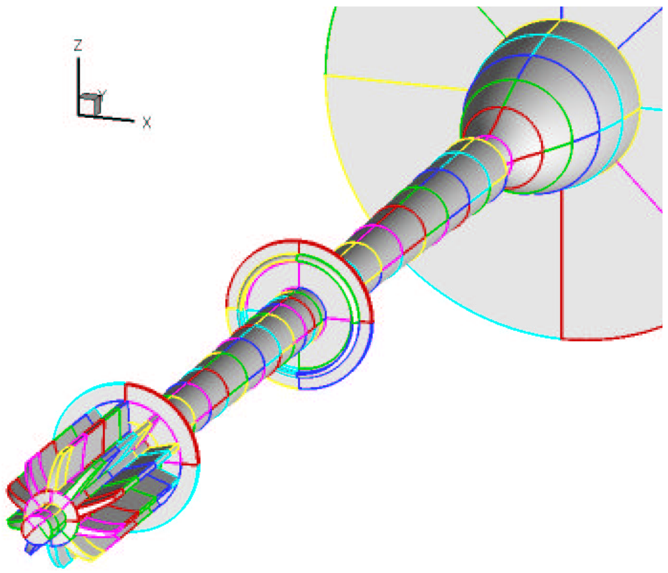
Rocflo Developer's Guide

Bono Wasistho and Jiri Blazek

Center for Simulation of Advanced Rockets

University of Illinois at Urbana-Champaign

Urbana, IL 61801



December 9, 2005

Contents

Chapter 1

Notation

c	speed of sound
c_p	heat coefficient at constant pressure
E	total energy per unit mass
E^P	energy source from particles
E^S	energy source from smoke
E^{SGS}	subgrid-scale energy flux
$\vec{f}_e$	vector of external volume forces
f^P	momentum source from particles
f^S	momentum source from smoke
$\vec{F}_c$	vector of convective fluxes
$\vec{F}_v$	vector of viscous fluxes
H	total (stagnation) enthalpy
i, j, k	node indices
I, J, K	cell indices
k	thermal conductivity coefficient
m^P	mass source from particles
$\vec{n}$	unit normal vector
n_x, n_y, n_z	components of unit normal vector in $x-, y-, z-$direction
p	static pressure
$\dot{q}_h$	heat source
$\vec{Q}$	source term
R	specific gas constant
$\vec{S}$	surface vector ($= \vec{n}\, dS$)
dS	surface element
ΔS	face area
t	time
Δt	time step
u, v, w	Cartesian velocity components
$\vec{v}$	velocity vector

V	contravariant velocity
$\vec{W}$	vector of conservative variables
x, y, z	Cartesian coordinates
α	angle of attack, inlet angle
α_m	coefficient of the Runge-Kutta scheme (in stage m)
β_m	blending coefficient (in stage m of the Runge-Kutta scheme)
γ	ratio of specific heat coefficients at constant pressure and volume
$\hat{\Lambda}_c$	spectral radius of convective flux Jacobian
$\hat{\Lambda}_v$	spectral radius of viscous flux Jacobian
μ	dynamic viscosity coefficient
ρ	density
τ	viscous stress
Ω	control volume
$\partial\Omega$	boundary of a control volume

Chapter 2

Introduction

Rocflo is a general purpose, structured, multiblock, 3-D flow solver for the solution of the Euler or the Navier-Stokes equations. Rocflo contains plug-ins for the following physical modules:

- Turbulence (LES, DES and RANS models) – *Rocturb*
- Lagrangian particles (burning Al droplets) – *Rocpart*
- Eulerian particles (smoke) – *Rocsmoke*
- Chemical species – *Rocspecies*
- Radiation – *Rocrad*

In addition, it also contains plug-ins for specific periodic flow cases, called *Rocperi*.

The goal of this manual is to enable code developers to modify or to extend the core solver, as well as to add physical modules like for turbulence, species, etc. The manual provides an overview of the numerical methodologies and of the structure of the code. Description related to the compilation and the execution of the flow solver and related utility programs can be found in Rocflo User's Guide and is not repeated in here. A more throughout discussion of the numerical schemes can be found in Ref. [1].

