostat = istat, status = 'old')

read (134, *) ntet, npent5, npent6, nhex

icell = 0
! Read in tetrahedral cells
do i = 1, ntet
   icell = icell + 1
   read (134, *) (c2n(icell,j), j = 1, 4)
end do

! Read in pentahedral cells with 5 nodes
do i = 1, npent5
   icell = icell + 1
   read (134, *) (c2n(icell,j), j = 1, 5)
end do

! Read in pentahedral cells with 6 nodes
do i = 1, npent6
   icell = icell + 1
   read (134, *) (c2n(icell,j), j = 1, 6)
end do

! Read in hexahedral cells
do i = 1, nhex
   icell = icell + 1
   read (134, *) (c2n(icell,j), j = 1, 8)
end do
```

6.1.4 *.mapbc File

The *.mapbc file contains information on how the surface boundaries of the computational mesh are grouped together and what boundary condition types are applied to each grouping of boundary faces. The file contains a header that specifies the total number of boundary groupings. This information is used to allocate the boundary condition arrays found in the input namelists &card0, &card4, and &card5. The file is then made up of three columns of information. The first contains the index of the boundary face grouping, the second contains the boundary condition type information, and third column contains character strings that can describe each boundary face grouping. The following Fortran snippet can be used to read such a file:

```
read(iom,*) itmax
```

```
allocate(itype(itmax))
bc_list: do i = 1,itmax
  read(iom,*) j, itype(i), bc_name
end do bc_list
```

6.1.5 typlim_ids.dat File

The typlim_ids.dat file specifies the process IDs and type-limiter to be set on that process along with the needed options to run the specified type-limiter. The first line of the file specifies: (1) the number of processes for which the type-limiter will be modified, and (2) the number of added options to be set: 0=only the typlimiter is specified, 1=typlim_opt is also specified, 2=typlim_opt and venka_c are specified. The subsequent lines specify the process ID, typlim. Optionally, the lines can also specify typlim_opt and venk_c, depending on the number of added options, that is, the value of the second integer specified in the first line of the file. For example, if processes 10 and 268 should use a limiter different from the limiter used for the rest of processes, and if these two processes should use the Venkatakrishnan limiter (*i.e.*, typlim=2) with Forrest's modification (*i.e.*, typlim_opt=2), then the typlim_ids.dat file should be:

```
2   1
10  2   2
268 2   2
```

6.1.6 Modal Definition File

The "Modal Defintion File" characterizes possible bending/torsion structural modes that might be applied to a given geometry in the form of a forced deformation. These mode shapes must be computed by external means. The following Fortran code snippet reads the mode shape file:

```
read(iof,*) nmax_s, nmax_m
do j = 1,nmax_m
   read(iof,*) marker, n, natf_glob(j), maxd_glob(j)
   do i = 1,nmax_s
      read(iof,*) glob(i), mdx_g(i,j), mdy_g(i,j), mdz_g(i,j)
   end do
end do
```

where:

nmax_s	Number of global surface points used to describe each mode shape
nmax_m	Number of mode shapes contained within the file
marker	Dummy Character, must equal M
n	Dummy index, not used for anything in Version 5.4
natf_glob	Natural frequency of the mode shape

maxd_glob	Maximum mode shape deformation amplitude, used to normalize the current mode shape deformation components
glob	Global node number of a surface node
mdx_g	Mode shape deformation x-component (Δx) associated with the global node
mdy_g	Mode shape deformation y-component (Δy) associated with the global node
mdz_g	Mode shape deformation z-component (Δz) associated with the global node

6.1.7 LinearCascadeFV.def File

For the blade forced vibration simulation, the file LinearCascadeFV.def is needed. The Fortran commands that read the information from this file are:

```
read(io,'(A,A,A)') '"',trim(title),'"'
read(io,*) type, gap
read(io,*) npass, sigma_deg
read(io,*) npoints_proc(p), npfore(p), npaft(p), kmax
do n = 1,npoints_proc(p)
   read(io,*) blade_points_proc(n,p), blade_num_proc(n,p), &
        offset_surf(n,p)
end do
do n = 1,npfore(p) ! H-grids only
   read(io,*) im_fore_proc(n,p), k_fore(n,p), gamma_fore(n,p)
end do
do n = 1,npaft(p) ! H-grids only
   read(io,*) im_aft_proc(n,p), k_aft(n,p), gamma_aft(n,p)
end do
do n = 1,kmax ! H-grids only
   read(io,*) IIN(n,p), ILE(n,p), ITE(n,p), IEX(n,p)
end do
```

The definitions of the variables used in the LinearCascadeFV.def file are:

title	C	Case title, for file description
type	C	Defines the grid type. Letter case independent = H or h $\rightarrow$ H-type grid = C or c $\rightarrow$ C-type grid
gap	R	Linear distance in the y-direction between a master and its slave node for linear cascade geometries.
npass	I	Number of passages, also the number of blades

sigma_deg	R	inter-blade phase angle in degrees
npoints_proc	I	Total number of blade surface nodes located found on the local processor mesh.
npfore	I	Total number of master periodic nodes upstream of the leading edge found on the current processor mesh. (>0 for H-grids; =0 for C-grids)
npaft	I	Total number of master periodic nodes downstream of the trailing edge found on the current processor mesh. (>0 for H-grids; =0 for C-grids)
kmax	I	Total number of spanwise layers
blade_points_proc	I	Blade surface node numbers for the current processor mesh.
blade_num_proc	I	Blade number for given surface node. $= 0, 1, 2, ...,$ npass
offset_surf	L	.TRUE. for surface points that are offset from the other half of the airfoil surface (see Fig. 6.2). .FALSE. for all other points.
im_fore_proc	I	Master periodic boundary node numbers upstream of the leading edge for the current processor mesh.
k_fore	I	Master periodic boundary node k-indices (upstream nodes)
gamma_fore	R	Ratio of arc lengths needed to linearly interpolate the new y and z coordinates of the deformed master periodic boundary. Computed using the undeformed mesh. (upstream nodes) $= (s_i - s_{i_{IN}}) / (s_{i_{LE}} - s_{i_{IN}})$, where s is an arc length and s_i is the arc length to any point along the edge.
im_aft_proc	I	Master periodic boundary node numbers downstream of the trailing edge for the current processor mesh.
k_aft	I	Master periodic boundary node k-indices (downstream nodes)
gamma_aft	R	Ratio of arc lengths needed to linearly interpolate the new y and z coordinates of the deformed master periodic boundary. Computed using the undeformed mesh. (downstream nodes) $= (s_i - s_{i_{TE}}) / (s_{i_{EX}} - s_{i_{TE}})$
IIN	I	Inlet i-index on the master periodic boundary for a given k-layer.
ILE	I	Leading edge i-index on the master periodic boundary for a given k-layer.
ITE	I	Trailing edge i-index on the master periodic boundary for a given k-layer.
IEX	I	Outlet i-index on the master periodic boundary for a given k-layer.

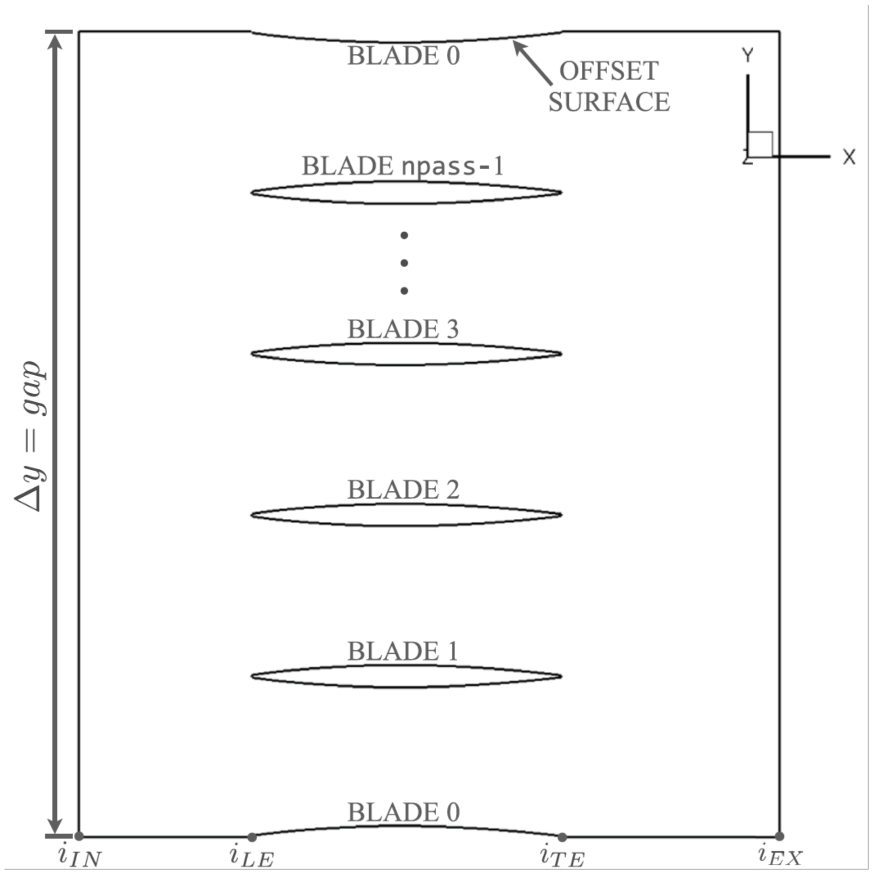


Figure 6.2: Typical H-type linear cascade geometry.

6.1.8 SurfaceSetInt.def File

An optional file SurfaceSetInt.def can be used to identify specific surfaces on which to compute surface forces. If this file is not present, the surface forces will be calculated on all wall boundaries.

The SurfaceSetInt.def file is created during the mesh generation process. The Fortran commands in splitmesh which read the parallelized definition files (needed by uns3d) are:

