RocfluCM Users Guide
Version 1.4.0

March 13, 2014

www.illinoisrocstar.com

License

RocfluCM sources, executables, and this document are the property of Illinois Rocstar LLC. Licensing and support of the software package, including full source access for government, industrial, and academic partners, are arranged on an individual basis. Please contact Illinois Rocstar at

- **tech@illinoisrocstar.com**
- **sales@illinoisrocstar.com**

for support and licensing.

Contents

1 Introduction

1.1 Objective

The objective of this user's guide is three-fold:

1. To enable users to compile and install the RocfluMP source code on computer systems.
2. To enable users to run the RocfluMP code for the example cases.
3. To enable users to run the RocfluMP code for their own cases.

2 Overview of RocfluMP

solves the three-dimensional time-dependent compressible Navier-Stokes equations on moving and/or deforming unstructured grids. The grids may consist of arbitrary combinations of tetrahedra, hexahedra, prisms, and pyramids. The spatial discretization is carried out using the finite-volume method. The inviscid fluxes are approximated by upwind schemes to allow for capturing of shock waves and contact discontinuities. Steady flows can be computed using an explicit multi-stage method tuned for fast convergence. Unsteady flows are computed with the fourth-order accurate Runge-Kutta method.

To solve fluid-dynamics problems in which processes other than those described by the Navier-Stokes equations are important, RocfluMP is designed to interface with so-called *multi-physics modules*. The multi-physics modules model phenomena such as turbulence, particles, chemical reactions, and radiation and their interaction. At present, the multi-physics modules are under development and have not yet been integrated with RocfluMP. When considering fluid flow problems with several chemical species, RocfluMP may be regarded as solving transport equations for the mixture variables.

RocfluMP may be used to solve problems involving fluid-structure interactions. More specifically, RocfluMP is designed to operate as a solution module inside CSAR's coupled rocket-simulation code GENx. To accommodate dynamically changing fluid domains arising from the deformation predicted by a structural simulation, RocfluMP allows for moving grids. The Geometric Conservation Law (GCL) is satisfied in a discrete sense to machine-precision to avoid the introduction of spurious sources of mass, momentum, and energy due to grid motion.

The relationship of RocfluMP and the other codes is depicted schematically in Fig. 1. A brief description of the multi-physics modules follows (they are described in detail in their respective manuals).

- Rocturb is the turbulence module which implements a variety of models for Reynolds-averaged Navier-Stokes (RANS) simulations and Large-Eddy Simulation (LES).
- Rocpart is the Lagrangian particle tracking module.

- Rocspecies is the module simulating the evolution of chemical species and Equilibrium Eulerian particles.
- Rocrad is the radiation module.

RocfluMP consists of several modules:

- rfluconv is the conversion module of RocfluMP. It converts RocfluMP solution and grid files from ASCII to binary format and vice versa, and converts RocfluMP grid files into a format supported by YAMS and TETMESH. rfluconv requires interactive user input.
- rfluinit is the initialization module of RocfluMP. It generates RocfluMP solution files for each region based on the information contained in the user input file.
- rflumap is the processor-mapping module of RocfluMP. It generates the mapping file which is required for parallel computations. It also generates the Rocin control files for computations with GENx. rflumap requires interactive user input.
- rflump is the actual solution module of RocfluMP.
- rflupick is used in conjunction with rflupost to visualize only specific cells in the grid, such as cells near boundaries or cells sharing faces or vertices. For parallel computations, rflupick can also be used to instruct rflupost to convert only specific regions for visualization. This allows the visualization of nominally large cases on small machines. rflupick requires interactive user input.
- rflupost is the postprocessing module of RocfluMP. It converts grid and solution files from the RocfluMP format into the formats recognized by visualization packages. At present, the following formats are supported:
  - TECPLOT format
- rflupart is the partitioning module of RocfluMP. It converts grid files from outside formats into binary or ASCII files in RocfluMP format and partitions the grids into regions. At present, the following formats are supported:
  - HYBRID format (CENTAUR grid generator, CentaurSoft, Austin, TX)
  - VGRIDns format (VGRIDns grid generator, Shahyar Pirzadeh, NASA Langley)
  - MESH3D format (MESH3D grid generator, Tim Baker, Princeton University)
  - TETMESH format (TETMESH grid generator, SIMULOG, France)
  - Cobalt format (Cobalt flow solver, Cobalt Solutions LLC, Springfield, OH)
  - GAMBIT format (GAMBIT grid generator, Fluent, Lebanon, NH)

2.1 Overview of User's Guide

The remainder of the user's guide consists of the following chapters:

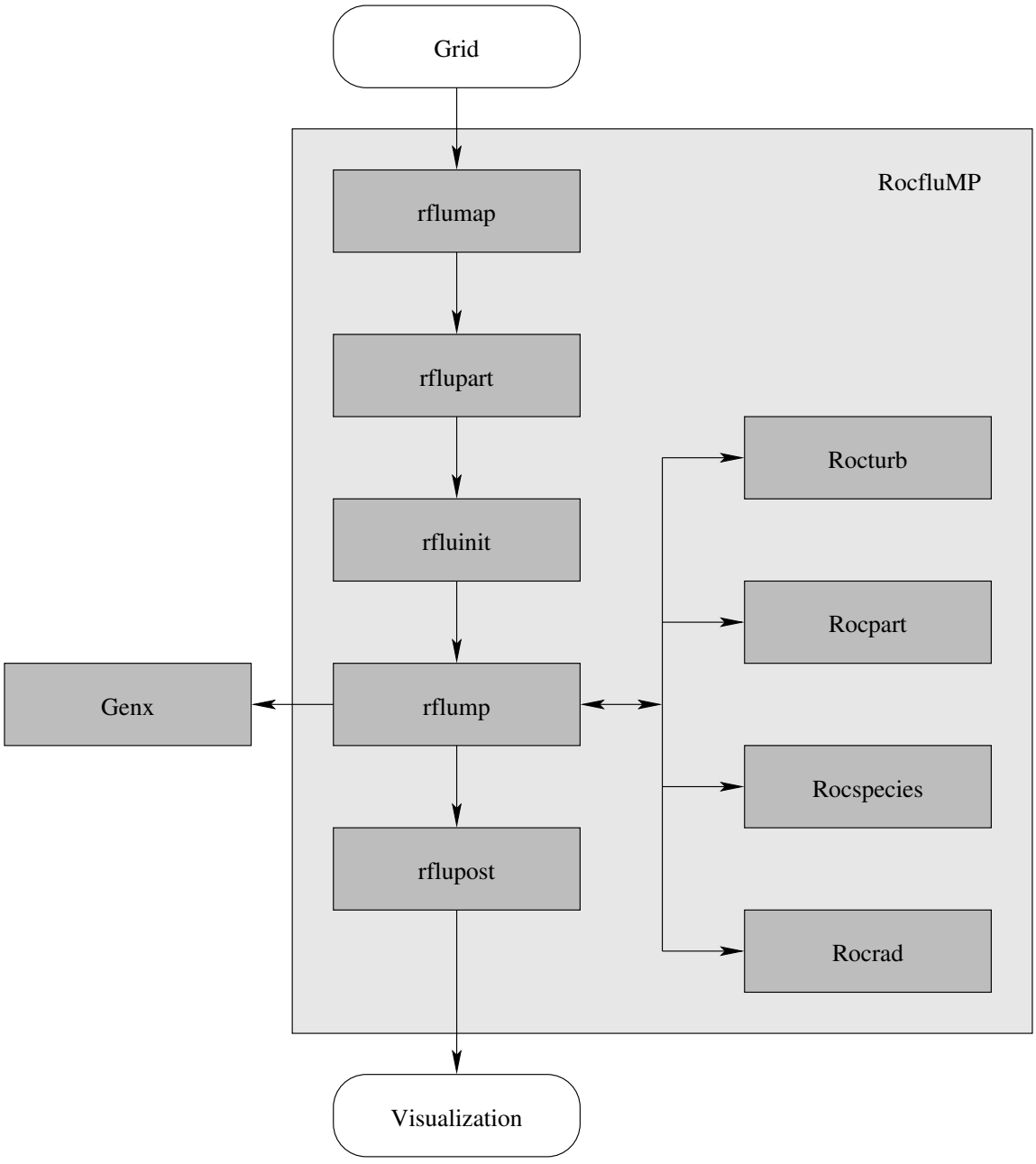


Figure 1: Overview of RocfluMP and related codes.

2.2 Related Documents

The information contained in this document is supplemented by the following documents:

- "RocfluMP Developer's Guide".

3 Installation and Compilation

3.1 Installation

The following assumes that RocfluMP is to be installed either from the CSAR CVS repository or from a `gzipped tar` file.

3.1.1 Installation from CVS Repository

o be able to access the CSAR CVS repository, set the `CVSROOT` environment variable to (taking the `bash` shell as an example)

```
export CVSROOT=:pserver:user@machine.uiuc.edu:/cvsroot
```

and either open a new terminal or type

```
[user@machine ~]$ source .bashrc
```

Then type

```
[user@machine ~]$ cvs login
```

and hit the `Enter` key at the prompt.

Now move into the directory where you want to install RocfluMP. In the following, this is assumed to be `directory`. Then type

```
[user@machine ~/directory]$ cvs co genx/Codes/RocfluidMP
```

which will check out the source code for RocfluMP from the repository.

Assuming the checkout command has completed successfully, you are now ready to compile the code for serial computations, and you can proceed to Sec. 3.2.

3.1.2 Installation from `.tar.gz` File

ove into the directory where you want to install RocfluMP. In the following, this is assumed to be `directory`. Move or copy the `gzipped tar` file, assumed to be `<file>.tar.gz` in the following, into `directory`. Then type

```
[user@machine ~/directory]$ gzip -d <file>.tar.gz
[user@machine ~/directory]$ tar -xvf <file>.tar
```

which will unpack the source code.

Assuming these commands to have completed successfully, you are now ready to compile the code for serial computations, and you can proceed to Sec. 3.2.

3.2 Compilation

3.2.1 Overview of Compilation Process

he compilation process for RocfluMP is automatic in the sense that the `Makefile`s determine the machine type and set the suitable compilation options. If you intend to run on Apple, IBM, Linux, SGI, or Sun machines, you do not need to modify any `Makefile`s. If you intend to run on other machines, you will need to create your own `Makefile`. You can pattern it after the existing machine-dependent `Makefile`s.

RocfluMP is compiled with MPI by default, which means that you must have installed MPI on your machine before attempting to compile RocfluMP.

The compilation process consists of two stages. The first stage is the actual computation, as described below. The output of the compilation process are several executables:

rfluconv The conversion module of RocfluMP.

rfluinit The initialization module of RocfluMP.

rflumap The region mapping module of RocfluMP.

rflupick The region and cell picking module of RocfluMP.

rflupost The postprocessing module of RocfluMP.

rflupart The partitioning module of RocfluMP.

rflump The flow solution module of RocfluMP.

The second stage consists of copying these executables into your `$(HOME)/bin` directory by typing

```
[user@machine ~/directory]$ gmake RFLU=1 install
```

3.2.2 Description of Compilation Options

o compile RocfluMP, type the following at the prompt:

```
[user@machine ~/directory]$ gmake RFLU=1 <options>
```

where the currently supported `<options>` are any of the following:

`CHECK_DATASTRUCT=1` Activates checking of data structures. This option will print out the content of the important data structures used by RocfluMP. Note that activating this option will lead to substantial screen output, so it should only be activated for small cases.

`DEBUG=1` Activates debugging compiler options. If this option is not specified, optimizing compiler options are chosen by default.

`PLAG=1` Activates compilation of Rocpart. This option must be specified if you wish to run computations with Lagrangian particles.

`SPEC=1` Activates compilation of Rocspecies. This option must be specified if you wish to run computations with chemical species and/or Equilibrium Eulerian particles.

4 Execution

This chapter contains detailed information on the command-line arguments and input and output files of rfluconv, rfluinit, rflumap, rflump, rflupart, rflupick, and rflupost.

4.1 rfluconv

4.1.1 Invocation

rfluconv is invoked by typing

```
rfluconv -c <casename> -s <stamp> -v <verbosity>
```

The following command-line arguments are read by rfluconv:

<casename> A character string used to label the input and output files.

<stamp> A variable indicating the iteration or time from which the grid and solution files are to be read.

<verbosity> An integer indicating the desired verbosity level of rfluconv. The verbosity level can take the following values:

- 0 No output. rfluconv will not write any information to standard output.
- 1 Low level of output. rfluconv will write some information to standard output.
- 2 High level of output. rfluconv will write detailed information to standard output.

rfluconv expects interactive user input after invokation, as described in Sect. 4.1.4.

4.1.2 Input Files

The following input files are read by rfluconv:

- A grid file in RocfluMP ASCII or binary format.
- A flow solution file in RocfluMP ASCII or binary format.
- A dimension file.
- A boundary condition file.

4.1.3 Output Files

The following output files are written by rfluconv:

- A grid file in RocfluMP ASCII or binary format.
- A flow solution file in RocfluMP ASCII or binary format.
- A surface-grid file for TETMESH or YAMS.
- A version file called `rfluconv.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

4.1.4 Interactive Input

The interactive input after invokation is self-explanatory and will not be described here.

4.2 rfluinit

4.2.1 Invocation

rfluinit is invoked by typing

```
rfluinit -c <casename> -v <verbosity>
```

The command-line arguments read by rfluinit are:

`<casename>` A character string used to label the input and output files.

`<verbosity>` An integer indicating the desired verbosity level of rfluinit. The verbosity level can take the following values:

- `0` No output. rfluinit will not write any information to standard output.
- `1` Low level of output. rfluinit will write some information to standard output.
- `2` High level of output. rfluinit will write detailed information to standard output.

4.2.2 Input Files

The following input files are read by rfluinit:

- An input file called `<casename>.inp`.
- A grid file in RocfluMP format.

- A boundary-condition file. The name of the file is `<casename>.bc`.
- A dimension file.
- A restart-information file.

4.2.3 Output Files

The following input files are written by rfluinit:

- A flow solution file in rfluinit format.
- A version file called `rfluinit.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

4.3 rflumap

4.3.1 Invocation

rflumap is invoked by typing:

```
rflumap -c <casename> -m <mode> -p <nprocs> -r <nregions> -v <verbosity>
```

The following command-line arguments are read by rflumap:

`<casename>` A character string used to label the input and output files.

`<mode>` An integer indicating the mode of invokation. The mode can take the following values:

1 Initial mode. rflumap will only create a mapping file.

2 Final mode. rflumap will read an existing mapping file and create an input file for Rocin.

`<nprocs>` An integer indicating the number of processes. The number of processes must be less or equal to the number of regions.