Chapter 3

Features and Algorithms

Rocflo employs cell-centered, finite-volume type of spatial discretization. It utilizes the Arbitrary Lagrangian-Eulerian (ALE) method for dynamic and/or moving grids. The code is implemented in Fortran-90 using its features for user defined data types, dynamic memory allocation, pointers, etc. Rocflo is applicable to a large variety of flow problems including external and internal flows, in a wide range of Mach numbers from nearly incompressible to high-speed compressible flows.

Rocflo has currently the following numerical features:

- spatial discretization:
  - 2nd-order central scheme with scalar artificial dissipation,
  - 2nd-order Roe upwind scheme,
- temporal discretization:
  - explicit, 5-stage hybrid scheme with local time stepping and implicit residual smoothing,
  - explicit, 4-stage classical Runge-Kutta scheme for unsteady flows.
  - dual time-stepping, for unsteady flows allowing large time steps.

In order to accelerate convergence to the steady state, Rocflo has the capability of grid sequencing (i.e., the solver is started on a coarse grid and after a number of iterations it proceeds on the next finer grid). The code is parallelized (blockwise) using MPI.

Rocflo has the capability to compute the thrust of a rocket motor by intergrating pressure and momentum over a specified plane (nozzle exit).

3.1 Governing Equations

Equations being solved are the time-dependent Navier-Stokes equations in integral form on moving and/or deforming grid

$$\frac{\partial}{\partial t}\int_{\Omega} \vec{W}\, d\Omega + \oint_{\partial\Omega} (\vec{F}_c - \vec{F}_v)\, dS = \int_{\Omega} \vec{Q}\, d\Omega\,. \tag{3.1}$$

The vector of the conservative variables $\vec{W}$ consists in three dimensions of the following components

$$\vec{W} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}. \tag{3.2}$$

Further components have to be added for the transport equations of the species and for turbulence equations (if a RANS model is used). The vector of convective fluxes reads

$$\vec{F}_c = \begin{bmatrix} \rho V \\ \rho u V + n_x p \\ \rho v V + n_y p \\ \rho w V + n_z p \\ \rho H V + V_t p \end{bmatrix} \tag{3.3}$$

with the contravariant velocity relative to the grid motion

$$V = n_x u + n_y v + n_z w - V_t\,, \tag{3.4}$$

and with the contravariant velocity of the face of the control volume

$$V_t = n_x \frac{\partial x}{\partial t} + n_y \frac{\partial y}{\partial t} + n_z \frac{\partial z}{\partial t}\,. \tag{3.5}$$

The vector of the viscous fluxes can be written as

$$\vec{F}_v = \begin{bmatrix} 0 \\ n_x\tau_{xx} + n_y\tau_{xy} + n_z\tau_{xz} \\ n_x\tau_{yx} + n_y\tau_{yy} + n_z\tau_{yz} \\ n_x\tau_{zx} + n_y\tau_{zy} + n_z\tau_{zz} \\ n_x\Theta_x + n_y\Theta_y + n_z\Theta_z \end{bmatrix}, \tag{3.6}$$

where

$$\begin{aligned} \Theta_x &= u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + k\frac{\partial T}{\partial x} + E_x^{SGS} \\ \Theta_y &= u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + k\frac{\partial T}{\partial y} + E_y^{SGS} \\ \Theta_z &= u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + k\frac{\partial T}{\partial z} + E_z^{SGS} \end{aligned} \tag{3.7}$$

are the terms describing the work of viscous stresses and the heat conduction in the fluid. In the case of LES, the viscous stresses read

$$\tau_{ij} = 2\mu\, S_{i,j} - \left(\frac{2\mu}{3}\right)\frac{\partial v_k}{\partial x_k}\delta_{ij} + \tau_{ij}^{SGS} \tag{3.8}$$

with $S_{i,j}$ being the components of the strain rate tensor

$$S_{i,j} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) \tag{3.9}$$

and τ_{ij}^{SGS} denoting the subgrid stresses. In the case of RANS equations, the subgrid terms E^{SGS} in Eq. (3.7) and τ_{ij}^{SGS} in Eq. (3.8) are omitted. Instead, the dynamic viscosity and the thermal conductivity are splitted into a laminar and a turbulent part, i.e.,

$$\mu = \mu_L + \mu_T \tag{3.10}$$

and

$$k = k_L + k_T = c_p\left(\frac{\mu_L}{Pr_L} + \frac{\mu_T}{Pr_T}\right), \tag{3.11}$$

where c_p stands for the specific heat coefficient at constant pressure, and Pr is the Prandl number.