```
read(io,*) nsurf
read(io,*) nface_proc(myid)
do n = 1,nface_proc(myid)
   read(io,*) nsurf_proc(n,myid), iface_proc(n,myid)
end do
```

The definitions of the variables used in the SurfaceSetInt.def file are:

nsurf	I	Total number of surfaces being defined.
nface_proc	I	Total number of mesh faces in the parallel definition files.
nsurf_proc	I	Surface number of the current mesh face. (nsurf_proc $\leq$ nsurf)
iface_proc	I	Local mesh face number.

6.2 uns3d Output Files

- monitorerr1.dat - monitorerr5.dat (UNIT=71-75)
  = log10 [max_over_nnode (abs (q(n+1) - q(n))], where q(n) is the state vector of primitive variables at iteration n

  for steady flows the file content is:
  iter, err?
  for unsteady flows the file content is:
  time, err?

- monitoravg1.dat - monitoravg5.dat (UNIT=81-85)
  average residual

- monitoriofl.dat
  mass flow rate error between inlet and outlet

- monitormach.dat
  maximum Mach number

- monitormax1.dat - monitormax5.dat (in `write_residual`)
  maximum residual of density, u, v, w, and pressure

  for steady flows the file content is:
  iter, rms?
  for unsteady flows the file content is:
  time, rms?

- monitorforx.dat, monitorfory.dat, and monitorforz.dat
  These three files return the time variation of the forces components.
- `estimated_run_time.dat`
  File that gives the estimated run time in seconds, minutes or hours. For unsteady, dual-time stepping simulations, the estimated run time assumes the worst case, that is, all subiterations are needed for each time step.
- `summarize.dat`
  File that saves inlet and outlet mass flow rates, their ratio, ratio of outlet to inlet stagnation pressures, ratio of outlet to inlet static pressures, ratio of outlet to inlet densities, total-to-total and total-to-static efficiencies with a frequency specified by `intev_freq`.
- `Forces-Moments.dat`
  File that saves forces and moments, if `force_l=.true.`
- `txt/*.txt_xxx`[4]
  Description of the input data and history of the run. The name of the file `txt/*.txt_xxx` is specified by variable `rsdfile` of &cardf in the main input file.
- `out/*.dat_xxx`
  File that contains the state variables rho, u, v, w, p and a^2 at the end of the run. The name of the file `out/*.dat_xxx` is specified by variable `fileoutq` of &cardh in the main input file.
- `out/*.tur_xxx`
  File that contains the state variables of the turbulence model at the end of the run. The name of the file `out/*.tur_xxx` is specified by variable `fileoutqt` of &cardi in the main input file.
- `out/wdist.dat_xxx`
  Files that save the distances from nodes to walls. These distances are calculated only once, unless the mesh is deforming. The names of these files are hardwired.
- `plt/*.plt_xxx`
  Files that save the flow data, if `dump_tecplot=.true.`, in Tecplot format.

[4] `xxx` is the processor number

- `plt/boundary_face_values.tec_xxx`
  Files that save the IDs of the boundary faces, if `bcplot=.true.`, in Tecplot format. If these files are already present, they are not overwritten unless `debug=.true.` The names of these files are hardwired.

- `plt/*.yplus.plt_xxx`
  Files that save the values of the y^+ number (along the walls), if `dump_yplus=.true.` and `invis=.false.`, in Tecplot format. The name of the file `plt/*.yplus_xxx` is specified by variable `tecplot_name` of &cardf in the main input file.

Chapter 7

`uns3d` Graphical User Interface

7.1 Introduction

The purpose of this chapter is to discuss the graphical user interface (GUI) that is provided in the `uns3d` software package. TclTk, the language the GUI is written in, is discussed first. Next, instructions are given on how to install the GUI and the additional required software. The last section includes an overview of the GUI and instructions on how to use it.

7.2 TclTk

This section describes the language that the GUI was written in. The GUI is written in TclTk. Tcl is a scripting language developed in the 1980's by John Ousterhout [15, xlix] and Tk is a toolkit for graphical user interfaces [15, li]. Tcl is a simple interpretive language that is platform-independent, making scripts written in the language highly-portable. Tcl is embeddable and extensible in applications because its interpreter is a library of C functions [9, xxxi]. This flexibility makes Tcl a perfect platform for GUI development.

GUI development in Tcl is facilitated by Tk. Tk was also developed by Ousterhout, and like Tcl, is a library of C functions [9, xxxi]. Tk allows the coder to create Tcl scripts to generate the graphical objects instead of writing more difficult C functions [9, xxxi]. Tk is also widely used by other languages, such as Python, to handle GUI generation.

The benefit of using TclTk as the basis for the GUI is multifaceted. TclTk is a powerful language that can easily be learned in a day. As the core language is written in C, many of the details other languages would require the coder to know and handle, such as C++ or Java as well as C, are hidden and handled by the core language [9, xxxi]. If a required functionality does not exist within the core language, the coder only needs to write an additional script to create it. Additionally, since TclTk is an interpreted language, development, debugging, and testing on the fly is extremely easy without the need to recompile or relaunch the application. TclTk is cross platform, so a TclTk application originally written for Linux is typically portable to Mac or Windows without any modifications. Also, TclTk is a glue language, allowing the coder to piece together several applications written in any number of languages into one larger code within a single framework, making cross application communication possible. Finally, TclTk, and most of the available extensions to the language, are freely available under a BSD license. All these reasons make TclTk the great language to generate the GUI in.

7.3 Installation

This section provides step-by-step instructions for obtaining and installing TclTk and the GUI. As mentioned in the previous section, TclTk is freely available under a BSD license. To obtain TclTk, download the latest maintained package (version 8.5) from Active State (www.activestate.com) for your platform. This is required to install the basic core packages.

Installation of the core packages is done by double clicking the executable or disk image. At the end of the basic installation, the install path for TclTk will need to be written down and added to the $PATH system variable of the machine. For the Bash shell on Unix/Linux operating system, this can be added to either the .bashrc or .bash_profile file as follows:

```
PATH=$PATH:/opt/ActiveTcl-8.5/bin/
export PATH  .
```

To install all of the additional required packages, the following command using `teacup`, TclTk's package handler, needs to be run from a terminal:

```
sudo teacup install --force --with-recommends

 ActiveState::ActiveTcl85  .
```

Additionally, `Expect` needs to be installed, which handles subprogram communication and execution. `Expect` can be installed as follows:

```
sudo teacup install - -force - -with-recommends Expect  .
```

To view the residual output from the GUI, the plotting program `xmgrace` will need to be installed. `xmgrace` is available in the package grace, which is freely available. For Linux users, grace is installed as follows:

```
sudo apt-get install grace
```

For Mac users, grace can be installed from Fink:

```
sudo fink install grace .
```

After installing TclTk, the .tgz file containing the GUI source files should be untarred using the following command:

```
tar -xvf uns3d_gui.tgz  .
```

The installation requires that the location of the `uns3d` executable be specified in the file 'executable_location.txt'. The file 'executable_location.txt' is located in the GUI source directory. The location of the `uns3d` executable must be specified using the absolute path, as follows:

```
$UNS3DHOME/uns3d  .
```

This completes installation of the GUI onto the computer. The next section will discuss how to use the GUI.

7.4 Using the GUI

This section describes how to use the GUI to create and run simulations. To run the GUI from the command line, type the following command in a terminal:

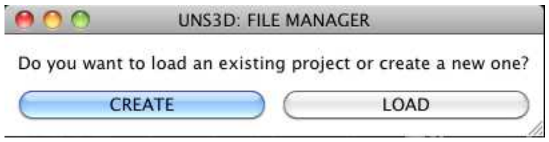
```
./uns3d_gui.tcl
```

This launches the GUI, presenting a splash screen with the `uns3d` logo. This screen is shown in Figure 7.1. Next, the user is allowed to create or load an existing project. This is discussed in the following subsection.

Figure 7.1: Splash screen for GUI.

7.4.1 Creating or Loading a Project

After the splash screen vanishes, the user is presented with the "File Manager" window, shown in Figure 7.2. The "File Manager" window allows the user to either create or load an existing project by clicking either the Create or Load button, respectively.


UNS3D: FILE MANAGER

Do you want to load an existing project or create a new one?

Figure 7.2: "File manager" screen for GUI.

Creating a Project

Clicking the Create button brings the user to the project creation dialog box, shown in Figure 7.3, where the user names the project. After choosing an appropriate name and clicking save, a .proj file, which stores the information about all of the runs for this project and sets variables for the GUI, is created along with a folder that stores the input files and all of the files created by `uns3d`. The user is then presented with another dialog box where the name is given for the first input file that the user would like to run, which looks similar to the project creation dialog box. Clicking save then writes the newly created project to the list of available projects in the 'project_locations.txt' file. This information is used when loading an existing project. Once this step is completed, the user is brought to the main GUI screen.

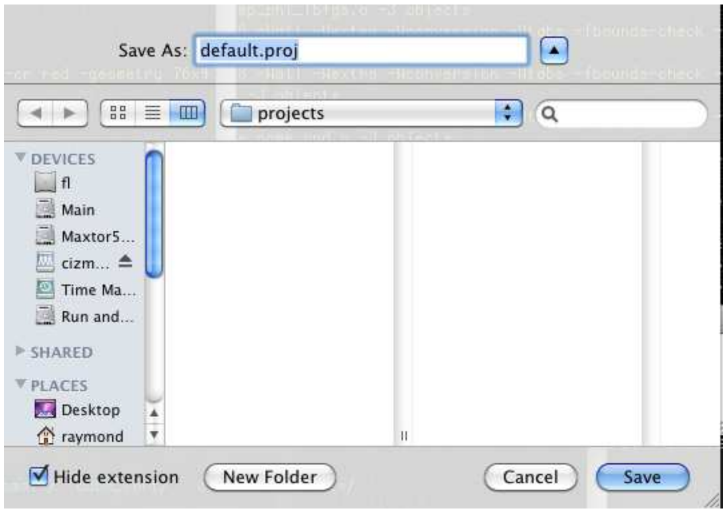


Figure 7.3: "Create a New Project" screen for GUI.

Loading a Project

Clicking the Load button brings the user to the project load dialog box, shown in Figure 7.4. The user is then presented with a list of available projects from which they can choose from. Each project also lists the currently available input files for the project, from which the user can choose to load a particular input file. Clicking the project name will by default load the last input file in the list. Once this step is completed, the user is brought to the main GUI screen.

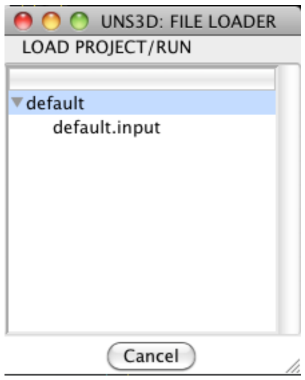
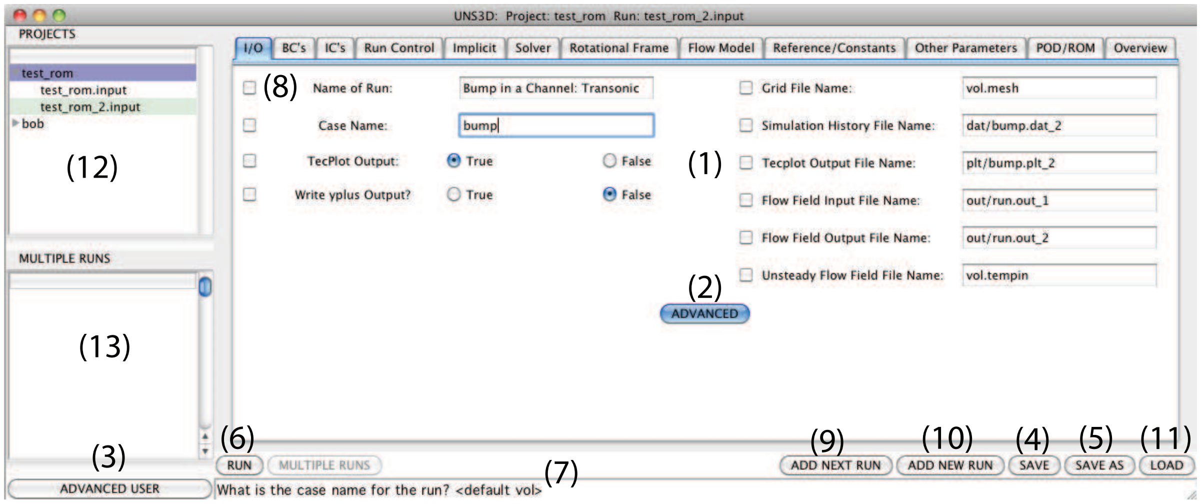


Figure 7.4: "Load an Existing Project" screen for GUI.

7.4.2 The Main Window

After the user has either created or loaded a project, they are presented with the main screen of the GUI, shown in Figure 7.5. From this screen, the user can generate an input file for `uns3d`, select another project, create multi-run jobs, execute and run `uns3d`, and view the residual history real-time. These options are explained in the following subsections.



UNS3D: Project: test_rom Run: test_rom_2.input

PROJECTS: test_rom (test_rom.input, test_rom_2.input), bob

Name of Run:	Bump in a Channel: Transonic
Case Name:	bump
TecPlot Output:	[✓ True] False
Write yplus Output?	True [✓ False]
Grid File Name:	vol.mesh
Simulation History File Name:	dat/bump.dat_2
Tecplot Output File Name:	plt/bump.plt_2
Flow Field Input File Name:	out/run.out_1
Flow Field Output File Name:	out/run.out_2
Unsteady Flow Field File Name:	vol.tempin

What is the case name for the run? <default vol>

Figure 7.5: The main screen of the GUI.

Creating Input

From the main screen, the user can create input for a current production run, edit an existing input file, or create a new input file from the current input file. The input is broken up into different tabs that best describe what the input is (item 1 on Figure 7.5). These tabs are as follows:

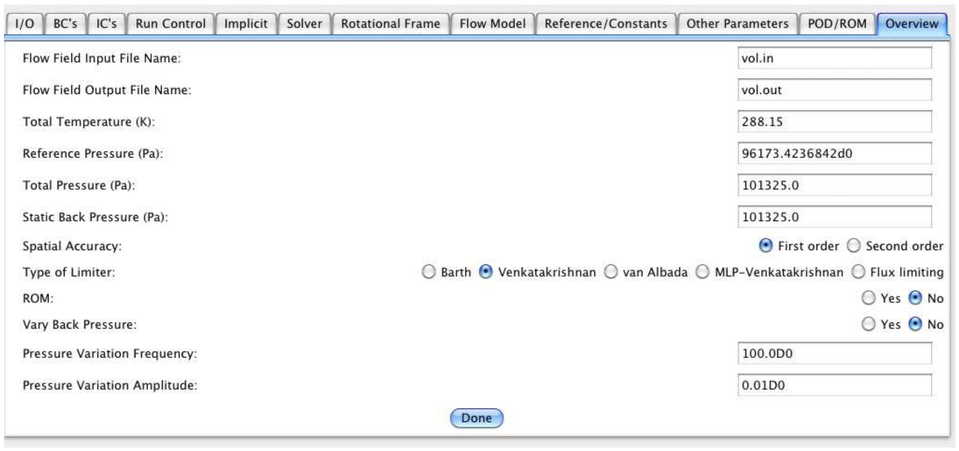
- Input/Output (I/O)
- Boundary Conditions (BC)
- Initial Conditions (IC)
- Run Control
- Implicit
- Solver
- Rotational Frame
- Flow Model
- Reference/Constants
- Other Parameters

- POD/ROM
- Overview
- Run

An explanation of each of the variables in the tabs can be found in Chapter 6. Each tab is split into basic and advanced sections. Basic input variables are the variables most often modified. Advanced variables, which can be viewed/edited by clicking either the Advanced button (item 2 on Figure 7.5) on each tab or clicking the Advanced User button on the main screen (item 3 on Figure 7.5), are those variables that are less often modified. Once the user is satisfied with the input, the project and input file can be saved by clicking either the Save, Save As, or Run button (items 4, 5, and 6 respectively on Figure 7.5).

When editing variables in the tabs, the user can hover over the variable name or entry and see information about the variable at the bottom of the screen in the message box, shown in Figure 7.5 (item 7). This dialog box presents the user with information about the variable, as well as its default value.

Adjacent to each variable are check boxes (item 8 on Figure 7.5). The check boxes are used to populate the Overview tab. The Overview tab allows the user to quickly check and review important input in one place. The user can modify the input from this tab as well. When the project is saved, those variables visible in the Overview tab are saved such that the user can view these variables each time the GUI is opened. The overview tab is shown in Figure 7.6.



Flow Field Input File Name:	vol.in
Flow Field Output File Name:	vol.out