`<nregions>` An integer indicating the number of regions. The number of regions must be greater or equal to the number of processes.

`<verbosity>` An integer indicating the desired verbosity level of rflumap. The verbosity level can take the following values:

0 No output. rflumap will not write any information to standard output.

1 Low level of output. rflumap will write some information to standard output.

2 High level of output. rflumap will write detailed information to standard output.

4.3.2 Input Files

rflumap does not read any input files.

4.3.3 Output Files

rflumap writes the region mapping file. For coupled runs, rflumap also produces the input file for Rocin.

4.4 rflump

4.4.1 Invocation

rflump is an MPI code and hence it's invokation is dependent on the type of machine and may also be dependent on the MPI distribution. For typical MPI distributions, rflump is invoked by typing

```
mpirun -np <n> rflump -c <casename> -v <verbosity>
```

where `<n>` is the number of processes.

The command-line arguments read by rflump are:

`<casename>` A character string used to label the input and output files.

`<verbosity>` An integer indicating the desired verbosity level of rflump. The verbosity level can take the following values:

- `0` No output. rflump will not write any information to standard output.
- `1` Low level of output. rflump will write some information to standard output.
- `2` High level of output. rflump will write detailed information to standard output.

It is important to be aware of the automatic restart capability of rflump. When flow-solution files are written, rflump writes the iteration number or time stamp into the so-called restart-information file. Whenever rflump is invoked, it checks for the existence of the restart-information file. If the restart-information file exists, rflump reads the last iteration number or time stamp at which flow-solution files were written. rflump reads in the flow-solution files corresponding to that iteration number or time stamp and starts the computation. Therefore, successive invokations of the above commands lead to different behaviour of rflump! This is done to simplify the execution of rflump within batch jobs.

4.4.2 Input Files

The following input files are read by rflump:

- An input file called `<casename>.inp`.
- A grid file in RocfluMP format.
- A flow-solution file in RocfluMP format.
- A boundary-condition file. The name of the file is `<casename>.bc`.
- A dimension file.
- A restart-information file.

4.4.3 Output Files

The following input files are written by rflump:

- A grid file in RocfluMP format.
- A flow solution file in RocfluMP format.
- A dimension file if the flow is unsteady and grid motion is active.
- A restart-information file.
- A probe file.
- A convergence file.
- A mass-conservation check file if grid motion is active and rflump is not run within GENx.
- A version file called `rflump.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

4.5 rflupart

4.5.1 Invocation

For serial computations, rflupart is invoked by typing

```
rflupart -c <casename> -v <verbosity>
```

The command-line arguments read by rflupart are:

<casename> A character string used to label the input and output files.

<verbosity> An integer indicating the desired verbosity level of rflupart. The verbosity level can take the following values:

0 No output. rflupart will not write any information to standard output.

1 Low level of output. rflupart will write some information to standard output.

2 High level of output. rflupart will write detailed information to standard output.

4.5.2 Input Files

The following input files are read by rflupart:

- An input file called `<casename>.inp`.
- A grid file. rflupart supports the following formats:
  - CENTAUR format. The CENTAUR grid file may be in ASCII or binary format. The file in ASCII format is called `<casename>.hyb.asc`, and the file in binary format is called `<casename>.hyb.bin`.
  - VGRIDns format. The file is called `<casename>.cgosg`.
  - MESH3D format. The file is called `<casename>.m3d`.
  - TETMESH format. The file is called `<casename>.noboite`.
  - Cobalt format. The file is called `<casename>.cgr`.
  - GAMBIT format. The file is called `<casename>.neu`.
- A boundary condition file called `<casename>.bc`.
- A mapping file called `<casename>.map` specifying how many processors are to be used for parallel computations, and how the regions are mapped to the processors. This file is only required for parallel computations. If the mapping file does not exist, rflupart assumes that a serial computation will be made.
- A file specifying the mapping between the boundary patches used during grid generation and how these patches translate to the patches to be used in RocfluMP. This file is only needed if a VGRIDns, MESH3D, or Cobalt grid file is read. The file is called `<casename>.vgi`, `<casename>.mgi`, or `<casename>.cgi`, depending on whether the grid file is in VGRIDns, MESH3D, or Cobalt format.

4.5.3 Output Files

The following output files are written by rflupart:

- A grid file in RocfluMP format. For parallel computations, the number of grid files written out depends on the number of regions specified in the mapping file.
- A dimension file. For parallel computations, the number of dimension files written out depends on the number of regions specified in the mapping file.
- A version file called `rflupart.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

4.6 rflupick

When running rflump in parallel, the output from rflupick can only be used if rflupost is instructed not to merge regions.

4.6.1 Invocation

rflupick is invoked by typing:

```
rflupick -c <casename> -s <stamp> -v <verbosity>
```

The following command-line arguments are read by rflupick:

`<casename>` A character string used to label the input and output files.

`<stamp>` A variable indicating the iteration or time from which the grid and solution files are to be read.

`<verbosity>` An integer indicating the desired verbosity level of rflupick. The verbosity level can take the following values:

`0` No output. rflupick will not write any information to standard output.

`1` Low level of output. rflupick will write some information to standard output.

`2` High level of output. rflupick will write detailed information to standard output.

rflupick expects interactive user input after invokation, as described in Sect. 4.6.4.

4.6.2 Input Files

The following input files are read by rflupick:

- A grid file in RocfluMP ASCII or binary format.
- A flow solution file in RocfluMP ASCII or binary format.

- A dimension file.
- A boundary condition file.
- A mapping file.

4.6.3 Output Files

The following output files are written by rflupick:

- A postprocessor info file.
- A version file called `rflupick.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

4.6.4 Interactive Input

After invokation, rflupick presents the following screen output:

```
Picking regions manually...
  Enter information on regions:
    a - Pick all regions
    s - Pick some regions
    n - Pick no regions
  Enter information type:
```

The input determines which regions will be postprocessed by rflupost.

Picking all regions. Assuming that the user enters option a, rflupick will loop over all regions, and for each region present the following screen output (the following output assumes that the region index is 00001):

```
Picking special cells...
  Global region: 00001
    Enter information on special cells:
      b - cell adjacent to boundary face
      c - single cell
      f - cells adjacent to interior face
      s - stencil members
      v - cells meeting at vertex
      q - quit
  Enter information type:
```

This output is self-explanatory. Once special cells were picked, rflupick will present the following screen output:

```
Picking special faces...
  Global region: 00001
    Enter information on special faces:
      b - boundary face
      i - interior face
      q - quit
  Enter information type:
```

Once again, the output is self-explanatory. Once special faces were picked, rflupick will repeat the same procedure for the next region.

Picking some regions. Assuming that the user enters option `s`, rflupick will present the following screen output:

```
Enter global region index (< 1 to exit):
```

rflupick will repeat the same request until the user enters an integer smaller than unity. Once some regions were picked, rflupick will loop over the picked regions and present the same output as described above.

4.7 rflupost

4.7.1 Invocation

For serial computations, rflupost is invoked by typing

```
rflupost -c <casename> -s <stamp> -v <verbosity>
```

The following command-line arguments are read by rflupost:

<casename> A character string used to label the input and output files.

<stamp> A variable indicating the iteration or time from which the grid and solution files are to be read.

<verbosity> An integer indicating the desired verbosity level of rflupost. The verbosity level can take the following values:

- 0 No output. rflupost will not write any information to standard output.
- 1 Low level of output. rflupost will write some information to standard output.
- 2 High level of output. rflupost will write detailed information to standard output.

4.7.2 Input Files

The following input files are read by rflupost:

- A grid file in RocfluMP format.
- A flow solution file in RocfluMP format.
- A dimension file.
- A boundary condition file.
- A file generated by rflupick detailing which regions are to be postprocessed and whether individual cells are to be visualized. If this file does not exist, all regions are postprocessed.

4.7.3 Output Files

The following output files are written by rflupost:

- A file in binary TECPLOT format called `<casename>.plt`.
- A version file called `rflupost.vrs`. It contains the version number and date of the executable. Successive runs append to the version file.

5 Capability Descriptions

his chapter describes the capabilities of RocfluMP and points the user to the relevant sections in Chapter 6 of this manual the developer and reference manual.

5.1 Pressure, Skin-Friction, and Heat-Transfer Coefficient Computation

can compute pressure, skin-friction, and heat-transfer coefficients for faces on patches. Definitions of these coefficients can be found in the Section "Pressure, Skin-Friction, and Heat-Transfer Coefficient Computation" in the RocfluMP Developer and Reference Manual.

5.2 Force and Moment Computation

can compute forces and moments exerted by the fluid on the patches. Definitions of the force and moment coefficients and their computation can be found in the Section "Force and Moment Computation" in the RocfluMP Developer and Reference Manual.

The computation of forces and moments is governed by the `FORCES` Section in the input file.

6 File Format and Content Specifications

The files read and written by the various modules are described in this chapter with the exception of the visualization files written by rflupost, which are described in Section 9. Note that the majority of files share a user-specified string, the so-called 'case name,' represented by `casename` below.

6.1 Filename Conventions

any of the files whose format is described below consist of a region index and either an iteration or a time index. The region index indicates the global region number with which a given file is associated. A region index of zero indicates that the given file is associated with a serial or unpartitioned data set. A region index of one or greater indicates that the given file is part of parallel data set.

6.2 Input File

he input file is called `<casename>.inp`. The input file is divided into sections. Each section contains several lines, each of which consists of a keyword and a value, as shown below.

```
# SECTION_NAME
KEYWORD_1 VALUE_1
KEYWORD_2 VALUE_2
KEYWORD_3 VALUE_3
#
```

Comments may be inserted after the specification of the values; they are ignored by the routines reading the input file.

The following sections describe each section and the associated keywords in detail. For simplicity, the sections are listed in alphabetical order, but they may appear in any order in the input file.

6.2.1 `FORCES` Section

The `FORCES` section contains the following keywords:

`FLAG` Specifies whether forces and moments are to be computed. It can take the following values:

- `0` Do not compute forces and moments.
- `1` Compute forces and moments.

`REFLENGTH` Specifies value of reference length [m].

`REFAREA` Specifies value of reference area [m^2].

`REFXCOORD` Specifies value of reference x-coordinate [m].

`REFYCOORD` Specifies value of reference y-coordinate [m].

`REFZCOORD` Specifies value of reference z-coordinate [m].

6.2.2 `FORMATS` Section

he `FORMATS` section contains the following keywords:

`GRID` Specifies the format of the RocfluMP grid file. It can take the following values:

- `0` Grid file is in ASCII format.
- `1` Grid file is in binary format.

`SOLUTION` Specifies the format of the RocfluMP flow file. It can take the following values:

- `0` Flow file is in ASCII format.
- `1` Flow file is in binary format.

`GRIDSRC` Specifies the format of the grid file read by rfluprep. It can take the following values:

- `0` Grid file is in CENTAUR ASCII format.
- `1` Grid file is in VGRIDns format.
- `2` Grid file is in MESH3D format.
- `3` Grid file is in TETMESH format.
- `4` Grid file is in Cobalt format.
- `5` Grid file is in GAMBIT format. Only GAMBIT files in ASCII neutral file format are supported.
- `10` Grid file is in CENTAUR binary format.

6.2.3 `FLOWMODEL` Section

he `FLOWMODEL` section contains the following keywords:

`MODEL` Specifies which equations are to be solved. It can take the following values:

- `0` rflump solves the Euler equations.
- `1` rflump solves the Navier-Stokes equations.

`MOVEGRID` Specifies whether grid motion is active or not. It can take the following values:

`0` Grid motion is inactive.

`1` Grid motion is active.

It is important to note that this flag influences the names of the grid file and dimension file. If grid motion is active and the flow is unsteady, these files aquire a time stamp, see Sec. 6.3.1.

6.2.4 `GRIDMOTION` Section

he `GRIDMOTION` section contains the following keywords:

`TYPE` Specifies the type of grid motion. It can take the following values:

`1` Move the grid by smoothing the boundary displacements. This option is only available for serial computations.

`2` Move the grid by smoothing the coordinates. This option is only available for serial computations.

`3` Move the grid using the MESQUITE package. This option is only available when running RocfluMP in GENx.

Moving the grid based on smoothing boundary displacements has the advantage that the vertices are not moved for vanishing displacements. This is not true if the grid is moved by smoothing coordinates. In particular, non-uniform or distorted grids can be strongly affected by smoothing the coordinates. Therefore, moving the grid by smoothing the boundary displacements is the recommended option.

`NITER` Specifies the number of smoothing iterations. Only applicable if `TYPE=1` or `TYPE=2`.

`SFACT` Specifies the smoothing coefficient. The values for the number of smoothing iterations and the smoothing coefficient should be chosen together. The recommended values are, for moving the grid by smoothing the boundary displacements, `NITER=4` and `SFACT=0.25`, and for moving the grid by smoothing the coordinates, `NITER=10` and `SFACT=0.1`. Only applicable if `TYPE=1` or `TYPE=2`.

6.2.5 `INITFLOW` Section

The `INITFLOW` section is relevant only to rfluprep. It contains the following keywords:

`FLAG` Specifies whether initial solution is to be generated. It can take the following values:

`1` Generate initial solution using the values assigned to the keywords `DENS`, `VELX`, `VELY`, `VELZ`, and `PRESS`.

2 Generate initial solution using the data contained in a file.

3 Generate initial solution using a hardcode. The hardcode depends on the `casename`. This option can only be used if appropriate code to initialize the solution was added to the file `RFLU_InitFlowHardCode.F90` in `rfluprep` for the given `casename`. If the appropriate code does not exist, rfluprep will return an error. At present, hardcoded initial solutions for the following casenames are provided:

`onera_c0` ONERA C0 case [3].

`ringleb` Ringleb flow [4].

`st_sod1` Sod shock tube [6], case 1.

`st_sod2` Sod shock tube [6], case 2.

`ssvort<t><mxn>l<p>` Supersonic free vortex.

`<t>=[h|p]` `h` and `p` denote hexahedral and prismatic grids, respectively.

`<mxn>` denotes the grid resolution for both hexahedral and prismatic grids. It can take the following values: `20x5`, `40x10`, `80x20`, `160x40`, `320x80`.

`<p>` denotes the number of layers in the z-coordinate direction. It can take values `1` or `3`.

`DENS` The density of the initial solution, [kg/m^3]. Only read if `FLAG=1`.

`VELX` The x-component of the velocity vector of the initial solution, [m/s]. Only read if `FLAG=1`.

`VELY` The y-component of the velocity vector of the initial solution, [m/s]. Only read if `FLAG=1`.

`VELZ` The z-component of the velocity vector of the initial solution, [m/s]. Only read if `FLAG=1`.

`PRESS` The static pressure of the initial solution, [Pa]. Only read if `FLAG=1`.