The source term is given by

$$\vec{Q} = \begin{bmatrix} m^P \\ \rho f_{e,x} + f_x^P + f_x^S \\ \rho f_{e,y} + f_y^P + f_y^S \\ \rho f_{e,z} + f_z^P + f_z^S \\ \rho\vec{f}_e\cdot\vec{v} + \dot{q}_h + E^P + E^S \end{bmatrix}, \tag{3.12}$$

where m^P, f^P, f^S, E^P, and E^S represent the source terms introduced by the particle and smoke modeling. The vector of external volume forces reads

$$\vec{f}_e = \vec{g} - \vec{a} \tag{3.13}$$

with $\vec{g}$ being the gravitational acceleration and $\vec{a}$ the acceleration of the rocket.

Finally, the Geometric Conservation Law (GCL) must be satisfied in order to avoid errors induced by deformation of control volumes. The integral form of the GCL reads [2]

$$\frac{\partial}{\partial t}\int_{\Omega} d\Omega - \oint_{\partial\Omega} V_t\, dS = 0\,. \tag{3.14}$$

The GCL results from the requirement that the computation of the control volumes or of the grid velocities must be performed in such a way that the resulting numerical scheme preserves the state of a uniform flow, independent of the deformation of the grid.

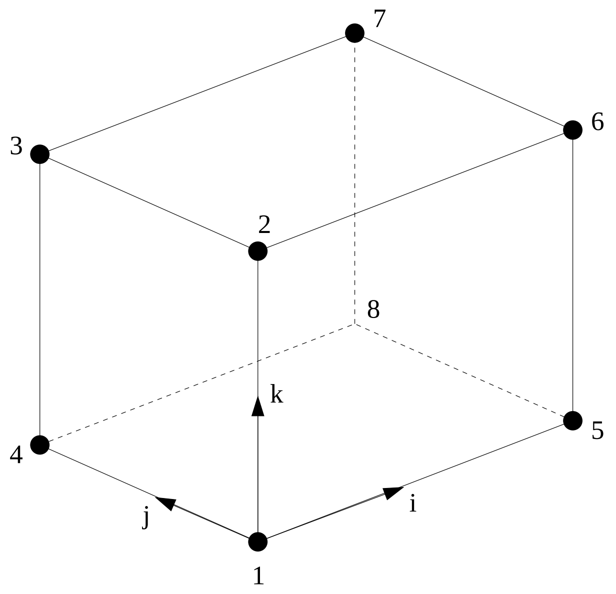


Figure 3.1: Numbering of the corners of the computational space.

3.2 Finite-Volume Scheme

Rocflo is based on a 3-D cell-centered, finite-volume type of spatial discretization. The control volume are hexahedra. The numbering of the corners and the edges of the control volume is depicted in Fig. 3.1 and 3.2. Corner 1 in Fig. 3.1 is associated with the grid node (i, j, k), corner 5 with $(i + 1, j, k)$, etc. The centroid of the cell is denoted as (I, J, K).

The face vectors $\vec{S} = \vec{n}\,\Delta S$ in Eq. (3.1) are computed and stored only for three faces of the control volume:

- in i-direction at 1-2-3-4;
- in j-direction at 1-5-6-2;
- in k-direction at 1-4-8-5.