Total Temperature (K):	288.15
Reference Pressure (Pa):	96173.4236842d0
Total Pressure (Pa):	101325.0
Static Back Pressure (Pa):	101325.0
Spatial Accuracy:	[✓ First order] Second order
Type of Limiter:	Barth [✓ Venkatakrishnan] van Albada MLP-Venkatakrishnan Flux limiting
ROM:	Yes [✓ No]
Vary Back Pressure:	Yes [✓ No]
Pressure Variation Frequency:	100.0D0
Pressure Variation Amplitude:	0.01D0

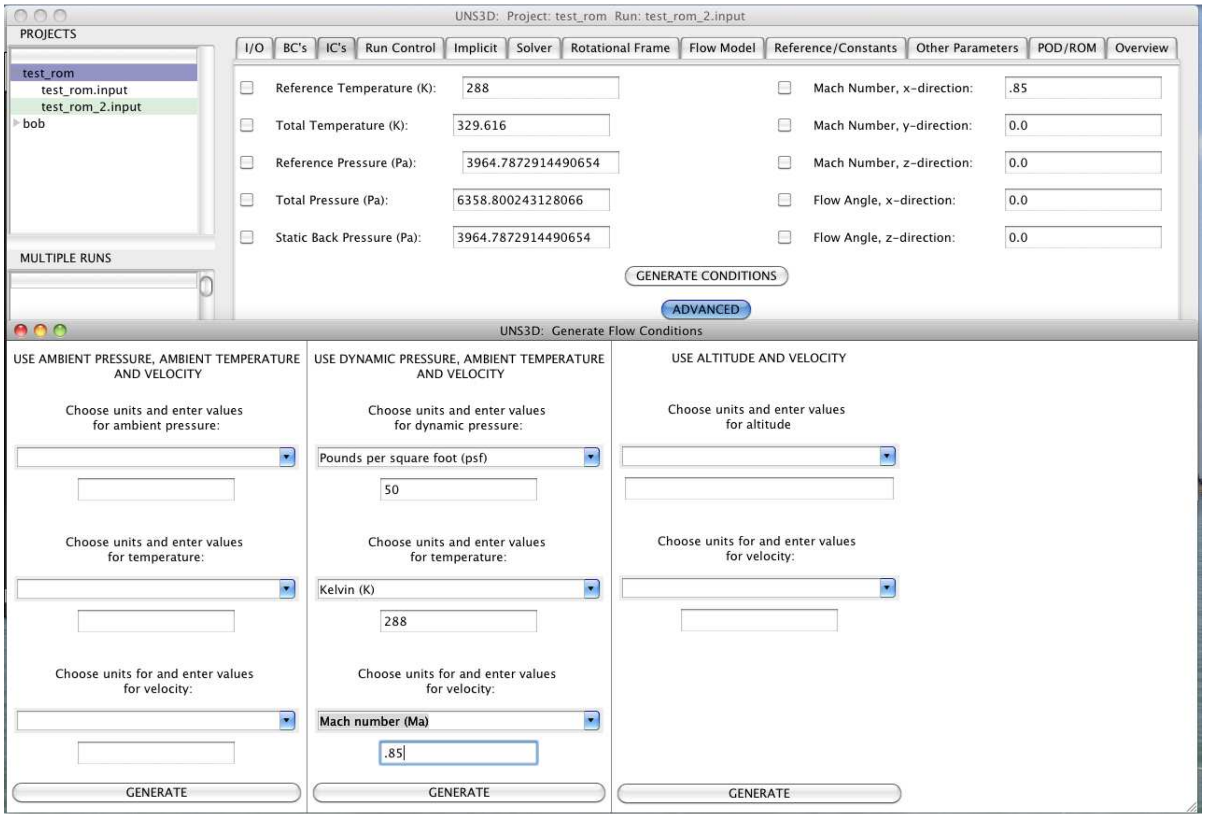
Figure 7.6: An example overview tab for the GUI.

Clicking the Add Next Run (item 9 on Figure 7.5) button will allow the user to generate a subsequent input file for a restarted or continued case. The user is presented with a dialog box where he can name the next input file. After choosing the name, the GUI will automatically increment file names (*e.g.*, from 1 to 2). The GUI will also change the input variable that specifies the type of run from new to restart. Clicking the Add New Run button (item 10 on Figure 7.5) will generate a completely new file using the code defaults. The user also has the option of loading an existing input file from other cases to populate the input by clicking the Load button (item 11 on Figure 7.5).

Of special note is the 'Generate Flow Conditions' button on the IC tab, shown in Figure 7.7. Clicking this button pops up a separate window. This window allows the user to set the run conditions for the current input file following the isentropic relations. Three options are given for generating initial conditions:

- Ambient Pressure, Ambient Temperature, Velocity
- Dynamic Pressure, Ambient Temperature, Velocity
- Altitude and Velocity

Choosing any of these options, the user inputs the relevant data and select units and then clicks the Generate button for the chosen option. This action computes the total and reference values for the input and then updates the relevant variables in the GUI.



Reference Temperature (K): 288
Total Temperature (K): 329.616
Reference Pressure (Pa): 3964.7872914490654
Total Pressure (Pa): 6358.800243128066
Static Back Pressure (Pa): 3964.7872914490654
Mach Number, x-direction: .85
Mach Number, y-direction: 0.0
Mach Number, z-direction: 0.0
Flow Angle, x-direction: 0.0
Flow Angle, z-direction: 0.0

USE AMBIENT PRESSURE, AMBIENT TEMPERATURE AND VELOCITY
Choose units and enter values for ambient pressure:
Choose units and enter values for temperature:
Choose units for and enter values for velocity:

USE DYNAMIC PRESSURE, AMBIENT TEMPERATURE AND VELOCITY
Choose units and enter values for dynamic pressure: Pounds per square foot (psf) 50
Choose units and enter values for temperature: Kelvin (K) 288
Choose units for and enter values for velocity: Mach number (Ma) .85

USE ALTITUDE AND VELOCITY
Choose units and enter values for altitude
Choose units for and enter values for velocity:

Figure 7.7: "Generate Flow Conditions" utility for the GUI.

Selecting Another Project

The user is allowed to select another project from the project tree view on the main screen, shown in Figure 7.8 (item 12 on Figure 7.5). The project tree view, just like the load screen, displays all existing projects and the associated input files. From the tree view, the user can select another input file from the list or another project. The user is also allowed to delete or rename inputs or projects from this tree view.

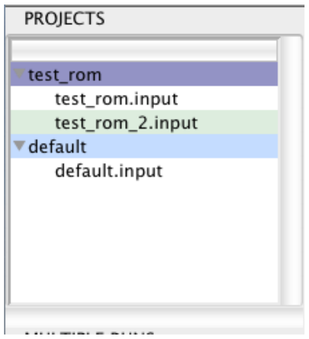


Figure 7.8: The project tree view for the GUI on the main screen.

Create Multi-Run Jobs

The user can create multi-run jobs using the multiple run tree view, shown in Figure 7.9 (item 13 on Figure 7.5). Multi-run jobs are simulations with several input files run sequentially. To create a multi-run job, the user right clicks input files in the project tree view and selects 'Add to Run List'. In this tree view, the user can delete input files from the list. Once the user is satisfied with the list of the input files, the Multiple Runs button is clicked. This generates a script file with the executable and the input files such that the system can handle them all at once. Once the run script is created, the run tab is opened. This is discussed in the next section.

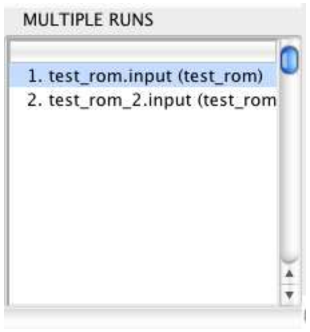
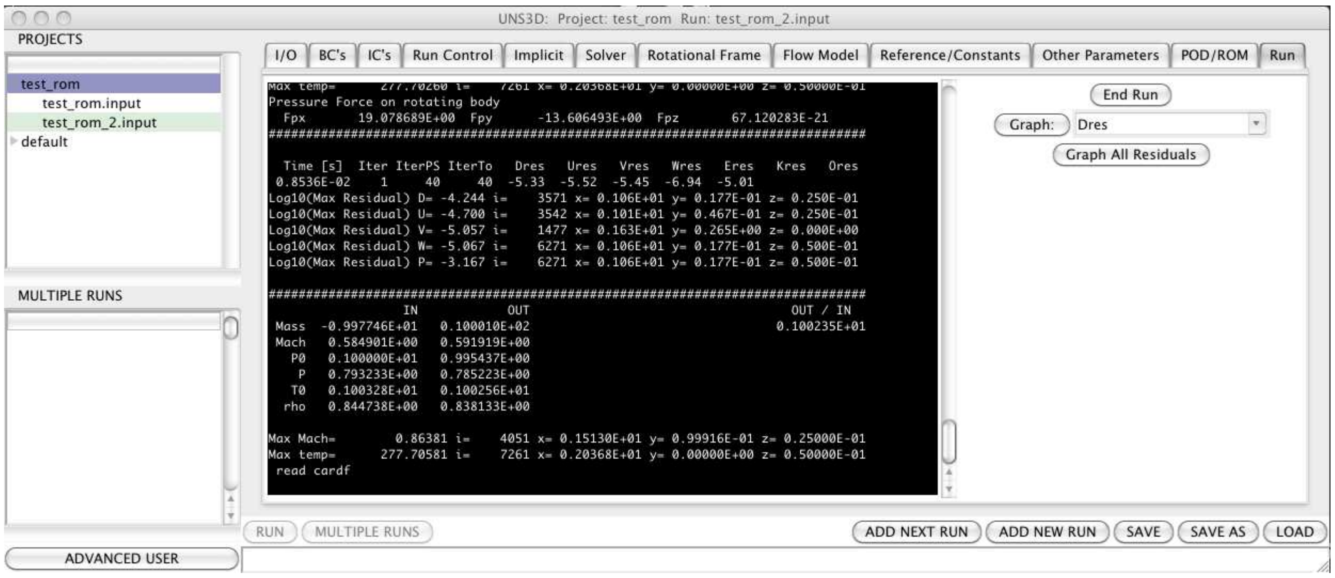


Figure 7.9: The multiple run tree view for the GUI on the main screen.

Running the Code

After the user is satisfied with the input for `uns3d`, running the code with the GUI is straight forward. By clicking either the Run or Multiple Runs button, the user sets in motion a series of events. The GUI will save the current state of the GUI into the project file, and the current input file is also saved. The GUI then checks the input file and generates folders for `uns3d` if the user has requested certain output from the code to be placed in them, such as the flow field results and the restart files. Once these folders have been created, the GUI then creates the Run tab. On the run tab, shown in Figure 7.10, a text box is created for output from `uns3d`. The GUI reads a log file generated from the code and displays the information in the text box for the user to read. If at any time the user is not satisfied with the progress of the run, the user can kill `uns3d` by pressing the

End Run button, which kills the process and returns the user to the main screen. On the Run tab, the user can also graph the residual histories computed from `uns3d`. Clicking the Graph button will allow the user to plot the residuals in `xmgrace`. Once the run is ended, the run buttons are reactivated, and the user can proceed to either generating new input data for the next run or close and exit the GUI.