`IVAL1` Integer variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`IVAL2` Integer variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`IVAL3` Integer variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`IVAL4` Integer variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`IVAL5` Integer variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`IVAL6` Integer variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`RVAL1` Real variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`RVAL2` Real variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`RVAL3` Real variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`RVAL4` Real variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`RVAL5` Real variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

`RVAL6` Real variable used to set hard-coded initial conditions. Only relevant if `FLAG=3`.

6.2.6 `NUMERICS` Section

The `NUMERICS` section contains the following keywords:

`CFL` Specifies the CFL number to be used during computations.

`DISCR` Specifies the discretization scheme for the inviscid fluxes. It can take the following values:

- `1` The flux-difference splitting of Roe [5].
- `3` The HLLC approximate Riemann solver of Batten et al. [2]

`DISSFACT` Factor multiplying the dissipation terms of the flux-difference splitting of Roe. The dissipation may need to be reduced to capture marginally resolved flow features such as separation or vortices. It is important to note however, that reducing the dissipation can compromise the stability of the computation. The default value of `DISSFACT` is `1`.

`ORDER` The order of accuracy of the flux discretization. It can take the following values:

- `1` Compute fluxes with first-order accuracy.
- `2` Compute fluxes with second-order accuracy.

`ENTROPY` Specifies value of entropy correction coefficient.

`DIMENS` Specifies the dimensionality of the computation. It can take the following values:

- `2` Run two-dimensional computation.
- `3` Run three-dimensional computation (default).

The dimensionality flag is provided so that truly two-dimensional computations can be performed, i.e., a computation in which the grid contains only one cell in the z-direction. Three-dimensional computations must contain at least three cells in all directions to be able to compute gradients. If you wish to run truly two-dimensional computations, there must be two boundary patches which coincide with $z =$ constant planes, and you should specify the boundary condition on those patches to be `BC_VIRTUAL`, see Sect. 6.5.1.8.

6.2.7 `POST` Section

The `POST` section is relevant only to rflupost(with the exception of the keyword `COORDFLAG`). It contains the following keywords:

`PLTTYPE` Specifies the data to be written to output file. It can take the following values:

- `1` Write only the grid to the output file.
- `2` Write the grid and the solution to output file.

`MERGEFLAG` Specifies whether the regions from a parallel computation are to be merged for postprocessing. It can take the following values:

- `0` Do not merge the regions for postprocessing. Each partition will be written separately to the output file. Virtual cells and boundary faces will be written separately for each region and patch, respectively.
- `1` Merge the regions for postprocessing.

Not merging regions for postprocessing can be useful for several reasons: First, one may want to make sure that solution contours are well-behaved across partition boundaries. Second, when using this option in conjunction with rflupick, the amount of data to be visualized can be reduced drastically by selecting only specific regions to be postprocessed in rflupost. Third, it becomes possible to visualize virtual cells and boundary faces which can be useful during debugging.

`PLTVOLFLAG` Specifies whether volume data are to be written to output file. It can take the following values:

- `0` Do not write volume data for postprocessing; only surface data is written.
- `1` Write volume and surface data for postprocessing.

The advantage of not writing volume data to the output file is to reduce the amount of data to be visualized.

`INTERTYPE` Specifies whether and how data is to be interpolated from the cell-centers to the vertices. It can take the following values:

- `0` Do not transfer the solution from cell centers to vertices. This option avoids the introduction of interpolation errors and can be useful if it is necessary to visualize small solution differences which may be smoothed out by the interpolation process. TECPLOT allows the visualization of cell-centered solution data.
- `1` Use simple arithmetic averaging.
- `2` Use k-exact averaging. This method interpolates polynomials of order k exactly on arbitrary grids. The order k is specified by the value assigned to the keyword `INTERORDER`.

Choosing between `INTERTYPE=1` and `INTERTYPE=2` is largely a matter of balancing the accuracy and cost of the interpolation. Simple arithmetic averaging is much faster but also less accurate than the k-exact interpolation method. In practice, the differences between the two methods are usually negligible unless solution gradients are very large or grids are highly distorted.

`INTERORDER` Specifies the polynomial order of interpolation. This keyword is only relevant if `INTERTYPE=2`. It can take the following values:

- `1` Use linear interpolation.
- `2` Use quadratic interpolation.

`SPECFLAG` Specifies whether the postprocessor-information file produced by rflupick is to be read. It can take the following values:

- `0` Do not read the postprocessor-information file.
- `1` Read the postprocessor-information file.

Reading the postprocessor-information file allows only specific regions to be postprocessed. See also `MERGEFLAG` and `COORDFLAG`.

`COORDFLAG` Specifies which regions are to be written to the output file. This option is only used by rflupick. It can take the following values:

- `0` Do not select regions for postprocessing based on the coordinates of their bounding box.
- `1` Select regions for postprocessing based on the coordinates of their bounding box. Only regions which lie within the bounding box specified by the values assigned to the keywords `XCOORDLOW`, `XCOORDUPP`, `YCOORDLOW`, `YCOORDUPP`, and `ZCOORDLOW`, and `ZCOORDUPP` will be selected for postprocessing.

If you wish to make use of the capabilities provided by this keyword, you need to execute rflupick before executing rflupost.

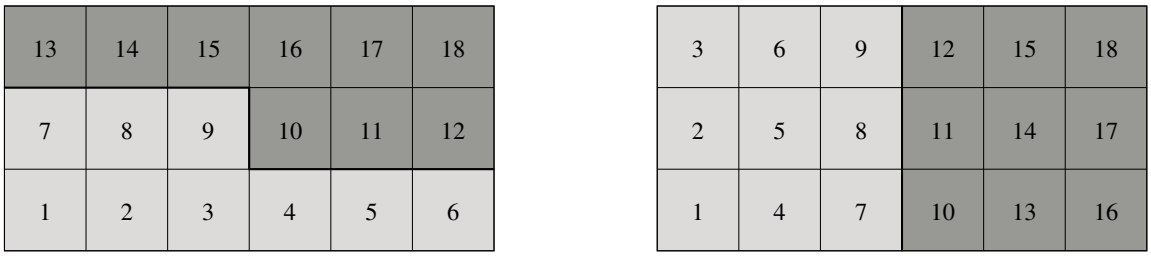
`XCOORDLOW` Specifies the lower x-coordinate limit of the bounding box. This option is only relevant if `COORDFLAG=1`. This option is only used by rflupick.

`XCOORDUPP` Specifies the upper x-coordinate limit of the bounding box. This option is only relevant if `COORDFLAG=1`. This option is only used by rflupick.

`YCOORDLOW` Specifies the lower y-coordinate limit of the bounding box. This option is only relevant if `COORDFLAG=1`. This option is only used by rflupick.

`YCOORDUPP` Specifies the upper y-coordinate limit of the bounding box. This option is only relevant if `COORDFLAG=1`. This option is only used by rflupick.

`ZCOORDLOW` Specifies the lower z-coordinate limit of the bounding box. This option is only relevant if `COORDFLAG=1`. This option is only used by rflupick.



(a) (b)

Figure 2: Partitioning of a quadrilateral grid into two regions using `PARTMODE=2`, demonstrating importance of cell numbering.

`ZCOORDUPP` Specifies the upper z-coordinate limit of the bounding box. This option is only relevant if `COORDFLAG=1`. This option is only used by rflupick.

6.2.8 `PREP` Section

he `PREP` section is relevant only to rfluprep. It contains the following keywords:

`PARTMODE` Specifies the partitioning mode. It can take the following values:

`1` Partition the grid with a general method. This will in general lead to well-balanced, but not perfectly balanced, partitions.

`2` Partition the grid with an imposed mapping to get perfect load balancing. Note that this will work only if the cells in the grid are numbered like the cells in a structured grid and if the number of regions is a divisor of the number of cells of the unpartitioned grid. Furthermore, it is important to note that the ordering of the cells will have a very strong impact on the resulting partitioning, see Fig. 2. Depending on the numbering, it is possible to generate non-contiguous partitions.

`SURFFLAG` Specifies whether the interface-grid file for Surfdiver is to be written. It can take the following values:

`0` Do not write the interface-grid file.

`1` Write the interface-grid file.

Note that if rfluprep is compiled as part of GENx, the interface-grid file will always be written.

It is important to note that hardcoded initial solutions are defined after geometric transformations.

6.2.9 `PROBE` Section

`PROBE` Section!Format of The `PROBE` section consists of two subsections separated by the character `#`. The first subsection contains the following keyword:

`NUMBER` Specifies the number of probes.

Immediately following the keyword `NUMBER`, there must be n lines, where $n =$ `NUMBER`. Each line must contain three real values, which represent the x-, y-, and z-coordinate of a given probe. rflump attempts to find the cell whose centroid is closest to the specified coordinates.

The second part contains the following keywords:

`WRITIME` Offset in seconds at which data is written to probe files.

`WRIITER` Offset between iterations at which data is written to probe files.

`OPENCLOSE` Specifies whether probe files are to closed and opened after writing data. It can take the following values:

`0` Do not close and open probe file after writing data.

`1` Close and open probe file after writing data.

Closing and opening the probe file after writing data can be useful because this forces write buffers to be flushed.

It is important to note that the separation character `#` must be present even if the keywords in the second subsection are not included in the input file. Otherwise the remainder of the input file will not be read correctly.

An example `PROBE` section for unsteady flow is given below.

```
# PROBE
NUMBER 4
0.001 0.000  0.499
0.001 0.000 -0.499
9.999 0.000  0.499
9.999 0.000 -0.499
#
WRITIME 5.0E-4
OPENCLOSE 1
#
```

6.2.10 `REFERENCE` Section

he `REFERENCE` section contains the following keywords:

`ABSVEL` Reference velocity magnitude, [m/s].

`PRESS` Reference static pressure, [Pa].

`DENS` Reference density, [kg/m^3].

`CP` Reference specific heat coefficient at constant pressure, [J/kg K].

`GAMMA` Reference ratio of specific heats, [-].

`LENGTH` Reference length, [m]

`RENUM` Reference Reynolds number, [-].

`PRLAM` Reference laminar Prandtl number, [-].

`PRTURB` Reference turbulent Prandtl number, [-].

`SCNLAM` Reference laminar Schmidt number, [-].

`SCNTURB` Reference turbulent Schmidt number, [-].

6.2.11 `TIMESTEP` Section

he `TIMESTEP` section contains the following keywords:

`FLOWTYPE` Specifies whether flow steady or unsteady. It can take the following values:

- `0` Flow is steady.
- `1` Flow is unsteady.

Note that the value of this keyword influences the name of the flow-solution files.

`STARTITER` The iteration number from which the computation is to be started. Only relevant if `FLOWTYPE=0`.

`MAXITER` The iteration number at which the computation is to be stopped. A calculation is stopped if either the maximum number of iterations is reached, or if the norm of the density residual has fallen below the residual tolerance. Only relevant if `FLOWTYPE=0`.

`RESTOL` The tolerance for the density residual below which the computation is judged to be converged. A calculation is stopped if either the maximum number of iterations is reached, or if the norm of the density residual has fallen below the residual tolerance. Only relevant if `FLOWTYPE=0`.

`WRIITER` Offset between iterations at which flow files are to be written. Only relevant if `FLOWTYPE=0`.

`PRNITER` Offset between iterations at which convergence information is printed on screen and written to the convergence file. Only relevant if `FLOWTYPE=0`.

`STARTTIME` The time in seconds from which the computation is to be started. Only relevant if `FLOWTYPE=1`.

`MAXTIME` The time in seconds at which the computation is to be stopped. Only relevant if `FLOWTYPE=1`.

`TIMESTEP` The maximum time step in seconds to be used in the computation. Only relevant if `FLOWTYPE=1`.

`WRITIME` Offset in time in seconds at which flow files are to be written. For unsteady flows with moving grids, the grid files are written also. Only relevant if `FLOWTYPE=1`.

`PRNTIME` Offset in time in seconds at which convergence information is printed on screen and written to the convergence file. Only relevant if `FLOWTYPE=1`.

`DTMINLIMIT` The time step in seconds below which information will be printed out about the region and cell in which the minimum time step occurs. The additional information can be helpful in diagnosing whether small time steps are due to unexpectedly small cells or cells of poor quality. Only relevant if `FLOWTYPE=1`.

6.2.12 `TRANSFORM` Section

he `TRANSFORM` section is relevant only to rfluprep. It contains the following keywords:

`FLAG` Specifies whether the grid is to be scaled rotated. It can take the following values:

- `0` Do not scale and rotate the grid.
- `1` Scale and rotate the grid.

`SCALE_X` Scaling factor for x-component of coordinates.

`SCALE_Y` Scaling factor for y-component of coordinates.

`SCALE_Z` Scaling factor for z-component of coordinates.

`ANGLE_X` Angle of rotation around x-axis, in degrees. The angle of rotation is positive in the counter-clockwise direction when looking down the x-axis.

`ANGLE_Y` Angle of rotation around y-axis, in degrees. The angle of rotation is positive in the counter-clockwise direction when looking down the y-axis.

`ANGLE_Z` Angle of rotation around z-axis, in degrees. The angle of rotation is positive in the counter-clockwise direction when looking down the z-axis.

It is important to note that hardcoded initial solutions are defined after geometric transformations.

6.2.13 VISCMODEL Section

he VISCMODEL section contains the following keywords:

MODEL Specifies the viscosity model. It can take the following values:

0 Sutherland viscosity model. The viscosity is computed from:

$$\frac{\mu}{\mu_{\text{ref}}} = \left(\frac{T}{T_{\text{ref}}}\right)^{\frac{3}{2}} \frac{T_{\text{ref}} + S_{\text{ref}}}{T + S_{\text{ref}}}$$

where μ is the dynamic viscosity, μ_{ref} is the reference dynamic viscosity, T is the static temperature, T_{ref} is the reference temperature, and S_{ref} is the Sutherland constant.

1 Fixed viscosity. The viscosity value is given by the value assigned to the keyword VISCOSITY.

2 Antibes viscosity model. The viscosity is computed from:

$T_{\text{ref}} \geq 120\,\text{K}$:

$$\mu = \begin{cases} \mu_{\text{ref}} \left(\dfrac{T}{T_{\text{ref}}}\right)^{\frac{3}{2}} \dfrac{T_{\text{ref}} + S_{\text{ref}}}{T + S_{\text{ref}}} & \text{if} \quad T \geq 120\,\text{K} \\ \mu_{120}\, \dfrac{T}{120} & \text{if} \quad T < 120\,\text{K} \end{cases}$$

$T_{\text{ref}} < 120\,\text{K}$:

$$\mu = \begin{cases} \mu_{\text{ref}}\, \dfrac{T}{T_{\text{ref}}} & \text{if} \quad T < 120\,\text{K} \\ \mu_{120} \left(\dfrac{T}{120}\right)^{\frac{3}{2}} \dfrac{120 + S_{\text{ref}}}{T + S_{\text{ref}}} & \text{if} \quad T \geq 120\,\text{K} \end{cases}$$

This formula was specified by the organizers of the Workshop on Hypersonic Flows for Reentry Problems [1]. S_{ref} is given the value 110 K and μ_{120} denotes the value of the viscosity at 120 K.