The three face vectors point outward of the control volume. They are most conveniently computed using the Gauss's for the area of a quadrilateral. Thus, for example, for the face 1-2-3-4 in Fig. 3.1 the face vector $\vec{S}_I$ results from

$$\vec{S}_I = \frac{1}{2}\begin{bmatrix} \Delta z_A\,\Delta y_B - \Delta y_A\,\Delta z_B \\ \Delta x_A\,\Delta z_B - \Delta z_A\,\Delta x_B \\ \Delta y_A\,\Delta x_B - \Delta x_A\,\Delta y_B \end{bmatrix} \tag{3.15}$$

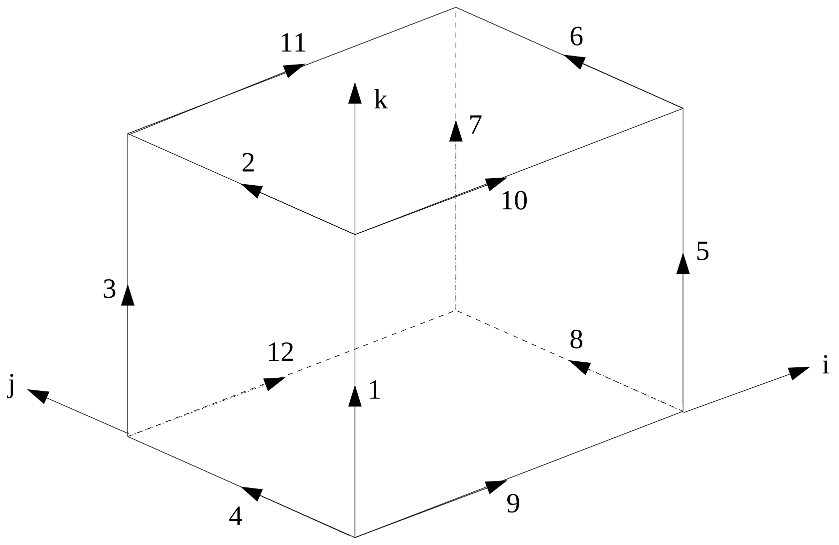


Figure 3.2: Numbering and directions of the edges of the computational space.

with the differences

$$\begin{aligned} \Delta x_A &= x_3 - x_1\,, \quad \Delta x_B = x_2 - x_4\,, \\ \Delta y_A &= y_3 - y_1\,, \quad \Delta y_B = y_2 - y_4\,, \\ \Delta z_A &= z_3 - z_1\,, \quad \Delta z_B = z_2 - z_4\,. \end{aligned} \tag{3.16}$$

The size of the control volume Ω can be computed using the divergence theorem [3]. This relates the volume integral of the divergence of some vector quantity to its surface integral. As a result, the volume of a hexahedra becomes

$$\Omega_{I,J,K} = \frac{1}{3} \sum_{m=1}^{m=6} (\vec{r}_{\text{mid}} \cdot \vec{S})_m\,, \tag{3.17}$$

where m denotes the faces of the control volume and $\vec{r}_{\text{mid}}$ the midpoints of the faces.

3.3 Inviscid Fluxes – Central Scheme

One of the spatial discretization methodologies implemented is the central scheme with scalar artificial dissipation [4]. The convective fluxes $\vec{F}_c$ through a face of the control volume are approximated using an average of the conserved variables. Artificial dissipation is then added to the central fluxes for stability. Thus, the total convective flux at the face $(I + 1/2, J, K)$ of the control volume reads

$$(\vec{F}_c\, \Delta S)_{I+1/2,J,K} \approx \vec{F}_c(\vec{W}_{I+1/2,J,K})\, \Delta S_{I+1/2,J,K} \,-\, \vec{D}_{I+1/2,J,K}\,, \tag{3.18}$$

where the indices denote cell centroids. The flow variables are averaged as

$$\vec{W}_{I+1/2,J,K} = \frac{1}{2}\left(\vec{W}_{I,J,K} + \vec{W}_{I+1,J,K}\right) . \tag{3.19}$$