```
Max temp=      277.70260 i=    7261 x= 0.20368E+01 y= 0.00000E+00 z= 0.50000E-01
Pressure Force on rotating body
  Fpx       19.078689E+00  Fpy       -13.606493E+00  Fpz       67.120283E-21
################################################################################

  Time [s]  Iter IterPS IterTo   Dres   Ures   Vres   Wres   Eres   Kres   Ores
 0.8536E-02    1     40     40  -5.33  -5.52  -5.45  -6.94  -5.01
Log10(Max Residual) D= -4.244 i=    3571 x= 0.106E+01 y= 0.177E-01 z= 0.250E-01
Log10(Max Residual) U= -4.700 i=    3542 x= 0.101E+01 y= 0.467E-01 z= 0.250E-01
Log10(Max Residual) V= -5.057 i=    1477 x= 0.163E+01 y= 0.265E+00 z= 0.000E+00
Log10(Max Residual) W= -5.067 i=    6271 x= 0.106E+01 y= 0.177E-01 z= 0.500E-01
Log10(Max Residual) P= -3.167 i=    6271 x= 0.106E+01 y= 0.177E-01 z= 0.500E-01

################################################################################
                  IN            OUT                                   OUT / IN
 Mass  -0.997746E+01   0.100010E+02                                 0.100235E+01
 Mach   0.584901E+00   0.591919E+00
   P0   0.100000E+01   0.995437E+00
    P   0.793233E+00   0.785223E+00
   T0   0.100328E+01   0.100256E+01
  rho   0.844738E+00   0.838133E+00