VISCOSITY Reference value for dynamic viscosity [kg/(ms)].

REFTEMP Specifies the value of the Sutherland constant [K].

SUTHCOEF Specifies the value of the reference temperature T_{ref} in the Sutherland and Antibes models [K].

6.3 Grid Files

6.3.1 RocfluMP Grid File

6.3.2 CENTAUR Grid File

6.3.3 VGRIDns Grid Files

The .mapbc file produced by VGRIDns is not read.

6.3.3.1 `.vbc` File The `.vbc` file corresponds to the `.bc` file written by VGRIDns. It is renamed because the boundary-condition file of RocfluMP has the extension `.bc`.

6.3.3.2 `.cgosg` File

6.3.3.3 `.vgi` File The number of boundary patches of the grids generated by VGRIDns appears to be given by the number of surface patches in the CAD file. It is convenient to be able to merge boundary patches because CFD simulations typically employ only a limited set of boundary conditions and because many patches do not need to be addressed in isolation. The `.vgi` file contains user-provided information in ASCII format which allows merging of boundary patches.

The format of the `.vgi` file is as follows:

Line 1: The number of patches after the mapping.

Line 2: The number of mappings (hereafter referred to as `nMappings`).

The remaining `nMappings` lines: Each line contains three integers. The first two integers represent the lower and upper limits of the patches in the VGRIDns file which are to be mapped to the patch indicated by the third integer. The lower limit must be less than or equal to the upper limit.

The following is an example `.vgi` file:

```
 5
 8
 5  6  1
 1  4  2
12 12  2
 7  7  3
10 10  3
 8  9  4
11 11  4
17 22  5
```

This file indicates that five patches will exist after merging and that eight mappings are listed. For example, the first mapping specifies that patches five and six are mapped to patch one, and the last mapping specifies that patches 17 to 22 are mapped to patch five. Note that it is possible to map several original patches to a given new patch in separate mappings. That is, the following example is equivalent to the one given above:

```
 5
10
 5  5  1
```

```
 6  6  1
 1  4  2
12 12  2
 7  7  3
10 10  3
 8  9  4
11 11  4
17 18  5
19 22  5
```

It is important to note that the extrema of the lower and upper original patch indices must be equal to unity and the number of patches specified in the `.bc` file, respectively. Furthermore, the extrema of the new patch indices must be equal to unity and the number of patches specified on the first line, respectively.

In practice, it is usually impossible to specify the mappings without having visually inspected the grid. It is therefore recommended that in a first try, a one-to-one mapping is specified. By inspecting the grid, it is possible to merge appropriate patches by writing the proper mapping file.

6.3.4 MESH3D Grid Files

6.3.4.1 `.m3d` File `.mgi` File The `.mgi` file serves the same purpose as the `.vgi` file. The format of the `.mgi` file is:

Line 1: The number of patches in the `.m3d` file.

Line 2: The number of patches after the mapping.

Line 3: The number of mappings (hereafter referred to as `nMappings`).

The remaining `nMappings` lines: Each line contains three integers. The first two integers represent the lower and upper limits of the patches in the MESH3D file which are to be mapped to the patch indicated by the third integer. The lower limit must be less than or equal to the upper limit.

The first line contains the original number of patches because this information is not contained the `.m3d` file.

One important difference between the `.vgi` and `.mgi` files is that the extrema of the lower and upper limits of the original patches in the `.mgi` file do not have to be equal to unity and the number of patches specified on the second line, respectively. This is because boundary faces in the `.m3d` are grouped by arbitrary flags and not by patch numbers. Hence the following is a valid `.mgi` file:

```
22
```

```
2     7
3    10
4     5    6    1
5     1    4    2
6    12   12    2
7     7    7    3
8    10   10    3
9     8    9    4
10   11   11    4
11   17   22    5
12  200  600    6
13  100  100    7
```

Note in particular the last two lines: They illustrate that the upper limit of the original patches does not have to be equal to the number of patches in the `.m3d` file.

6.3.5 TETMESH Grid Files

6.3.5.1 `.noboite` File he format of the `.noboite` file is described in the TETMESH user's manual.

6.3.5.2 `.tmi` File

6.3.6 Cobalt Grid Files

6.3.6.1 `.cgr` File he first line of the Cobalt grid file contains three integers,

```
nDimensions  nZones  nBoundaryPatches
```

where the meaning is self-explanatory. For use within rflump, Cobalt grid files must satisfy the following restrictions: `nDimensions` must be equal to `3` and `nZones` must be equal to `1`.

The next line contains five integers,

```
nVertices  nFaces  nCells  nVerticesPerFaceMax  nFacesPerCellMax
```

where the last two quantities represent the maximum number of vertices defining a face and the maximum number of faces defining a cell. For example, if the grid consisted purely of tetrahedra, `nVerticesPerFaceMax` and `nFacesPerCellMax` would be equal to `3` and `4`, respectively. On the other hand, if the grid consisted of tetrahedra, prisms, and pyramids, `nVerticesPerFaceMax` and `nFacesPerCellMax` would be equal to `4` and `5`, respectively.

The next `nVertices` lines contain the x-, y-, and z-coordinates for each vertex.

The next `nFaces` lines contain the face-connectivity information,

```
nVerticesPerFace  <nVerticesPerFace> vertices  Cell1  Cell2
```

where `Cell1` and `Cell2` are the two cells which share a given face. If a face lies on a boundary patch, the respective cell is given by the negative patch index.

6.3.6.2 `.cgi` File sec:cgifile The `.cgi` file serves the same purpose as the `.vgi` and `.mgi` files. The format of the `.cgi` file is:

Line 1: The number of patches after the mapping.

Line 2: The number of mappings (hereafter referred to as `nMappings`).

The remaining `nMappings` lines: Each line contains three integers. The first two integers represent the lower and upper limits of the patches in the Cobalt file which are to be mapped to the patch indicated by the third integer. The lower limit must be less than or equal to the upper limit.

6.4 Flow-Solution File

The name of the flow solution file depends on whether a steady or unsteady flow is computed. The name of the flow solution file is:

- `<casename>.flo_mmmmm_nnnnnn` for steady flows
- `<casename>.flo_mmmmm_n.nnnnnnnE+nn` for unsteady flows

where `mmmmm` is the region number, `nnnnnn` is the iteration number, and `n.nnnnnnnE+nn` is the time stamp.

6.5 Boundary-Condition File

sec:bcfileformat The boundary-condition file is called `<casename>.bc`. Like the input file, the boundary-condition file is divided into sections. Each section contains several lines, each of which consists of a keyword and a value, with the exception of the line containing the keyword `PATCH`, on which two values are listed.

```
# SECTION_NAME
PATCH      PATCH_1  PATCH_2
KEYWORD_1  VALUE_1
KEYWORD_2  VALUE_2
#
```

Each section assigns a boundary condition to either a single patch or to a range of patches. The boundary-condition file must be terminated by the string:

```
# END
```

The sections may be listed in any order in the boundary-condition file, but are listed below in alphabetical order for simplicity.

6.5.1 Physical Boundary Conditions

6.5.1.1 Farfield Boundary: `BC_FARF` Section

he `BC_FARF` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`CORR` Specifies whether farfield point vortex correction is to be applied. It can take the following values:

- `0` Do not apply point-vortex correction.
- `1` Apply point-vortex correction.

Note that at present the correction can only be used for two-dimensional computations and has not been thoroughly tested.

`MACH` Mach number [-].

`ATTACK` Angle of attack [deg]. See Fig. 3 for the definition of `ATTACK`.

`SLIP` Sideslip angle [deg]. See Fig. 3 for the definition of `SLIP`.

`PRESS` Static pressure [Pa].

`TEMP` Static temperature [K].

6.5.1.2 Inflow Boundary: `BC_INFLOW/BC_INFLOW_TOTANG` Section

he `BC_INFLOW` or `BC_INFLOW_TOTANG` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`TYPE` Specifies whether inflow is supersonic or subsonic. It can take the following values:

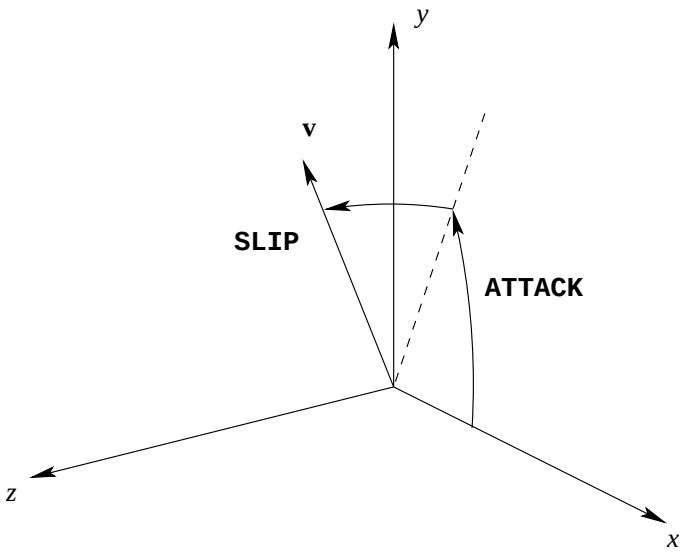


Figure 3: Definition of angles `ATTACK` and `SLIP` for farfield boundary condition. Angles are positive in direction of arrows.

`0` Inflow is supersonic. The Mach number must be specified by the keyword `MACH`.

`1` Inflow is subsonic.

`TTOT` Total temperature [K].

`PTOT` Total pressure [Pa].

`BETAH` Angle between velocity vector and its projection onto xz-plane [deg]. See Fig. 4 for the definition of `BETAH`.

`BETAV` Angle between the projection of the velocity vector onto the xz-plane and the positive x-axis [deg]. See Fig. 4 for the definition of `BETAV`.

`MACH` Mach number [-]. The Mach number must be specified only if the inflow is supersonic.

`FIXED` Specifies whether the flow is assumed to be normal to the boundary. It can take the following values:

`0` Inflow not assumed to be normal to boundary.

`1` Inflow assumed to be normal to boundary.

Specifying the flow to be normal to the boundary can be particularly helpful if the inflow boundary represents a reservoir condition or other conditions in which the flow velocity is very small. It is important to note that the motivation for and effect of `FIXED` is different from specifying the flow direction through `BETAH` and `BETAV`. The keyword `FIXED` influences the angle computed from the extrapolated velocity vector. For vanishing velocity, the determination of this angle becomes ill-conditioned which can lead to failure of computations. Instead, the angle can be fixed so that the velocity, no matter how small, is always normal to the boundary.

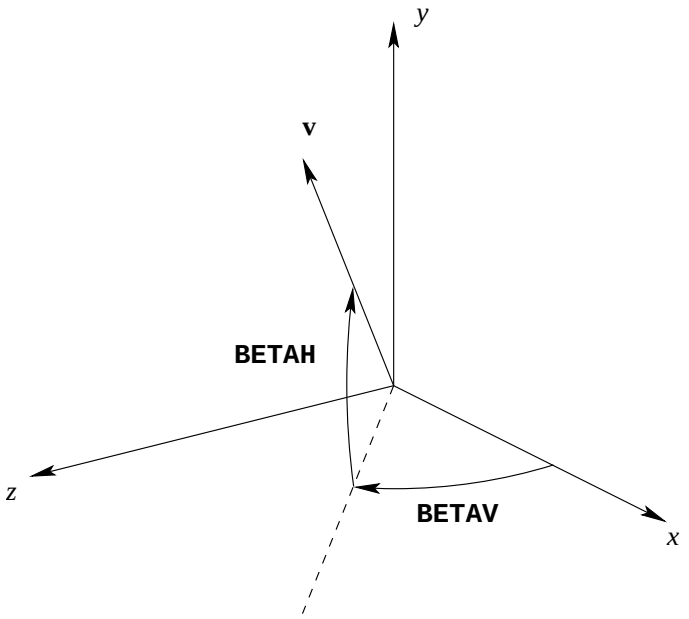


Figure 4: Definition of angles `BETAH` and `BETAV` for inflow boundary condition. Angles are positive in direction of arrows.

Specification of Inflow Angles. For planar boundary patches with known normal vector $\mathbf{n} = \{n_x, n_y, n_z\}^t$ and inflow normal to the boundary, the inflow angles are given by

$$\beta_v = \tan^{-1}\left(\frac{n_z}{n_x}\right), \quad (1)$$

$$\beta_h = -\sin^{-1} n_y. \quad (2)$$

For imposed velocities and flow which is not necessarily normal to the boundary, the inflow angles are given by

$$\beta_v = \tan^{-1}\left(\frac{w}{u}\right), \quad (3)$$

$$\beta_h = \sin^{-1}\left(\frac{v}{\|\mathbf{v}\|}\right). \quad (4)$$

The section name `BC_INFLOW` is obsolete and should be replaced by `BC_INFLOW_TOTANG`. For backward compatibility, RocfluMP treats the two boundary conditions in the same way.

6.5.1.3 Inflow Boundary: `BC_INFLOW_VELTEMP` Section

he `BC_INFLOW_VELTEMP` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`TYPE` Specifies whether inflow is supersonic or subsonic. It can take the following values:

`0` Inflow is supersonic. The static pressure must be specified by the keyword `PRESS`.

`1` Inflow is subsonic.

`VELX` x-component of fluid velocity [m/s].

`VELY` y-component of fluid velocity [m/s].

`VELZ` z-component of fluid velocity [m/s].

`TEMP` Static temperature [K].

`PRESS` Static pressure [Pa]. Only read if `TYPE=0`

6.5.1.4 Injection Boundary: `BC_INJECT` Section

he `BC_INJECT` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`MFRATE` Injection mass flux [kg/(m^2 s)].

`TEMP` Injection static temperature [K].

`RFVFU` Injection momentum flux in x-direction [kg/(m s^2)].

`RFVFV` Injection momentum flux in y-direction [kg/(m s^2)].

`RFVFW` Injection momentum flux in z-direction [kg/(m s^2)].

`COUPLED` Specifies whether patch is interacting during a computation with GENx. It can take the following values:

`1` Patch is interacting.

`2` Patch is not interacting.

`BFLAG` Specifies whether this patch is burning at $t = 0$. It can take the following values:

`0` Patch is not burning at $t = 0$.