The artificial dissipation consists of a blend of adaptive second- and fourth-order differences which result from the sum of first- and third-order difference operators

$$\begin{aligned} \vec{D}_{I+1/2} = \hat{\Lambda}^S_{I+1/2} \Big[& \epsilon^{(2)}_{I+1/2} (\vec{W}_{I+1} - \vec{W}_I) \\ & - \epsilon^{(4)}_{I+1/2} (\vec{W}_{I+2} - 3\vec{W}_{I+1} + 3\vec{W}_I - \vec{W}_{I-1}) \Big] . \end{aligned} \tag{3.20}$$

The dissipation is scaled by a directionally dependent sum of the spectral radii of the convective flux Jacobians [6], [7]

$$\hat{\Lambda}^S_{I+1/2} = (\hat{\Lambda}^I_c)_{I+1/2} + \max\left[(\hat{\Lambda}^J_c)_{I+1/2},\, (\hat{\Lambda}^K_c)_{I+1/2}\right] . \tag{3.21}$$

The spectral radius at the cell face $(I + 1/2)$, e.g., in I-direction (represented by the superscript I), results from the average

$$(\hat{\Lambda}^I_c)_{I+1/2} = \frac{1}{2}\left[(\hat{\Lambda}^I_c)_I + (\hat{\Lambda}^I_c)_{I+1}\right] . \tag{3.22}$$

It is evaluated using the formula

$$\hat{\Lambda}_c = (\,|V| + c)\,\Delta S\,, \tag{3.23}$$

where V stands for the contravariant velocity (3.4) and c for the speed of sound, respectively.

A pressure-based sensor is used to switch off the fourth-order differences at shocks, where they would lead to strong oscillation of the solution. The sensor also switches off the second-order differences in smooth parts of the flow field, in order to reduce the dissipation to the lowest possible level. Herewith, the coefficients $\epsilon^{(2)}$ and $\epsilon^{(4)}$ in Eq. (3.20) are defined as

$$\begin{aligned} \epsilon^{(2)}_{I+1/2} &= k^{(2)} \max(\Upsilon_I, \Upsilon_{I+1}) \\ \epsilon^{(4)}_{I+1/2} &= \max\left[0,\, (k^{(4)} - \epsilon^{(2)}_{I+1/2})\right] . \end{aligned} \tag{3.24}$$

Typical values of the parameters are $k^{(2)} = 1/2$ and $1/128 \le k^{(4)} \le 1/64$. The standard pressure sensor is given by

$$\Upsilon_I = \frac{|\,p_{I+1} - 2p_I + p_{I-1}|}{p_{I+1} + 2p_I + p_{I-1}} . \tag{3.25}$$

Another form of the pressure sensor is based on the TVD (Total Variation Diminishing) principle [8]. It reads

$$\Upsilon_I = \frac{|\,p_{I+1} - 2p_I + p_{I-1}|}{(1-\omega)P_{TVD} + \omega P} , \tag{3.26}$$

where

$$\begin{aligned} P_{TVD} &= |\,p_{I+1} - p_I| + |\,p_I - p_{I-1}| \\ P &= |\,p_{I+1} + 2p_I + p_{I-1}|\,. \end{aligned} \tag{3.27}$$

Typical value of the parameter ω is $1/2$.

3.4 Inviscid Fluxes – Upwind Scheme

The second spatial discretization method implemented in Rocflo is the flux-difference splitting scheme of Roe [9]. This scheme is often applied because of its high accuracy for boundary layer flows and a good shock resolution.

The convective fluxes $\vec{F}_c$ through a face of the control volume are evaluated as

$$(\vec{F}_c)_{I+1/2} = \frac{1}{2}\left[\vec{F}_c(\vec{W}_R) + \vec{F}_c(\vec{W}_L) - |\bar{A}_{Roe}|_{I+1/2}\,(\vec{W}_R - \vec{W}_L)\right]. \tag{3.28}$$