Max Mach=        0.86381 i=    4051 x= 0.15130E+01 y= 0.99916E-01 z= 0.25000E-01
Max temp=      277.70581 i=    7261 x= 0.20368E+01 y= 0.00000E+00 z= 0.50000E-01
 read cardf
```

Figure 7.10: The run tab with code output from the GUI.

Appendix A

Boundary Condition Descriptions

A.1 Symmetry Boundary Conditions

This section describes the various implementations of the symmetry boundary condition within `POD-UNS3D`. Figure A.1 shows an illustration of single symmetry boundary face, "face n". In the figure:

- The closed circles represent the nodes of the mesh
- $\vec{q}_i$ is the state vector at node i
- The dashed lines are the boundaries of the k sub-domains of the boundary face
- The open squares are the boundary face quadrature points located in each sub-domain of the face
- $\hat{n}_{n,k}$ is the boundary face normal of the k^{th} sub-domain

Please use this figure for reference throughout this section.

A.1.1 `isymbc` = 0: "Pseudo-Ghost Cell" Implementation

The "pseudo-ghost cell" implementation seeks to mimic a true ghost-cell implementation of a symmetric boundary. The boundary state vector is defined using the values stored at the adjacent node without any form of interpolation. The boundary state vector for the Navier–Stokes equations is given by

$$\vec{q}_{b_{n,k}} = \left\{ \begin{matrix} \rho_i \\ \vec{v}_i - 2\left(\hat{n}_{n,k} \cdot \vec{v}_i\right)\hat{n}_{n,k} \\ p_i \end{matrix} \right\} \tag{A.1}$$

where the resultant boundary face velocity vector is a mirrored copy of the nodal velocity vector. No interpolation was used to define $\vec{q}_b$ so that the values are equal to or directly mirror the values used on the other side of the flux function when computing the convective fluxes. The boundary state vector for the turbulence model is defined to be identical to the adjacent nodal values: $\vec{qt}_{b_{n,k}} = \vec{qt}_i$.

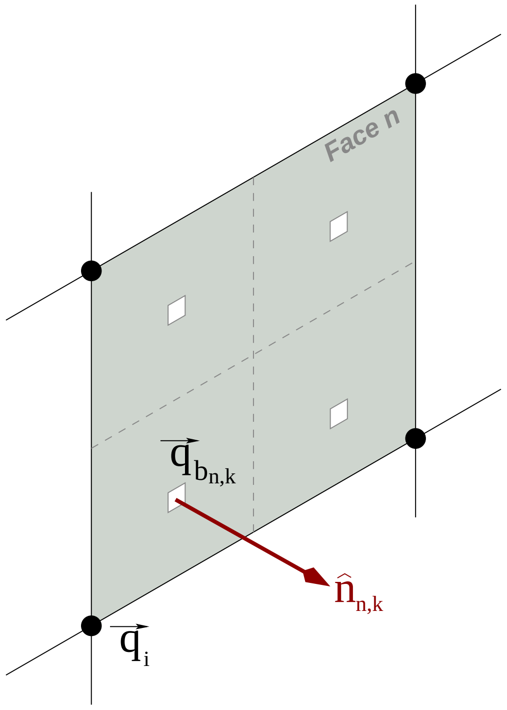


Figure A.1: Illustration of a symmetry boundary face, "face n".

This particular implementation of the symmetric wall boundary condition requires a modification of the gradient vectors of each state variable. The normal component of the gradient vector is manually zeroed after the calculation of the gradients is completed:

$$\vec{\nabla}\phi\Big|_b = \vec{\nabla}\phi\Big|_b - \left(\vec{\nabla}\phi\Big|_b \cdot \hat{n}_b\right)\hat{n}_b$$

where ϕ can be any scalar variable and $\hat{n}_b$ is the boundary face normal ($\hat{n}_b = \hat{\imath}$, $\hat{n}_b = \hat{\jmath}$, or $\hat{n}_b = \hat{k}$). The gradient vector modification is only completed if `isymbc` = 0.

A.1.2 `isymbc` = 1: "Inviscid Wall" Implementation

This symmetry boundary condition implementation treats the boundary as if it were an inviscid wall, and seeks to satisfy only the no-penatration condition. Nodal values of the state vector are interpolated to the quadrature points of a given face using the subroutine `linearint2d`. The boundary state vector, defined using interpolated values, is given by

$$\vec{q}_{b_{n,k}} = \left\{ \begin{matrix} \rho'_{n,k} \\ \vec{v}'_{n,k} - \left(\hat{n}_{n,k} \cdot \vec{v}'_{n,k}\right)\hat{n}_{n,k} \\ p'_{n,k} \end{matrix} \right\} \tag{A.2}$$

where the interpolated values are indicated with a prime. No interpolation is used for the turbulent boundary state vector, *i.e.*, $\vec{qt}_{b_{n,k}} = \vec{qt}_i$.

A.2 Farfield Boundary Conditions

This section presents, in detail, the different options for describing a farfield boundary currently implemented in `POD-UNS3D`. Table A.1 lists the available options. The default option in `POD-UNS3D` is the slip wall condition, `ifarbc` = 0. Refer to Section A.1.2 for a description of the slip wall implementation, and to Section A.2.1 for a description of the extrapolation condition. The Riemann invariant implementation is now described.

Table A.1: Farfield boundary implementation options.

Farfield Flag (**`ifarbc`**)	**Implementation Type**
0	Slip wall condition
1	Extrapolation condition
2	Riemann Invariant

A.2.1 `ifarbc` = 1: Extrapolation

The extrapolation boundary condtion as implemented in `POD-UNS3D` uses interpolated values of the state vector at the face quadrature points to define the boundary state vector. The boundary state vector is then given by

$$\vec{q}_b = \begin{Bmatrix} \rho_f \\ \vec{V}_f \\ p_f \end{Bmatrix}, \tag{A.3}$$

where the subscript f refers to an interpolated value. The current interpolation scheme used in `POD-UNS3D` results in a linear interpolation of the variables across a given face.

A.2.2 `ifarbc` = 2: Riemann Invariant

The Riemann invariant boundary condition uses the incoming and outgoing characteristcs to determine the local Mach number normal to the boundary. The local normal Mach number is then used to determine the character of the flow and whether the flow is entering or leaving the domain.

To determine the Riemann invariants, an inner and outer state must first be defined. The outer state is defined to be the freestream conditions: $\vec{q}_\infty = \{\rho_\infty,\, \vec{V}_\infty,\, p_\infty\}^T$. On a given boundary face, the inner state is taken directly from the boundary node nearest the quadrature point, resulting in the inner state vector $\vec{q}_i = \{\rho_i,\, \vec{V}_i,\, p_i\}^T$. The outgoing and incoming Riemann invariants are then defined, respectively, as

$$\begin{aligned} R^+ &= \begin{cases} \vec{V}_i \cdot \hat{n} + \frac{2c_i}{\gamma-1}, & \text{if } \left|\vec{V}_i \cdot \hat{n}\right| < c_\infty \\ \vec{V}_\infty \cdot \hat{n} + \frac{2c_\infty}{\gamma-1}, & \text{if } \vec{V}_i \cdot \hat{n} \le -c_\infty \end{cases} \\ R^- &= \begin{cases} \vec{V}_\infty \cdot \hat{n} - \frac{2c_\infty}{\gamma-1}, & \text{if } \left|\vec{V}_i \cdot \hat{n}\right| < c_\infty \\ \vec{V}_i - \frac{2c_i}{\gamma-1}, & \text{if } \vec{V}_i \cdot \hat{n} \ge c_\infty \end{cases} \end{aligned} \tag{A.4}$$

where $\hat{n}$ is the outward facing bounadry normal, and the values of the inner and outer sound speed were computed by $c_i = \sqrt{\gamma p_i/\rho_i}$ and $c_\infty = \sqrt{\gamma p_\infty/\rho_\infty}$, respectively. The form of the Riemann invariant equations in Eqn A.4 is dependent on the magnitude and direcetion of the velocity. There can be no outgoing Riemann invariant if supersonic inflow ($\vec{V_i} \cdot \hat{n} \leq -c_\infty$) is found, necessitating a special definition for the outgoing invariant. Likewise, there can be no incoming invariant if supersonic outflow ($\vec{V_i} \cdot \hat{n} \geq c_\infty$) is detected, resulting in the modified form.

The computed incoming and outgoing invariants are then used to compute both the speed of sound at the boundary,

$$c_b = \frac{\gamma - 1}{4} \left(R^+ - R^- \right),$$

and the magnitude of the velocity normal to the boundary,

$$V_\perp = \frac{1}{2} \left(R^+ + R^- \right).$$

The sign of the boundary normal velocity magnitude will dictate whether the flow is entering or leaving the domain. A positive value of $V_\perp$ indicates that the flow is exiting the domain. Based on the direction of the flow, the components of the velocity at the boundary and the entropy can then be calculated using

$$\vec{V_b} = \begin{cases} \vec{V_i} + \left(V_\perp - \vec{V_i} \cdot \hat{n} \right) \hat{n}, & \text{if } V_\perp > 0 \\ \vec{V}_\infty + \left(V_\perp - \vec{V}_\infty \cdot \hat{n} \right) \hat{n}, & \text{if } V_\perp \leq 0 \end{cases} \tag{A.5}$$

$$s_b = \begin{cases} \frac{c_i^2}{\gamma \rho_i^{\gamma-1}}, & \text{if } V_\perp > 0 \\ \frac{c_\infty^2}{\gamma \rho_\infty^{\gamma-1}}, & \text{if } V_\perp \leq 0 \end{cases} \tag{A.6}$$

The bounadry state vector is then defined as $\vec{q_b} = \{\rho_b, \vec{V_b}, p_b\}^T$, where the density is given by

$$\rho_b = \left(\frac{c_b^2}{\gamma s_b} \right)^{1/(\gamma-1)},$$

and the pressure is given by

$$p_b = \frac{\rho_b c_b^2}{\gamma}.$$

Appendix B

Grid Generation Utilities

Grid generation is an important step in the process of numerically simulating the flow. Therefore, a poor-quality grid will certainly delay the convergence and reduce the accuracy of the solution. Several mesh formats are used in grid generation for CFD, such as: CGNS, FUN3D, OpenFOAM, SU2 or UGRID. The UGRID is a NASA format for meshes that is being used by several CFD solvers and grid generation software, *e.g.*, Pointwise.

This chapter explains how to convert a mesh file in UGRID format into a format usable by `uns3d` . The first section will give an overview of `prep` code which converts a mesh in the UGRID format to the two files that needed for the `uns3d` program. The second section will explain the use of `splitmesh` code, the domain decomposition program for parallel processing. For meshes that exceed tens of millions of grid nodes, it is profitable to split the UGRID before generating the `uns3d` grid files. This approach will be described in the third section of this chapter.

B.1 Grid Preprocessing (`prep`)

`prep` , the preprocessing program for `uns3d` , converts a mesh in UGRID format into a mesh file for use by the sequential version of `uns3d` . In conjunction with `splitmesh` , it can be used to create a mesh for the parallel version of `uns3d` .

B.1.1 Installation

It is convenient to define first the folder where software is being installed. For the C-shell add the following line to the `.cshrc` file:

```
setenv GGENHOME /wherever_you_want_it_to_be
```

For the Bourne, Bash or Korn shell, add the following line to the `.bashrc` file:

```
export GGENHOME=/wherever_you_want_it_to_be
```

To install the preprocessor, follow these steps:

1. Copy the distribution file `uns3dprep.tgz` from where you downloaded to `$GGENHOME` and untar it:

   ```
   cp uns3dpod.tgz $GGENHOME

   cd $GGENHOME

   tar xvfz uns3dprep.tgz
   ```

2. Edit the makefile to match the FORTRAN compiler that you have available and then make the makefile to generate the executable `prep`:

   ```
   make
   ```

3. For convenience, you might want to add the executable `prep` to a folder that is in the `$PATH`, for example `/usr/local/bin`. To do this, copy the executable `prep`:
   ```
   sudo cp prep /usr/local/bin
   ```
   The `sudo` command requires the superuser password for your computer.

B.1.2 Usage

Once the executable is installed, `prep` needs two things to generate a sequential mesh:

1. UGRID style mesh
2. `parameter.nml`

The boundary conditions used for the UGRID mesh are as follows:

Table B.1: List of boundary conditions.

0	Periodic slave
1	Inlet
2	Outlet
3	Wall
4	Rotating wall
7	Symmetry
9	Farfield
100	Periodic master

The file `parameter.nml` has the following format:

```
&global
echo       =    .true.    ! echo output from prep at run time
dvol       =    .false.   ! not used in prep, used in parallel_vol
spring     =    .false.   ! not used in prep
```

```
edgegen    =    .false.   ! not used in prep, used in parallel_vol
dump_prep  =    .true.    ! output a Tecplot file of the processed
                          ! .ugrid file
dump_debug =    .false.   ! output connectivity information for
                          ! debugging purposes
/

&filename
filemesh   =   "vol.mesh"   ! Name for .mesh file
fileplt    =   "prep.plt"   ! Name for Tecplot file if dump_prep is true
fileugrid  =   "vol.ugrid"  ! Name used for input .ugrid file
/

&preplist
pointwise  =    .true.      ! Flag for using pointwise .ugrid
                            ! files, which use positive boundary
    ! condition flags
ptol1      =    1d-5        ! Tolerance for translational
                            ! and rotational periodicity
                            ! (x position)
ptol2      =    1d-5        ! Tolerance for rotational periodicity,
                            ! radial position
radial     =    .false.     ! Flag for radial periodicity
angle      =    45.0d0      ! Angle between master/slave faces,
                            ! given in degrees
dx         =     0.0d0      ! x-coordinate difference between
                            ! master/slave face, translational
                            ! periodicity
dy         =     0.0d0      ! y-coordinate difference between
                            ! master/slave face, translational
                            ! periodicity
dz         =     0.0d0      ! z-coordinate difference between
                            ! master/slave face, translational
                            ! periodicity
/
```

Running `prep` with this input will read the file `vol.ugrid` and output a file `vol.mesh`. The mesh can be viewed in Tecplot with the output `prep.plt`.

Note that using the boundary conditions in this guide requires that `pointwise` be set to `.true.`.

To run `prep`, simply run the executable while in the folder that contains the `parameter.nml` file.

B.2 Domain Decomposition (splitmesh)

The domain decomposition program `splitmesh` must be used if the parallel `uns3d` executable is used. `splitmesh` is intended to convert a sequential mesh into several parallel meshes. The mesh is split into regions with user-defined criteria. These regions are then made into individual meshes for the processors. Several changes are made to the format of the split mesh, even if the case is to be run with only a single processor.

B.2.1 Installation

Installation proceeds much as it does for `uns3d` and `prep`.

It is convenient to define first the folder where software is being installed. For the C-shell add the following line to the `.cshrc` file:

```
setenv GGENHOME /wherever_you_want_it_to_be
```

For the Bourne, Bash or Korn shell, add the following line to the `.bashrc` file:

```
export GGENHOME=/wherever_you_want_it_to_be
```

To install the preprocessor, follow these steps:

1. Copy the distribution file `splitmesh.tgz` from where you downloaded to `$GGENHOME` and untar it:

   ```
   cp splitmesh.tgz $GGENHOME

   cd $GGENHOME

   tar xvfz splitmesh.tgz
   ```

2. Edit the makefile to match the FORTRAN compiler that you have available and then make the makefile to generate the executable `splitmesh`:

   ```
   make
   ```

3. For convenience, you might want to add the executable `splitmesh` to a folder that is in the `$PATH`, for example `/usr/local/bin`. To do this, copy the executable `prep`:
   `sudo cp prep /usr/local/bin`
   The `sudo` command requires the superuser password for your computer.

B.2.2 Input and Output Files

Input

`vol.mesh` - Sequential mesh file for `uns3d`. Sequential mesh file in vol.mesh format (converted from UGRID by `prep`)

`c2n.def` - Sequential cell-to-node connectivity file for `uns3d`. Sequential cell-to-node connectivity file (converted from UGRID by `prep`)

`LinearCascadeFV.def` <optional> - Additional data file for forced vibration of linear cascades. It will only be split if present.

SurfaceSetInt.def <optional> - Pointer file that contains boundary information for various surfaces. It will only be split if present.

Output

vol.mesh_??? : uns3d parallel mesh files. Each file contains a portion of the grid after domain decomposition

c2n.def_??? : uns3d parallel cell-to-node connectivity file. Each file contains a portion of the grid after domain decomposition

loc2glob_??? : Local node to global node pointer list. Its format is described on page 76

*.plt : Meshes in Tecplot format for viewing

LinearCascadeFV.def_??? : <optional> - It will only be split if present.

SurfaceSetInt.def_??? : <optional> - It will only be split if present.

B.2.3 Usage

Once the executable is installed, splitmesh needs only the sequential output from prep, typically named vol.mesh and c2n.def.

If splitmesh is executed without arguments, the following error message will be shown:

```
---------------------------------------------------------------
|  The correct usage of this executable is as follows:        |
|   splitmesh [meshfile] [c2nfile] [-nx #] [-ny #] [-nz #]    |
|        [-nr #] [-nz #] [-na #] [-nb #] [-nc #]              |
|                                                             |
|  -nx # -- specifies the number of splits in the x direction |
|  -ny # -- specifies the number of splits in the y direction |
|  -nz # -- specifies the number of splits in the z direction |
|                                                             |
|  -nr # -- specifies the number of splits along the radius   |
|  -nt # -- specifies the number of splits along theta        |
|                                                             |
|  -na # -- number of splits (user defined 1 in usr_def_srt)  |
|  -nb # -- number of splits (user defined 2 in usr_def_srt)  |
|  -nc # -- number of splits (user defined 3 in usr_def_srt)  |
|                                                             |
|  If no flags are specified, three values are read directly. |
|  This assumes a Cartesian coordinate system, in x, y, z     |
|    order.                                                   |
---------------------------------------------------------------
```

An example of correct usage would be the following:

```
splitmesh vol.mesh c2n.def -nx 2
```

This would split the mesh files `vol.mesh` and `c2n.def` into two parallel meshes, split with a plane of constant x with an equal number of nodes on either side (to the nearest integer).

Here is another example, which highlights the usage of multiple splits:

```
splitmesh vol.mesh c2n.def -nr 2 -nx 4
```

This would split the sequential mesh into eight parallel sections. This is done in two steps. First, the sequential mesh is split based on the radius from x-axis (the x-axis is the axis of rotation for rotational cases). Then, both of those meshes are split an additional 4 times by planes of constant x. Note that the planes may be different for each of the two meshes.

B.2.4 `usr_def_srt` Subroutine

The subroutine `usr_def_srt` is intended to be modified by the users who are not fully satisfied by the splitting options available. With it, any split can be obtained that is based on the node number or on the geometric location of the nodes. An ideal split is one that has the least number of nodes shared by multiple processors. This subroutine allows the user to modify the sorting algorithm to allow splits in different ways. Nodal positions and indices are provided within the subroutine.

The subroutine is given below:

```
subroutine  usr_def_srt(xout, which, x, y, z, nnode)

  implicit none

  integer, intent(in) :: nnode
  character, intent(in) :: which
  real(8), intent(in) :: x(nnode)
  real(8), intent(in) :: y(nnode)
  real(8), intent(in) :: z(nnode)

  real(8), intent(out) :: xout(nnode)
  integer :: i
  real(8) :: tmp

  ! ----------------------------------------------------------------------
  select case (which)
  case ('a')
     xout = x
  case ('b')
     do i = 1, nnode
        tmp = z(i) - 0.5d0 * x(i)
        xout(i) = tmp*tmp - y(i)*y(i)
     end do
  case ('c')
     xout = z
```

```
    end select

  end subroutine usr_def_srt
```

Each case corresponds to one of the three flags -na, -nb, or -nc.

As before, the ordering of these flags determines the order used to split the mesh.

The additional sorting options can be accessed using `-na`, `-nb`, and `-nc`.

NOTE: If the grid is single-block structured, `splitmesh` will automatically assume splitting is desired along i, j, and k. Splitting based on node position will not be possible. In almost all cases, this will lead to optimal splits.

B.2.5 Compatibility

Not all of the options are compatible with one another. There are three possible sets of coordinates:

Cartesian : This uses only the flags `-nx, -ny`, and `-nz`.

Cylindrical : This uses only the flags `-nx, -nr`, and `-nt`.

User-defined : This uses only the flags `-na, -nb`, and `-nc`.

If desired, coordinates can be combined based on the user defined functionality.

B.3 File Format

B.3.1 UGRID Format

The UGRID format is given below.

Header