`1` Patch is burning at $t = 0$.

6.5.1.5 No-Slip Boundary: `BC_NOSLIP` Section

he `BC_NOSLIP` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`ADIABAT` Specifies whether the wall is adiabatic. It can take the following values:

- 0 Wall is isothermal.
- 1 Wall is adiabatic.

`TWALL` Specifies value of the wall temperature if the wall is isothermal.

`COUPLED` Specifies whether patch is interacting during a computation with GENx. It can take the following values:

- 1 Patch is interacting.
- 2 Patch is not interacting.

6.5.1.6 Outflow Boundary: `BC_OUTFLOW` Section

he `BC_OUTFLOW` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`TYPE` Specifies whether outflow is supersonic or subsonic. It can take the following values:

- 0 Outflow is supersonic.
- 1 Outflow is subsonic. The static pressure must be specified by the keyword `PRESS`.
- 2 Outflow is mixed subsonic/supersonic. The static pressure must be specified by the keyword `PRESS`.

`PRESS` Static pressure [Pa]. The static pressure must only be specified if the outflow is subsonic or mixed subsonic/supersonic.

6.5.1.7 Slip Boundary: `BC_SLIPW` Section

he `BC_SLIPW` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

`COUPLED` Specifies whether patch is interacting during a computation with GENx. It can take the following values:

`0` Patch is interacting.

`2` Patch is not interacting.

6.5.1.8 Virtual Boundary: `BC_VIRTUAL` Section

The `BC_VIRTUAL` section contains the following keywords:

`NAME` Specifies the name of the boundary.

`PATCH` Specifies the range of patches to which the data in this section is to be applied using two integers.

This boundary condition should only be applied with truly two-dimensional computations, see the description of the keyword `DIMENS` in the `NUMERICS` Section. It is important to note that virtual boundaries can only be applied to $z =$ constant planes and must come in pairs.

The effect of this boundary condition is that no fluxes are computed. Because virtual boundaries always come in pairs and each cell in a truly two-dimensional computation must have one face each on the virtual boundaries, the effect of computing no fluxes on this boundary is to simulate constant properties in the z-direction.

6.5.2 Time-Dependent Boundary Conditions

ny of the physical quantities specified by the user in the above-described sections can be specified to vary in time. The time variations can be specified to be piecewise linear, sinusoidal, or stochastic using additional sections in the boundary-condition file. Each of these sections can be used to modify one user-specified physical quantity on one boundary patch. If more than one physical quantity on a given patch or physical quantities on several patches are to be modified, multiple additional sections have to be included in the boundary-condition file.

6.5.2.1 `TBC_PIECEWISE` Section

he `TBC_PIECEWISE` section allows the specification of piecewise constant and piecewise linear variations. Consider the following example section:

```
# TBC_PIECEWISE
INJECT    MFRATE ! BC and variable to which TBC applies
PATCH     1  1   ! applies to patch
ONTIME   -1.0E6  ! time to start using this TBC
OFFTIME   1.0E6  ! time to stop  using this TBC
ORDER     0      ! 0 = piecewise constant (default), 1 = piecewise linear
NJUMPS    4      ! number of points at which behavior changes
#
FRAC  0.0        ! fraction of input value of variable before first time
```

```
10 TIME  0.001      ! first time at which behavior changes
11 FRAC  0.1        ! next fraction attained (constant) or ramped to (linear)
12 TIME  0.002      ! second time at which behavior changes
13 FRAC  0.3
14 TIME  0.003
15 FRAC  0.6
16 TIME  0.004      ! final time at which behavior changes
17 FRAC  1.0        ! final value for constant case; *ignored* for linear case
18 #
```

Line 2: Specifies that the variable `MFRATE` on an `INJECT` boundary is to be modified. Note that this does not yet specify which injection boundary is to be modified.

Line 3: Specifies that variables on patch one are to be modified.

Line 4: Specifies the lower bound on the time window in which the values are to be modified.

Line 5: Specifies the upper bound on the time window in which the values are to be modified.

Line 6: Specifies the polynomial order of interpolation. It can take the following values:

`0` For piecewise constant interpolation.

`1` For piecewise linear interpolation.

Line 7: Specifies the number of data points through which the time-dependent behavior is specified.

Lines 9-17: Specify the behaviour of the user-specified variable in time. The behaviour is specified through pairs of values for the user-specified value and the time at which the variation changes. The user-specified value of the variable `MFRATE` is modified by the fraction `FRAC`; i.e., at any time, the actual time-dependent value is given by the product of `MFRATE` and `FRAC`. Figure 5 illustrates the piecewise constant and linear variations arising from the input of the above example. It is important to note that the piecewise constant and linear representations differ in their final value, because the final value of `FRAC` is ignored for linear variations.

6.5.2.2 `TBC_SINUSOIDAL` Section

he `TBC_SINUSOIDAL` section allows the specification of sinusoidal variations of the form

$$\alpha(t) = c\,(1 + A\sin(\omega t + \phi))$$

where c is the user-specified constant value, A is the amplitude if the sinusoid, ω is the angular frequency, and ϕ is the phase.

Consider the following example section:

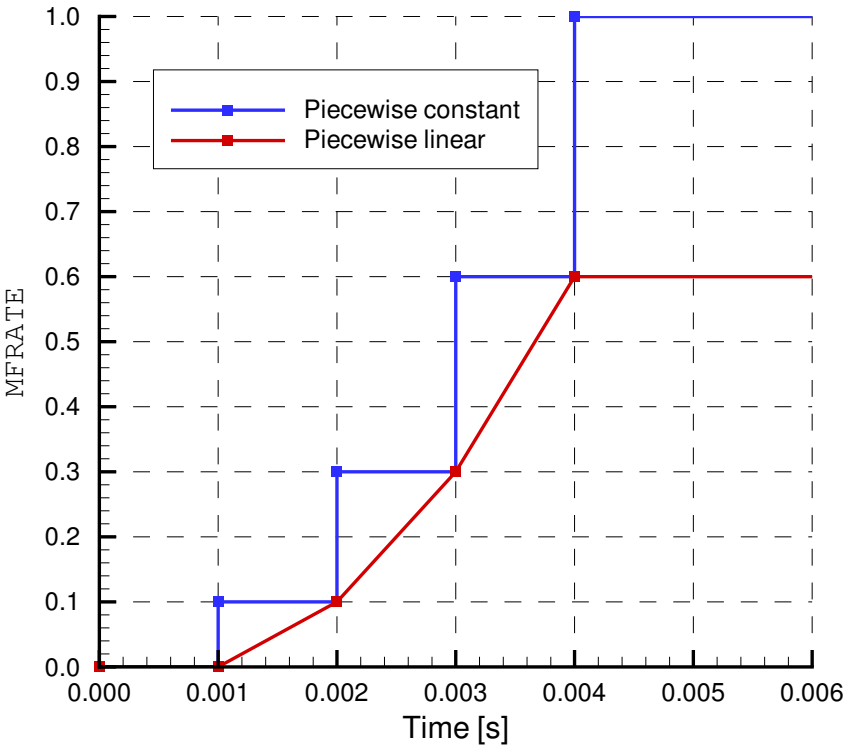


Figure 5: Illustration of piecewise constant and linear interpolations for `TBC_PIECEWISE` time-dependent boundary conditions.

```
1 # TBC_SINUSOIDAL
2 OUTFLOW   PRESS  ! BC and variable to which TBC applies
3 PATCH     2  2   ! applies to patch
4 ONTIME    1.0E-3 ! time to start using this TBC
5 OFFTIME   2.0E-3 ! time to stop  using this TBC
6 AMP       0.2    ! amplitude of sinusoid
7 FREQ      1.0E4  ! frequency of sinusoid
8 PHASE     0.0    ! argument of sin() for t=0
9 #
```

Line 2: Specifies that the variable `PRESS` of an `OUTFLOW` boundary is to be modified. Note that this does not yet specify which outflow boundary is to be modified.

Line 3: Specifies that variables on patch two are to be modified.

Line 4: Specifies the lower bound on the time window in which the values are to be modified.

Line 5: Specifies the upper bound on the time window in which the values are to be modified.

Line 6: Specifies the amplitude A of the sinusoidal variation.

Line 7: Specifies the angular frequency ω of the sinusoidal variation [rad/s].

Line 8: Specifies the phase ϕ of the sinusoidal variation [deg].

6.5.2.3 `TBC_STOCHASTIC` Section

6.5.2.4 `TBC_WHITENOISE` Section

6.5.3 Grid-Motion Boundary Conditions

omputations with moving boundaries require the specification of boundary conditions for the grid-motion algorithm. The following keywords can be specified in any of the above sections:

`MVPATCH` Specifies whether patch is moving. It can take the following values:

- `0` Patch is not moving.
- `1` Patch is moving.

`SMGRID` Specifies whether grid on patch is to be smoothed. It can take the following values:

- `0` Do not smooth surface grid.
- `1` Smooth surface grid.

It is important to note that surface grids can only be smoothed if the associated patches are flat. rflump checks whether patches with active smoothing are flat. If such patches are not flat, smoothing is deactivated automatically and a warning is printed to the screen.

`MOVEDIR` Specifies in which direction(s) the vertices on the patch are allowed to move. The direction(s) in which movement is allowed are indicated by integers:

`0` Vertices on patch are not allowed to be moved in any direction.

`1` Vertices on patch are allowed to be moved in x-coordinate direction.

`2` Vertices on patch are allowed to be moved in y-coordinate direction.

`4` Vertices on patch are allowed to be moved in z-coordinate direction.

The non-zero values can be combined to specify movement in planes normal to coordinate axes or arbitrary movement:

`3` Vertices on patch are allowed to be moved in xy-plane, i.e., movement normal to the z-coordinate direction is not allowed.

`5` Vertices on patch are allowed to be moved in xz-plane, i.e., movement normal to the y-coordinate direction is not allowed.

`6` Vertices on patch are allowed to be moved in yz-plane, i.e., movement normal to the x-coordinate direction is not allowed.

`7` Vertices on patch are allowed to be moved in xyz-space.

6.6 Dimension File

Dimension file

6.7 Restart-Information File

Restart-information file The restart-information file contains the iteration numbers or time stamps at which restart files were written by rflump. It is read by rflump to determine the iteration number or time stamp from which a restart should be made. Restarts are always made from the iteration number or time stamp on the last line in the restart-information file. The initial restart-information file is written by rfluprep.

The restart-information file allows jobs to be restarted automatically without user intervention. This is particularly useful when running on computers which require jobs to be submitted through batch queues.

6.8 Convergence File

Convergence file The convergence file is called `<casename>.con` and is written in ASCII format. For steady flows, the convergence file contains the following information:

Column 1: The iteration number.

Column 2: The residual.

Column 3: The x-component of the net force on solid walls.

Column 4: The y-component of the net force on solid walls.

Column 5: The z-component of the net force on solid walls.

Column 6: The mass flow entering the solution domain.

Column 7: The mass flow exiting the solution domain.

For unsteady flows, the convergence file contains the following information:

Column 1: The time.

Column 2: The time step.

Column 3: The x-component of the net force on solid walls.

Column 4: The y-component of the net force on solid walls.

Column 5: The z-component of the net force on solid walls.

Column 6: The mass flow entering the solution domain.

Column 7: The mass flow exiting the solution domain.

6.9 Probe File

Probe file The probe file is written by rflump if the input file contains the `PROBE` section and setting the variable `NUMBER` to an integer greater than `0`.

The name of the probe file is `<casename>.prb_mmmmm`, where `mmmmm` is the number of the probe.

Each line of the probe file consists of seven columns, which contain the following pieces of data:

Column 1: For steady flows, the iteration number; for unsteady flows, the time.

Column 2: The density.

Column 3: The x-velocity component.

Column 4: The y-velocity component.

Column 5: The z-velocity component.

Column 6: The static pressure.

Column 7: The static temperature.

6.10 Mass-Conservation Check File

he mass-conservation check file is written by rflump for unsteady flows with grid motion outside of GENx. The file is called `<casename>.mass` and contains the following information:

Column 1: The time.

Column 2: The mass contained in the solution domain.

Column 3: The mass flow entering the solution domain.

Column 4: The mass flow exiting the solution domain.

Column 5: The volume of the solution domain.

As implied by its name, the mass-conservation check file is used to check that mass is conserved during moving-grid computations.

6.11 Statistics File

6.12 GENx Control File

When rflump is run within GENx, an additional input file is required because rflump is not invoked from the command line. This so-called GENx control file is always called `RocfluControl.txt` and contains the following information:

Line 1: The case name.

Line 2: The directory name containing the rflump input files, relative to the directory from which GENx is invoked.

Line 3: The directory name containing the rflump output files, relative to the directory from which GENx is invoked. written by Roccom.

Line 4: The verbosity level.

Line 5: The checking level.

7 Problem Setup

his chapter gives a summary of the problem setup. The objective is to assist users in setting up a run, especially for coupled runs within GENx.

1. Generate a grid. The names and number of output files will differ depending on which grid generator is used:

   - CENTAUR: Output file is `*.hyb.asc` or `*.hyb.bin`. Rename the output files to `<casename>.hyb.asc` or `<casename>.hyb.bin`.
   - VGRIDns: Output files are `*.bc` and `*.cgosg`. Rename `*.bc` to `<casename>.vbc`. This is important because rflump uses the extension `.bc` for its boundary-condition file. Rename `*.cgosg` to `<casename>.cgosg`. Generate a patch-mapping file `<casename>.vgi` as described in Sec. 6.3.3.3.
   - MESH3D: Output file is `*.m3d`. Rename to `<casename>.m3d`. Generate a patch-mapping file `<casename>.mgi` as described in Sec. 6.3.4.1.
   - TETMESH: Output file is `*.noboite`. Rename to `<casename>.noboite`. Generate a patch-mapping file `<casename>.tmi` as described in Sec. 6.3.5.2.
   - GRIDGEN: Specify Cobalt as the output format. Output files are `*.bc` and `*.inp`. Because these extensions are used by rflump, it is recommended that you rename these files as `*-COBALT.bc` and `*-COBALT.inp`. Link `<casename>.cgr` to `*-COBALT.inp`,

     ```
     ln -s *-COBALT.inp <casename>.cgr
     ```

     Generate a patch-mapping file `<casename>.cgi` as described in Sec. ??.

2. Generate a boundary-condition file `<casename>.bc`. The format of the boundary-condition file is described in ??. Pay particular attention to the following:

   - Make sure that the patch-mapping files generated above are consistent with the boundary-condition file, i.e., each patch in the grid is mapped to a patch in the boundary-condition file. Changing the patch-mapping file means that the preprocessor will need to be rerun.
   - For coupled simulations: Make sure that each patch has the correct values for the following parameters: `COUPLED`, `MVPATCH`, and `SMGRID`. Incorrect values of the `COUPLED` parameter are likely to lead to failure of Surfdiver.