```
nnode, nbtrias, nbquads, nctets, ncpent5, ncpent6, nchexs
```

For each node:

```
xnd(i), ynd(i), znd(i)
```

For each triangular boundary face:

```
n_face(i, 1) n_face(i, 2) n_face(i, 3)
```

For each quadrilateral boundary face:

```
n_face(i, 1) n_face(i, 2) n_face(i, 3) n_face(i, 4)
```

For each triangular boundary face:

```
bcond(i)
```

For each quadrilateral boundary face:

```
bcond(i)
```

For each tetrahedral cell:

```
n_cell(i, 1) n_cell(i, 2) n_cell(i, 3) n_cell(i, 4)
```

For each pyramidal pentahedral cell:

```
n_cell(i, 1) n_cell(i, 2) n_cell(i, 3) n_cell(i, 4) n_cell(i, 5)
```

For each prismatic pentahedral cell:

```
n_cell(i, 1) n_cell(i, 2) n_cell(i, 3) n_cell(i, 4) n_cell(i, 5) n_cell(i, 6)
```

For each hexahedral cell:

```
n_cell(i, 1) n_cell(i, 2) n_cell(i, 3) n_cell(i, 4)
n_cell(i, 5) n_cell(i, 6) n_cell(i, 7) n_cell(i, 8)
```

The definitions of the variables used in UGRID file are:

`nnode`	I	Number of nodes
`nbtrias`	I	Number of triangular boundary faces
`nbquads`	I	Number of quadrilateral boundary faces
`nctets`	I	Number of tetrahedral cells
`ncpent5`	I	Number of pyramidal pentahedral cells
`ncpent6`	I	Number of prismatic pentahedral cells
`nchexs`	I	Number of hexahedral cells
`xnd(i)`	I	x-location of node i
`ynd(i)`	I	y-location of node i
`znd(i)`	I	z-location of node i
`n_face(i, n)`	I	nth node of face i
`bcond(i)`	I	boundary condition for boundary face i
`n_cell(i, n)`	I	nth node of cell i

B.3.2 `loc2glob.dat` Format

Header:

nread nnode nnode_proc_max nproc

Data:

nloc ncpu nglob (<nread> lines)

The definitions of the variables used in `loc2glob.dat` file are:

`nread`	I	number of lines in file (sum of nodes in each parallel mesh)
`nnode`	I	number of nodes in sequential mesh
`nnode_proc_max`	I	maximum number of nodes on any one processor
`nproc`	I	number of parallel meshes
`nloc`	I	local node number (1–no more than nnode_proc_max)
`ncpu`	I	domain identity (0–nproc-1)
`nglob`	I	global node number (1–nnode)

B.4 splitout and combineout

The `uns3d` code writes its outputs based off of processor number when run in parallel. Sometimes the user may wish to combine these files into one. Or, the user may wish to split a single file into multiple files based off of processor number. For these cases, the `split_out` and `combineout` codes can be used.

Both codes are contained in the `split_out` directory. The codes are compiled using gfortran. If another compiler is desired, the user will need to modify the makefile within the directory. To compile the codes type

```
make
```

in the directory. The makefile is designed to compile both codes.

B.4.1 splitout

The `splitout` code takes a single file and splits it to be read by multiple processors based off of a `loc2glob` file. When executing the `splitout` code without any command line input arguments, the `splitout` code will output the following:

```
------------------------------------------------------------
|  The correct usage of this executable is as follows:     |
|     splitout <filename> <filetype> <l2gfile>,            |
|                                                          |
|  <filename> -- File to be split                          |
|  <filetype> -- Can be "out", "def", "tur", or "#",       |
|             where "#" refers to the number of values/node |
|  <l2gfile>  -- Output file from splitmesh                |
------------------------------------------------------------
```

This lets the user know how to run the code. There are three inputs: `filename`, `filetype`, and `l2gfile`.

`filename` - The name of the file to be split (without the . extension)

`filetype` - The type of file: ”out”, ”def”, ”tur”, or ”#”, where ”#” refers to the number of values/node. This code does not work on snapshot files. The `filetype` refers to the . extension of the `filename`.

`l2gfile` - The `loc2glob` file read in by the `uns3d` code.

B.4.2 combineout

The `combineout` code takes multiple files split by processor number and combines them into a single file using a provided `loc2glob` file. When executing the `combineout` code without any command line input arguments, the `combineout` code will output the following:

```
------------------------------------------------------------
|  The correct usage of this executable is as follows:     |
|   combineout <filename> <filetype> <l2gfile> [<c2nfile>],|
|                                                          |
|  <filetype> -- Can be "out", "def", "tur", "plt",        |
|            "snaps", or "#", where "#" refers to the      |
|            number of values/node                         |
|  <filename> -- Base name of files to be combined         |
|  <l2gfile>  -- Output file from splitmesh                |
|  <c2nfile>  -- Name of c2n connectivity (plt only)       |
|  <niter>    -- Number of iterations (snaps only)         |
------------------------------------------------------------
```

This lets the user know how to run the code. There are three to five inputs depending on the `filetype`: `filename`, `filetype`, `l2gfile`, `c2nfile`, and `niter`.

`filename` - The name of the file to be split (without the . extension)

`filetype` - The type of file: ”out”, ”def”, ”tur”, ”plt”, ”snaps”, or ”#”, where ”#” refers to the number of values/node. For all except ”snaps”, the `filetype` refers to the . extension of the `filename`. The ”snaps” input is for combining snapshot files.

`l2gfile` - The `loc2glob` file read in by the `uns3d` code.

`c2nfile` - The connectivity file `c2n.dex` read in by the `uns3d` code.

`niter` - The number of snapshots.

Bibliography

[1] S. R. Allmaras, F. T. Johnson, and P. R. Spalart. Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model. In *Seventh International Conference on Computational Fluid Dynamics*, number ICCFD7-1902, Big Island, HI, July 2012.

[2] M. F. Barone and J. L. Payne. Methods for simulation-based analysis of fluid–structure interaction. Technical Report SAND2005-6573, Sandia National Laboratories, Albuquerque, NM, October 2005.

[3] P. G. A. Cizmas, J. I. Gargoloff, T. W. Strganac, and P. S. Beran. A parallel multigrid algorithm for aeroelasticity simulations. *Journal of Aircraft*, 47(1):53–63, January–February 2010.

[4] A. de Boer, M. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85(11–14):784–795, June–July 2007.

[5] J. Gressier and J.-M. Moschetta. On the pathological behavior of upwind schemes. 36th AIAA Aerospace Sciences Meeting & Exhibit, AIAA Paper 98-0110, Reno, NV, January 1998.

[6] A. Harten. Self adjusting grid methods for one dimensional hyperbolic conservation laws. *Journal of Computational Physics*, 50:235–269, 1983.

[7] S. Kim, C. Kim, O. Rho, and S. Hong. Cure for shock instability: Development of an improved roe scheme. 40th AIAA Aerospace Sciences Meeting & Exhibit, AIAA Paper 2002-0548, Reno, NV, January 2002.

[8] F. R. Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA Journal*, 32(8):1598–1605, August 1994.

[9] J. Ousterhout and K. Jones. *Tcl and the Tk Toolkit*. Pearson Education, 2009.

[10] P. Roe. Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[11] P. Roe. Characteristic Based Schemes for the Euler Equations. *Annual Review of Fluid Mechanics*, 18:337–365, 1986.

[12] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th Aerospace Sciences Meeting & Exhibit*, AIAA-92-0439, Reno, NV, January 1992.

[13] J. Trépanier, M. Reggio, H. Zhang, and R. Camarero. A finite-volume method for the Euler equations on arbitrary Lagrangian–Eulerian grids. *Computers & Fluids*, 20(4):399–409, 1991.

[14] M. Vinokur. An analysis of finite-difference and finite-volume formulations of conservation laws. *Journal of Computational Physics*, 81(1):1–52, March 1989.

[15] B. Welch, K. Jones, and J. Hobbs. *Practical Programming in Tcl and Tk.* Practical Programming in Tcl/Tk. Prentice Hall PTR, 2003.

[16] K. Xu. Does perfect Riemann solver exist? 14th AIAA Computational Fluid Dynamics Conference, AIAA Paper 99-3344, Norfolk, VA, June 1999.