3. Generate an input file `<casename>.inp`. Pay particular attention to the following sections:

   - Format of grid file: Make sure you set the `GRIDSRC` keyword in the `FORMATS` section to the correct value.

- Initial condition: Make sure that the initial condition is correct because changing the initial condition means that the preprocessor will need to be rerun. In particular, make sure that the initial condition is sensible and compatible with the boundary conditions.
- Reference values: Make sure that the values for `CP` and `GAMMA` are correct. They have a very strong effect on steady-state pressure levels for typical rocket problems. Changing these values influences the initial solution and means that the preprocessor will need to be rerun.
- For coupled computations, make sure that the `PREP` section contains the keyword `SURFFLAG` set to `1` if rfluprep was not compiled as part of GENx.
- For coupled computations, make sure that the units are correct. If the grid was generated in units other than meters (as is often the case), add a `TRANSFORM` section with the appropriate scaling factors. Inconsistent units will lead to failure of Surfdiver.

4. Generate a processor-mapping file if parallel runs are to be made by executing

```
rflumap -c <casename> -m 1 -p <nprocs> -r <nregions> -v <verbosity>
```

   Note that the number of regions can be larger than the number of processes. See Sec. 4.3.

5. Run the partitioning module by executing

```
rflupart -c <casename> -v <verbosity>
```

   See Sec. 4.5. The partitioning module generates grid files in rflump format.

6. Run the initialization module by executing

```
rfluinit -c <casename> -v <verbosity>
```

   See Sec. 4.2. The initialization module generates solution files in rflump format.

7. For coupled computations, you need to generate an input file for Rocin by running rflumap again,

```
rflumap -c <casename> -m 2 -p <nprocs> -r <nregions> -v <verbosity>
```

   See Sec. 4.3.

8. For coupled computations, generate the `RocfluControl.txt` file, see Sec. 6.12.

9. You are ready to run either rflump or GENx.

8 Example Cases

his chapter illustrates the execution of rflumap, rflupart, rfluinit, rflump, and rflupost for several example cases. For the sake of brevity, the output produced by these program is not shown.

8.1 Shocktube

he flow in a shocktube is used to test the shock-capturing capabilities of the spatial discretization and the accuracy of the temporal discretization. The CVS repository contains coarse and fine grids for the first and second cases specified by Sod [6]. The grid is not shown on account of the simple geometry.

The following files are required for this case:

```
shocktube.bc
shocktube.inp
shocktube.hyb.bin
```

The `GRIDSRC` variable in the `FORMATS` section of the `shocktube.inp` file indicates that the original grid file is in CENTAUR format.

By typing

```
rflupart -c shocktube -v 2
```

the partitioner runs and writes the following output files:

```
shocktube.dim
shocktube.grd_00000
```

The initialization module is executed by typing

```
rfluinit -c shocktube -v 2
```

which produces the file

```
shocktube.flo_00000_000000
```

The solver may be run by typing

```
rflump -c shocktube -v 2
```

With the settings contained in the files from the CVS repository, rflump will take as many time steps as needed to reach a physical time of $t = 1.0 \cdot 10^{-3}$ s.

The results may be visualized with TECPLOT. The postprocessor is used to interpolate the solution variables from cell centroids to vertices and to write the TECPLOT data file by typing

```
rflupost -c shocktube -s 1.0E-3 -v 2
```

The resulting data file may be read into TECPLOT by typing

```
tecplot shocktube.plt
```

The contours of density are depicted in Fig. 6

The CVS repository contains a TECPLOT macro file called `lineplots.mcr` which can be used to extract line plots of density, velocity, and pressure along the x-axis as shown in Fig. 7.

8.2 GAMM Bump

he transonic flow over a circular arc bump in a straight-walled channel is often used to test the inflow and outflow boundary conditions and the shock-capturing capabilities of the spatial discretization. This case will be used to illustrate both serial and parallel computations.

The grid used in this computation is shown in Fig. 8. The green surface indicates the inflow boundary. The outflow boundary is hidden from view in this figure.

The following files are required for this case:

```
gamm8.bc
gamm8.inp
gamm8.hyb.bin
```

The `GRIDSRC` variable in the `FORMATS` section of the `gamm8.inp` file indicates that the original grid file is in CENTAUR format.

8.2.1 Serial Computation

y typing

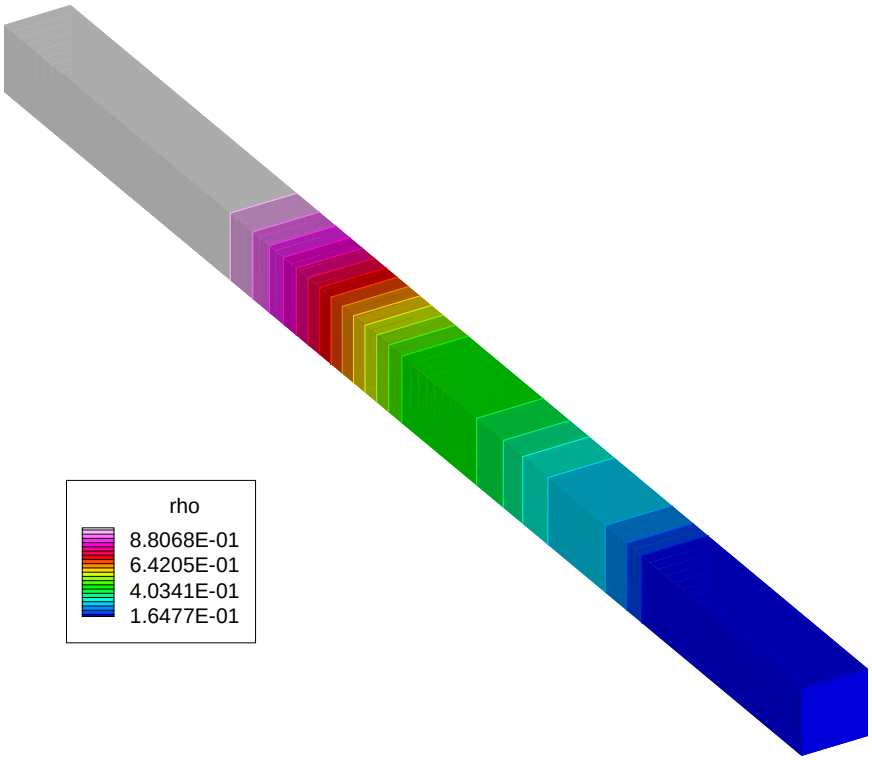


Figure 6: Density contours for shock-tube computation at $t = 1.0 \cdot 10^{-3}$ s with initial conditions given by Sod's first case.

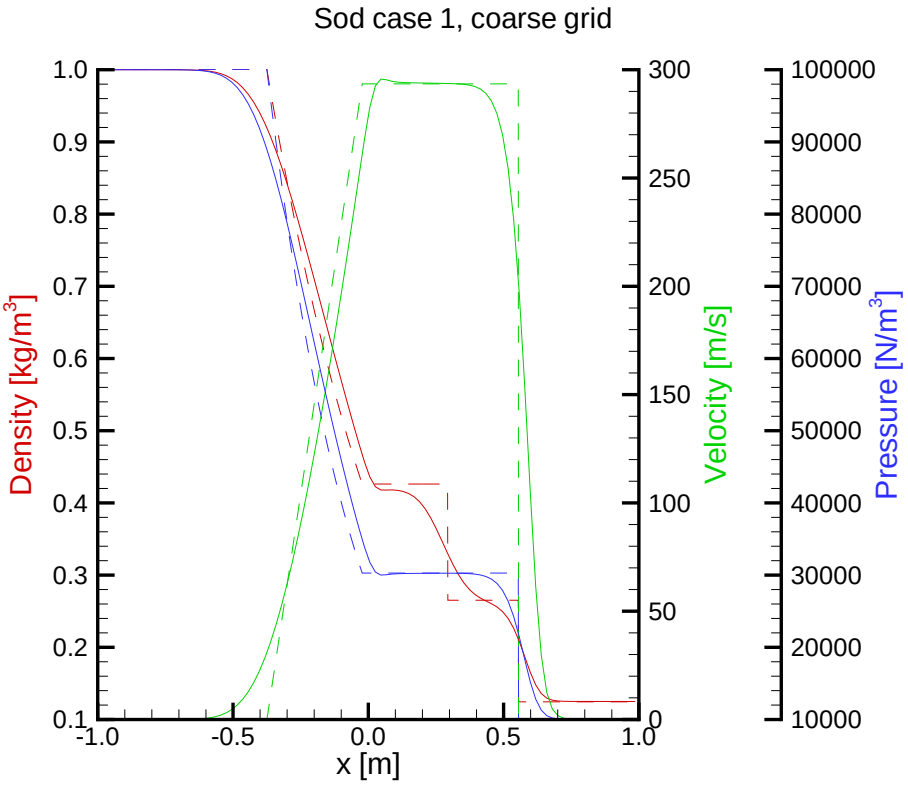


Figure 7: Line plots for shock-tube computation at $t = 1.0 \cdot 10^{-3}$ s with initial conditions given by Sod's first case.

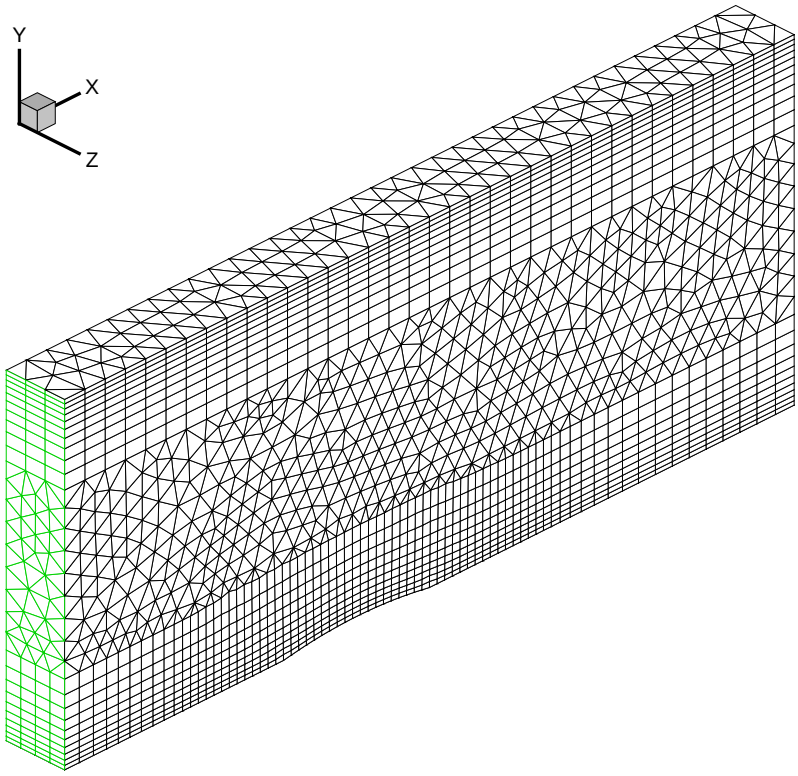


Figure 8: Grid used for GAMM bump computation.

```
rflupart -c gamm8 -v 2
```

the partitioner runs and writes the following output files:

```
gamm8.dim
gamm8.grd_00000
gamm8.flo_00000_000000
```

By typing

```
rfluinit -c gamm8 -v 2
```

the partitioner runs and writes the following output files:

```
gamm8.flo_00000_000000
```

The solver may be run by typing

```
rflump -c gamm8 -v 2
```

With the settings contained in the files from the CVS repository, rflump will take 500 steps to reach the convergence tolerance.

The results may be visualized with TECPLOT. The postprocessor is used to interpolate the solution variables from cell centroids to vertices and to write the TECPLOT data file by typing

```
rflupost -c gamm8 -s 500 -v 2
```

The resulting data file may be read into TECPLOT by typing

```
tecplot gamm8.plt
```

The contours of static pressure are depicted in Fig. 9

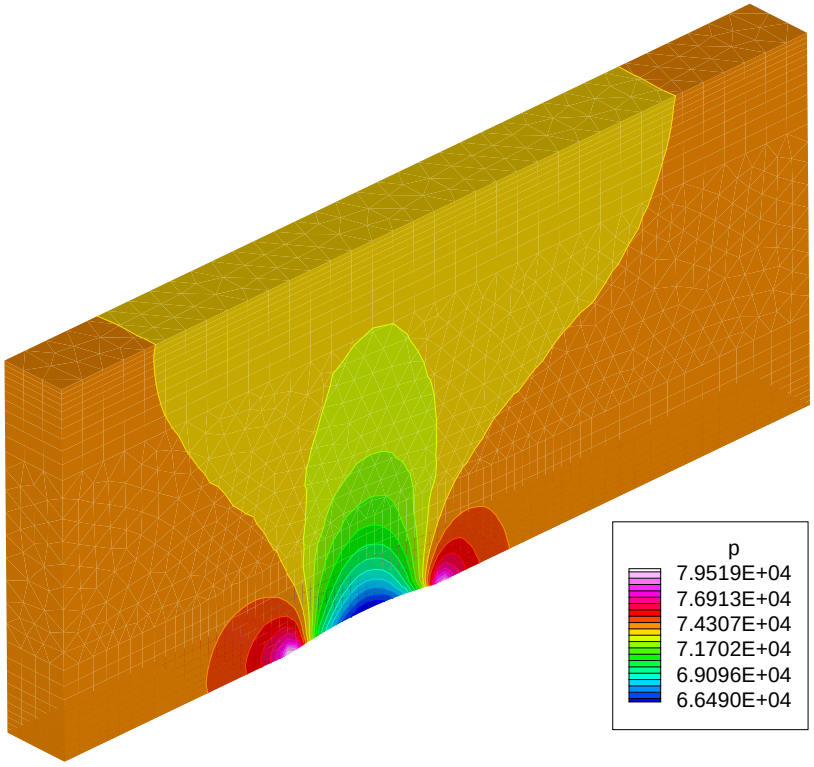


Figure 9: Static pressure contours for transonic GAMM bump computation.

8.2.2 Parallel Computation

he first step in preparing for a parallel computation is to decide how many regions should be used. For this example, we will use five regions and hence type

```
rflumap -c gamm8 -m 1 -p 5 -r 5 -v 2
```

which leads to the region-mapping file

```
gamm8.map
```

To partition the grid files, the partitioner is invoked by typing

```
rflupart -c gamm8 -v 2
```

which produces the following files:

```
gamm8.dim_00001    gamm8.grd_00001
gamm8.dim_00002    gamm8.grd_00002
gamm8.dim_00003    gamm8.grd_00003
gamm8.dim_00004    gamm8.grd_00004
gamm8.dim_00005    gamm8.grd_00005
```

To initialize the solution, the initializer is invoked by typing

```
rfluinit -c gamm8 -v 2
```

which produces the following files:

```
gamm8.flo_00001_000000
gamm8.flo_00002_000000
gamm8.flo_00003_000000
gamm8.flo_00004_000000
gamm8.flo_00005_000000
```

The parallel computation is initiated by

```
mpirun -np 5 rflump -c gamm8 -v 2
```

The precise command line may vary depending on which computer is used

The results of the computation will not be shown again. Instead, the capability of rflupost to allow the user to visualize the partitioned geometry will be demonstrated. Generate the TECPLOT file through

```
rflupost -c gamm8 -s 500 -v 2
```

Figure 10 depicts the five regions generated for this particular run. More importantly, rflupost can be used to visualize individual regions, the interregion faces, the virtual faces associated with particular regions and boundary patches, as well as the virtual cells of any region. Some of these capabilities are illustrated in Fig. 11.

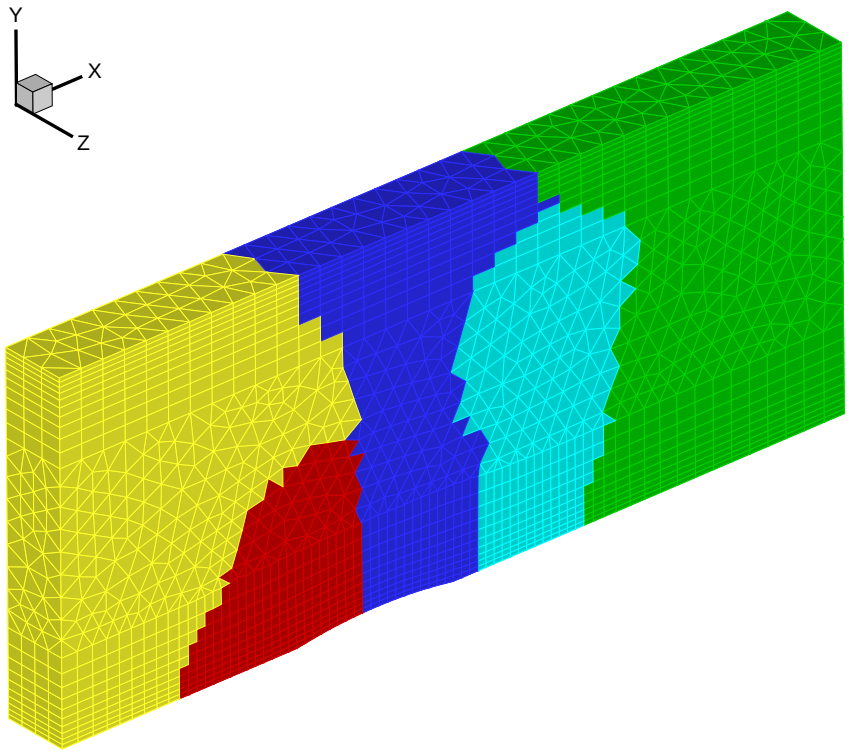


Figure 10: Partitioned grid for parallel GAMM bump computation.

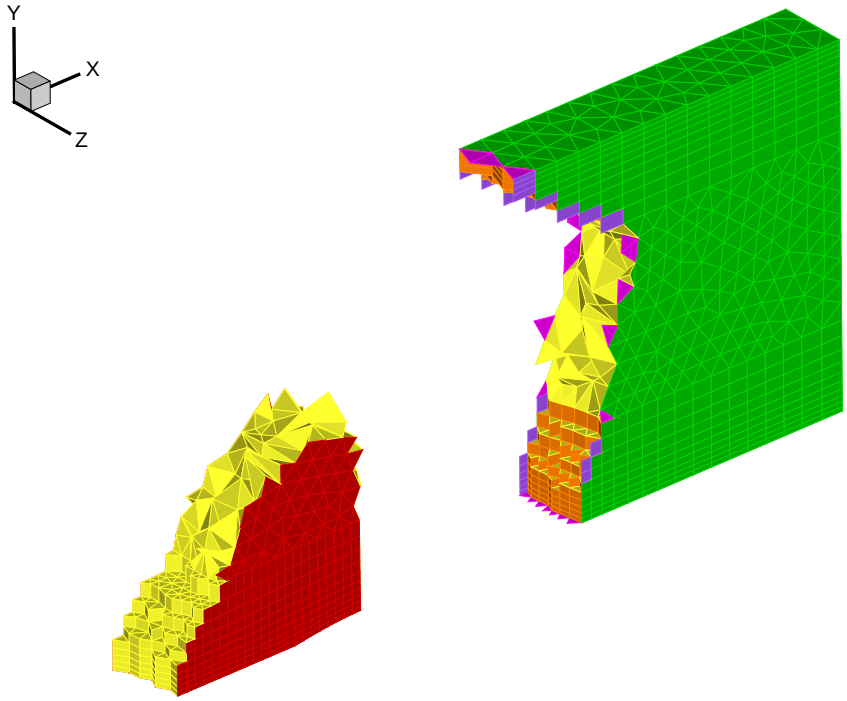


Figure 11: Partitioned grid for parallel GAMM bump computation, showing interregion faces as well as virtual boundary faces.

9 Visualization

This chapter describes how results obtained with rflump can be visualized. The focus is on presenting information which will be helpful in visualizing results obtained specifically with rflump. Consult the appropriate manuals for information on how the visualization tools themselves should be used.

This chapter assumes that the user has executed rflupost already and thereby produced one or more files for visualization. The behavior of rflupost is influenced by the values assigned to the keywords in the `POST` Section of the input file.

At present, rflupost writes output files for TECPLOT only.

9.1 TECPLOT

he TECPLOT files produced by rflupost are in binary format to reduce file size and loading time. The binary file are compatible across machines, i.e., files written on little-endian machines can be read on big-endian machines and vice versa. The files produced by rflupost make use of TECPLOT features which require version 10 or newer.

9.1.1 File Naming Convention

File Naming Convention!TECPLOT files The name of the TECPLOT files produced by rflupost depends on whether a steady or unsteady flow is computed. The name of the TECPLOT file is:

- `<casename>_<iteration stamp>.plt` for steady flows, where `<iteration stamp>` is a six-digit string denoting the iteration number.
- `<casename>_<time stamp>.plt` for unsteady flows, where `<time stamp>` is a string of the form `n.nnnnnnnE+nn` denoting time.

It should be noted that the TECPLOT files produced for unsteady flows cannot be loaded into TECPLOT from the command-line because TECPLOT interprets the character + as combining two file names. Instead, these files need to be loaded into TECPLOT using the `File -> Load Data File(s)` option after having started TECPLOT from the command line.

9.1.2 File Content

Solution Data rflupost writes all conserved and dependent variables into the TECPLOT files. In the TECPLOT file, the variables are denoted by dedicated names as listed in Table 1.

9.1.2.1 Patch-Coefficient Data

Table 1: Names and meaning of variables written to TECPLOT files.

Variable	Meaning	Units
r	Density	kg/m^3
ru	x-component of momentum	$kg/(m^2\,s)$
rv	y-component of momentum	$kg/(m^2\,s)$
rw	z-component of momentum	$kg/(m^2\,s)$
rE	Total internal energy	J/m^3
p	Pressure	Pa
T	Temperature	K
a	Speed of sound	m/s

9.1.3 Zone Naming Conventions

Zone Naming Conventions!TECPLOT files TECPLOT organizes data by zones. Each zone is assigned a name and may consist of either volume or surface data. Because TECPLOT allows each zone to be activated (displayed) or deactived (not displayed) separately, the separation of data into zones is useful in allowing detailed and selective investigation of data. For this reason, the files produced by rflupost make extensive use of zones. The following explains the zone naming conventions used by rflupost.

The zone naming convention adopted by rflupost is best explained with reference to Fig. 12. Data written to the TECPLOT file is split into volume and boundary patch data. Volume data is further split according to the cell type (tetrahedron, hexahedron, prism, or pyramid), and, for each cell type, according to the cell kind (actual or virtual cell). Boundary patch data is further split according to the face type (triangular or quadrilateral), and, for each face type, according to the face kind (actual or virtual face).

9.1.3.1 Volume Zones

he names of volume zones are:

```
<cell type>-<cell kind>_<region index>
```

where

`<cell type>` is a four-letter string indicating the cell type:

`TET` if the zone consists of tetrahedral cells.

`HEX` if the zone consists of hexahedral cells.

`PRI` if the zone consists of prismatic cells.

`PYR` if the zone consists of pyramidal cells.

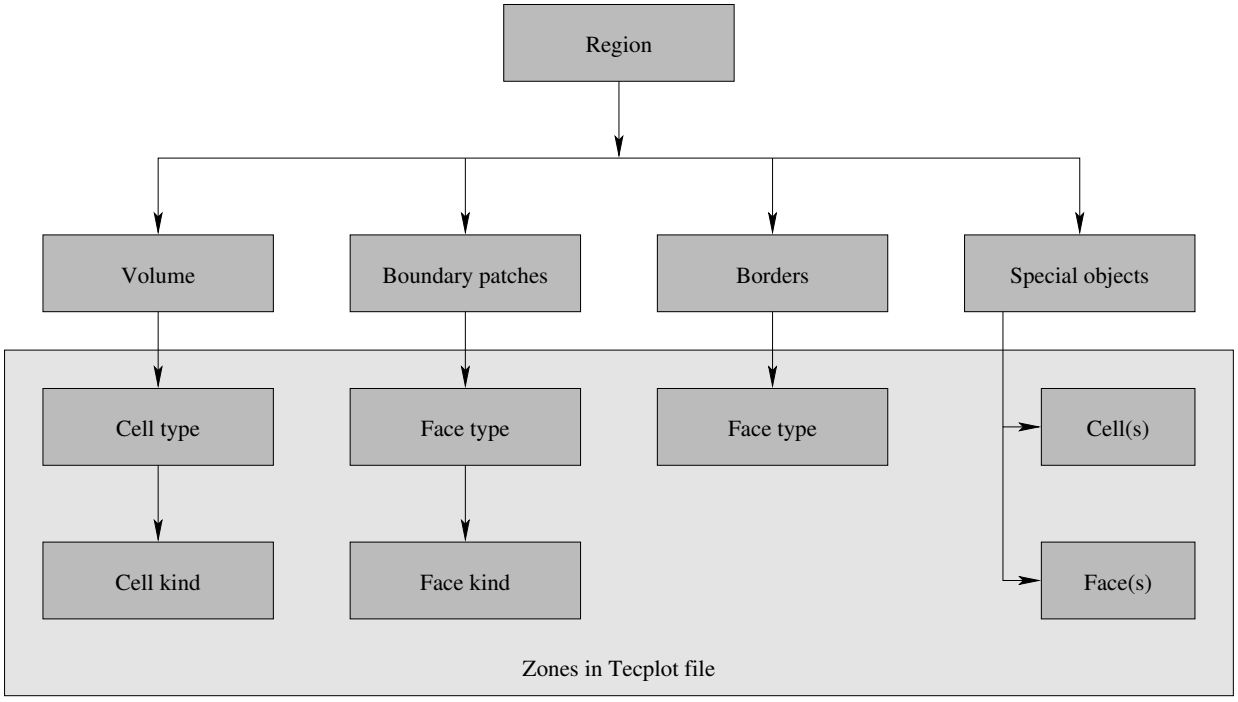


Figure 12: Illustration of zones written to TECPLOT files by rflupost.

`<cell kind>` is a one-letter string indicating the cell kind:

`A` if the zone consists of actual cells.

`V` if the zone consists of virtual cells.

`<region index>` is a five-digit string indicating the global region index.

Examples are shown in Fig. 13, which represents a screen dump of part of the zone style menu in TECPLOT. Zone number 1 contains the actual hexahedra of region 1 while zone number 23 contains the virtual prisms of region 2.

9.1.3.2 Boundary Patch Zones

he names of boundary patch zones are:

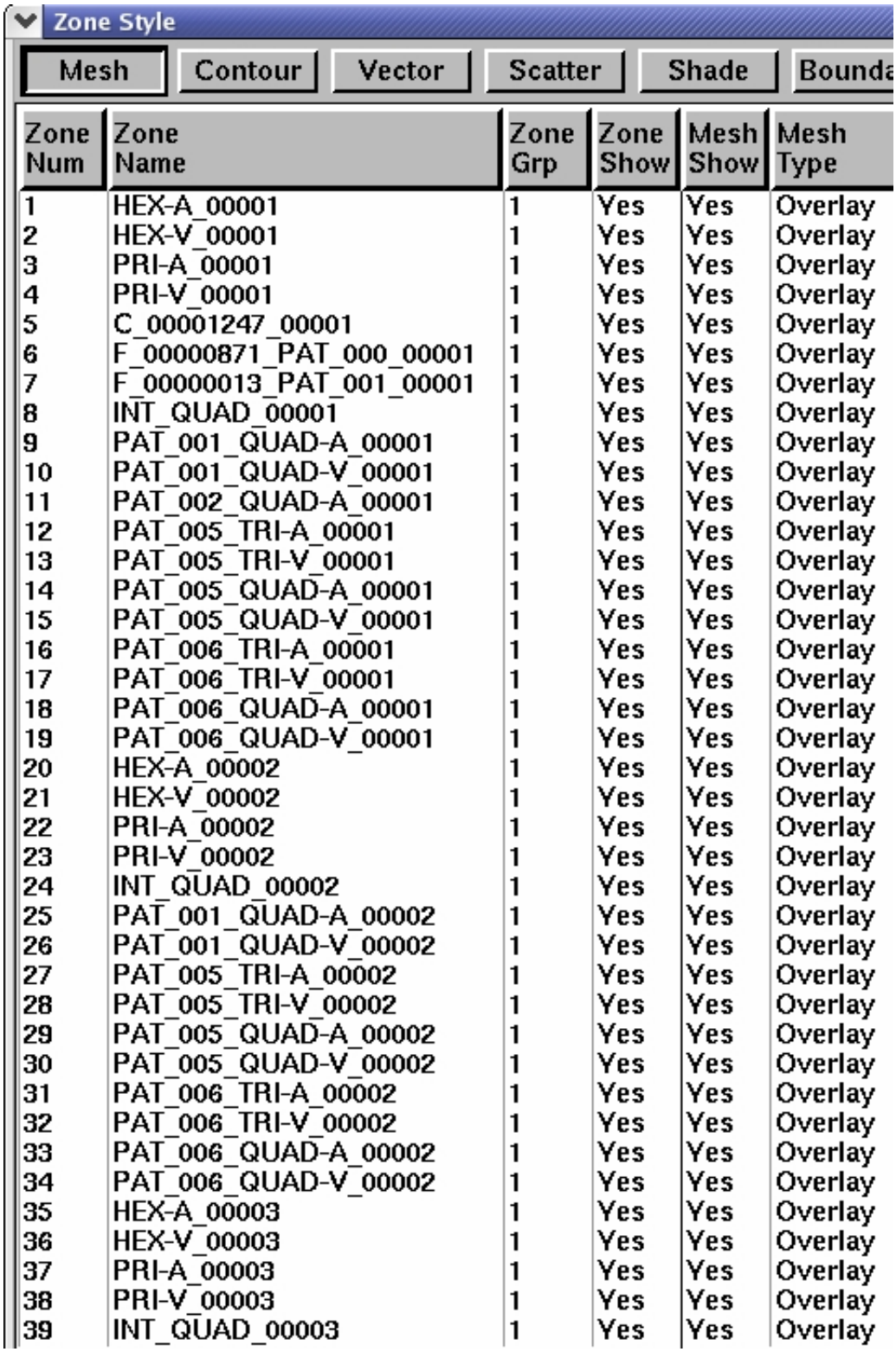
```
PAT_<patch index>_<face type>-<face kind>_<region index>
```

where

`<patch index>` is a three-digit string indicating the global patch index. It is important to note that this is a *global* patch index.

`<face type>` is a string indicating the face type:

`TRI` if the zone consists of triangular cells.



Zone Num	Zone Name	Zone Grp	Zone Show	Mesh Show	Mesh Type
1	HEX-A_00001	1	Yes	Yes	Overlay
2	HEX-V_00001	1	Yes	Yes	Overlay
3	PRI-A_00001	1	Yes	Yes	Overlay
4	PRI-V_00001	1	Yes	Yes	Overlay
5	C_00001247_00001	1	Yes	Yes	Overlay
6	F_00000871_PAT_000_00001	1	Yes	Yes	Overlay
7	F_00000013_PAT_001_00001	1	Yes	Yes	Overlay
8	INT_QUAD_00001	1	Yes	Yes	Overlay
9	PAT_001_QUAD-A_00001	1	Yes	Yes	Overlay
10	PAT_001_QUAD-V_00001	1	Yes	Yes	Overlay
11	PAT_002_QUAD-A_00001	1	Yes	Yes	Overlay
12	PAT_005_TRI-A_00001	1	Yes	Yes	Overlay
13	PAT_005_TRI-V_00001	1	Yes	Yes	Overlay
14	PAT_005_QUAD-A_00001	1	Yes	Yes	Overlay
15	PAT_005_QUAD-V_00001	1	Yes	Yes	Overlay
16	PAT_006_TRI-A_00001	1	Yes	Yes	Overlay
17	PAT_006_TRI-V_00001	1	Yes	Yes	Overlay
18	PAT_006_QUAD-A_00001	1	Yes	Yes	Overlay
19	PAT_006_QUAD-V_00001	1	Yes	Yes	Overlay
20	HEX-A_00002	1	Yes	Yes	Overlay
21	HEX-V_00002	1	Yes	Yes	Overlay
22	PRI-A_00002	1	Yes	Yes	Overlay
23	PRI-V_00002	1	Yes	Yes	Overlay
24	INT_QUAD_00002	1	Yes	Yes	Overlay
25	PAT_001_QUAD-A_00002	1	Yes	Yes	Overlay
26	PAT_001_QUAD-V_00002	1	Yes	Yes	Overlay
27	PAT_005_TRI-A_00002	1	Yes	Yes	Overlay
28	PAT_005_TRI-V_00002	1	Yes	Yes	Overlay
29	PAT_005_QUAD-A_00002	1	Yes	Yes	Overlay
30	PAT_005_QUAD-V_00002	1	Yes	Yes	Overlay
31	PAT_006_TRI-A_00002	1	Yes	Yes	Overlay
32	PAT_006_TRI-V_00002	1	Yes	Yes	Overlay
33	PAT_006_QUAD-A_00002	1	Yes	Yes	Overlay
34	PAT_006_QUAD-V_00002	1	Yes	Yes	Overlay
35	HEX-A_00003	1	Yes	Yes	Overlay
36	HEX-V_00003	1	Yes	Yes	Overlay
37	PRI-A_00003	1	Yes	Yes	Overlay
38	PRI-V_00003	1	Yes	Yes	Overlay
39	INT_QUAD_00003	1	Yes	Yes	Overlay

Figure 13: Screen dump of part of zone style menu in TECPLOT illustrating zone naming convention used in rflupost.

`QUAD` if the zone consists of quadrilateral cells.

`<face kind>` is a one-letter string indicating the face kind:

`A` if the zone consists of actual faces.

`V` if the zone consists of virtual faces.

`<region index>` is a five-digit string indicating the global region index.

Examples are shown in Fig. 13, which represents a screen dump of part of the zone style menu in TECPLOT. Zone number 9 contains the actual quadrilateral faces on global patch number 1 in region 1. Zone number 13 contains the virtual triangular faces on global patch number 5 in region 1. Note once again that the patch numbers are *global* patch numbers. Global patch number 5 corresponds to local patch number 3 in region 1.

9.1.3.3 Border Face Zones

he names of border face zones are:

```
INT-<face type>_<region index>
```

where

`<face type>` is a string indicating the face type:

`TRI` if the zone consists of triangular cells.

`QUAD` if the zone consists of quadrilateral cells.

`<region index>` is a five-digit string indicating the global region index.

An example is shown in Fig. 13, which represents a screen dump of part of the zone style menu in TECPLOT. Zone number 8 contains the quadrilateral border faces of region 1. In this particular case, the zone contains only quadrilateral faces.

9.1.3.4 Special Cell Zones

he names of special cell zones are:

```
C_<cell index>_<region index>
```

where

`<cell index>` is an eight-digit string indicating the cell index.

`<region index>` is a five-digit string indicating the global region index.

Note that the names of special cell zones do not distinguish between cell types and cell kinds. An example is shown in Fig. 13, which represents a screen dump of part of the zone style menu in TECPLOT. Zone number 5 contains cell number 1247 of region 1.

9.1.3.5 Special Face Zones

he names of special cell zones are:

```
F_<face index>_PAT_<patch index>_<region index>
```

where

`<face index>` is an eight-digit string indicating the face index.

`<patch index>` is a three-digit string indicating the global patch index. It is important to note that this is a *global* patch index. The patch number is zero if the face is an interior face.

`<region index>` is a five-digit string indicating the global region index.

Note that the names of special face zones do not distinguish between faces types. Examples are shown in Fig. 13, which represents a screen dump of part of the zone style menu in TECPLOT. Zone number 6 contains face number 1247 of region 1. Zone number 7 contains face number 13 of global patch 1 of region 1.

10 Troubleshooting

classifies problems arising during execution into two categories:

1. Non-critical problems: rflump attempts to make an automatic recovery. It will print a warning message and continue with the execution. The recovery attempt may entail changing input variables specified by the user.
2. Critical problems: rflump cannot recover without user intervention. It will print an error message and stop execution.

10.1 General Considerations

f a problem is encountered and it proves difficult to determine the source, the following suggestions may prove helpful:

1. Repeat the run or restart from the last output dump with maximum verbosity level. The additional output printed by activating the maximum verbosity level can indicate possible problem areas, see Sec. **??**
2. Repeat the run with smaller CFL number/time step. Decreasing the CFL number/time step can mitigate the effect of strong transients. It is possible that some computations have to be started with smaller a CFL number/time step to ensure stability.
3. Repeat the run with different flux function. Some flux functions exhibit pathological behavior under certain conditions or do not guarantee preservation of positive-definite quantities. For example, of the flux functions available in RocfluMP, the HLLC scheme often exhibits more robustness in strong transients than the Roe scheme.
4. Repeat the run with a first-order scheme. Failing first-order accurate computations often indicate problems with problem setup.
5. Repeat the run with modified boundary conditions. For example, if problems persist during strong transients at an outflow boundary, it may prove helpful to attempt a new run with supersonic or subsonic conditions.

10.2 Explanations of Warnings

`Inflow detected at outflow boundary!`

`Outflow detected at inflow boundary!`

One or more faces on an inflow/outflow boundary were detected with outflow/inflow, respectively. This is often caused by the precise values of the boundary conditions imposed by the user. For example, too high a value of the static pressure at outflow boundaries can lead to inflow in some conditions. Reverse flow on inflow/outflow boundaries can be a temporary

problem, i.e., the expected flow direction becomes established as the flow develops. In some cases, however, reverse flow on boundaries can indicate a more serious problem, namely a poorly chosen location of boundaries. This may be reflected in the persistence of inflow at an outflow boundary despite lowering the static pressure, the persistence of outflow at an inflow boundary despite increasing the stagnation pressure, or recirculation regions across outflow boundaries. In these cases, it is necessary to reposition the relevant boundary patches.

These warnings will be accompanied by additional output which provides more information about the precise location of the boundary faces with reverse flow. Section **??** describes how this output can be used to visualize the cells affected by reverse flow.

10.3 Explanations of Errors

`Absolute difference in volumes larger than specified limit.`

rflump computes the total volume of the computational domain using two methods. The first method simply sums the volumes of all the cells in a given region. The second method computes the total volume from the boundary and interregion faces. If the two methods disagree by more than a (presently hardcoded) tolerance, it is likely that the interior grid in invalid and the above error message is printed.

`Invalid quantity detected.`

rflump detected solution variables with the value `NaN`. This indicates a serious problem. This is often the result of using a CFL number or a time step which is too large. For coupled simulations, invalid variables may also arise if the geometry becomes heavily deformed. This error message may also indicate problems with the specification of boundary conditions.

`Negative positive-definite quantity detected.`

Negative positive-definite solution variables, such as density, pressure, or temperature were detected. This is often the result of using a CFL number or a time step which is too large. For coupled simulations, negative positive-definite variables may also arise if the geometry becomes heavily deformed.

`Negative volume(s) detected.`

One or more cells with negative volumes were detected. For computations without moving grids, this indicates that the cell or boundary-face connectivity is incorrect. For computations with moving grids, negative volumes indicate that either the boundary deformation is too strong given the grid-motion parameters specified by the user, or the grid quality is poor and deteriorates with grid motion. In the latter case, the grid quality needs to be improved before a new run is started.

`Face sum greater than minimum face area.`

rflump checks whether cells are closed by computing the sum of the face vectors and comparing them against a (presently hardcoded) tolerance. It the sum of the face vectors exceeds the tolerance, it is likely that the cell or boundary-face connectivity is incorrect.

10.4 Other Problems

his section lists problems and suggested remedies for problems which do not lead to warnings or errors.

Cells near moving boundaries become highly stretched.

Highly stretched cells near moving boundaries indicate that the boundary motion is not propagated well enough into the interior. This problem can be remedied by increasing the number of smoothing interations `NITER` or by increasing the smoothing coefficient `SFACT` in the `GRIDMOTION` section of the input file.

10.5 Locating Troublespots

sec:locspots

If problems are encountered, rflump prints additional information to allow the user to determine precisely the locations at which problems occur. For example, consider the following output:

```
Printing location information...
  Global region: 00001
  Cell location information:
     #      Cell   x-coordinate    y-coordinate    z-coordinate   Location
     1        51  0.32081289E-02 -0.83698854E-01 -0.12040992E-01  Global patches:   1   6
     2       300  0.12485188E-01 -0.83750698E-01  0.61473528E-03  Global patches:   1   6
     3       357  0.35957776E-02 -0.83795613E-01 -0.74193030E-02  Global patch:   6
     4      1696  0.11128962E-01 -0.83783063E-01 -0.56762047E-02  Global patches:   1   6
     5      2048  0.10507133E-01 -0.83728027E-01 -0.66900307E-02  Global patches:   1   6
     6      2066  0.87561363E-02 -0.83784932E-01 -0.89096344E-02  Global patches:   1   6
     7      2116  0.47333590E-02 -0.83785244E-01 -0.97767197E-02  Global patch:   6
     8      2574  0.78548965E-03 -0.83765772E-01 -0.12463422E-01  Global patches:   1   6
     9      2995  0.48426506E-02 -0.83799824E-01 -0.90511979E-02  Global patch:   6
    10      6461  0.45099568E-02 -0.83819846E-01 -0.11576659E-01  Interior
Printing location information done.
```

The output lists the cells and the coordinates of the cell centroids along with their locations. The location indicates whether the cell shares one or more faces with boundary patches. In some cases, the above information may suffice because all or a majority of cells lie on a given boundary patch, indicating that the problem originates with the boundary conditions applied to this patch. If the location of the cells is not immediately apparent or scattered around the computational domain, it can be helpful to select individual cells for visualization

using rflupick. This can be useful to determine whether the problems originate in a clusters of cells of poor quality, or along the interface between layers of different cell types.

If the source of the problem is not readily apparent from the additional output, it is often helpful to place probes at the location of the cell centroids before restarting the computation or starting a new computation. Taking the first three cells of above example, the following `PROBE` may be inserted into the `<casename>.inp` file:

```
# PROBE
NUMBER 3
0.32081289E-02 -0.83698854E-01 -0.12040992E-01
0.12485188E-01 -0.83750698E-01  0.61473528E-03
0.35957776E-02 -0.83795613E-01 -0.74193030E-02
#
#
```

The probe files produced by repeating the run can be visualized with TECPLOT (or other packages) and may help in determining the cause of the problem.

References

[1] Abgrall R., Désidéri J.-A., Glowinski R., Mallet M., and Périaux J. (Eds.), *Hypersonic Flows for Reentry Problems*, Vol. III, Proceedings of the INRIA-GAMNI/SMAI Workshop on Hypersonic Flows for Reentry Problems, Part II. Antibes, France, April 15-19, 1991, Springer Verlag.

[2] Batten P., Clarke N., Lambert C., and Causon D.M., *On the Choice of Wavespeeds for the HLLC Riemann Solver*, SIAM J. Sci. Comp., 18(6):1553-1570, 1997.

[3] Ciucci A., Iafrati A., and Schettino A., *Numerical Analysis of Pressure Oscillations in a Duct - Test Case C0*, Technical Report TR-96-102, Centro Italiano Ricerche Aerospaziali, September 1996.

[4] Ringleb F., *Exakte Lösungen der Differentialgleichung einer adiabaten Gasströmung*, ZAMM, 20(4):185-198, 1940.

[5] Roe P.L., *Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes*, J. Comp. Phys., 43:357-372, 1981.

[6] Sod G., *A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws*, J. Comp. Phys., 27:1-31, 1978.