The product of $|\bar{A}_{Roe}|$ and the difference of the left and right state can be computed as follows

$$|\bar{A}_{Roe}|(\vec{W}_R - \vec{W}_L) = |\Delta\vec{F}_1| + |\Delta\vec{F}_{2,3,4}| + |\Delta\vec{F}_5|\,, \tag{3.29}$$

where

$$|\Delta\vec{F}_1| = |\tilde{V} - \tilde{c}\,|\left(\frac{\Delta p - \tilde{\rho}\tilde{c}\Delta V}{2\tilde{c}^2}\right)\begin{bmatrix} 1 \\ \tilde{u} - \tilde{c}n_x \\ \tilde{v} - \tilde{c}n_y \\ \tilde{w} - \tilde{c}n_z \\ \tilde{H} - \tilde{c}\tilde{V} \end{bmatrix} \tag{3.30}$$

$$\begin{aligned} |\Delta\vec{F}_{2,3,4}| = |\tilde{V}| \Bigg\{ & \left(\Delta\rho - \frac{\Delta p}{\tilde{c}^2}\right)\begin{bmatrix} 1 \\ \tilde{u} \\ \tilde{v} \\ \tilde{w} \\ \tilde{q}^2/2 \end{bmatrix} \\ & + \tilde{\rho}\begin{bmatrix} 0 \\ \Delta u - \Delta V n_x \\ \Delta v - \Delta V n_y \\ \Delta w - \Delta V n_z \\ \tilde{u}\Delta u + \tilde{v}\Delta v + \tilde{w}\Delta w - \tilde{V}\Delta V \end{bmatrix}\Bigg\} \end{aligned} \tag{3.31}$$

$$|\Delta\vec{F}_5| = |\tilde{V} + \tilde{c}\,|\left(\frac{\Delta p + \tilde{\rho}\tilde{c}\Delta V}{2\tilde{c}^2}\right)\begin{bmatrix} 1 \\ \tilde{u} + \tilde{c}n_x \\ \tilde{v} + \tilde{c}n_y \\ \tilde{w} + \tilde{c}n_z \\ \tilde{H} + \tilde{c}\tilde{V} \end{bmatrix}. \tag{3.32}$$

The jump condition is defined as $\Delta(\bullet) = (\bullet)_R - (\bullet)_L$ and the Roe-averaged variables are given by

$$\begin{aligned}
\tilde{u}_{I+1/2} &= \frac{u_L\sqrt{\rho_L} + u_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\tilde{v}_{I+1/2} &= \frac{v_L\sqrt{\rho_L} + v_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\tilde{w}_{I+1/2} &= \frac{w_L\sqrt{\rho_L} + w_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\tilde{H}_{I+1/2} &= \frac{H_L\sqrt{\rho_L} + H_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\tilde{c}_{I+1/2} &= \sqrt{(\gamma-1)\left(\tilde{H} - \frac{\tilde{u}^2 + \tilde{v}^2 + \tilde{w}^2}{2}\right)_{I+1/2}} \\
\tilde{V}_{I+1/2} &= \tilde{u}_{I+1/2}\, n_x + \tilde{v}_{I+1/2}\, n_y + \tilde{w}_{I+1/2}\, n_z\,.
\end{aligned} \tag{3.33}$$

The left and the right state are determined using the MUSCL (Monotone Upstream-Centred Schemes for Conservation Laws) approach [10]

$$\begin{aligned}
U_R &= U_{I+1} - \frac{\epsilon}{4}\left[(1+\hat{\kappa})\Delta_- + (1-\hat{\kappa})\Delta_+\right] U_{I+1} \\
U_L &= U_I \quad + \frac{\epsilon}{4}\left[(1+\hat{\kappa})\Delta_+ + (1-\hat{\kappa})\Delta_-\right] U_I\,.
\end{aligned} \tag{3.34}$$

The forward (Δ_+) and the backward (Δ_-) difference operators are defined as

$$\begin{aligned}
\Delta_+ U_I &= U_{I+1} - U_I \\
\Delta_- U_I &= U_I - U_{I-1}\,.
\end{aligned} \tag{3.35}$$

All higher-order schemes ($\hat{\kappa} = -1$, $\hat{\kappa} = 0$, and $\hat{\kappa} = 1/3$) have to be supplemented by limiters, if the flow field contains any discontinuities. The parameter ϵ can be set equal to zero to obtain a first-order accurate upwind discretisation.

Because of the formulation in Eq. (3.28), Roe's approximate Riemann solver will produce an unphysical expansion shock in the case of stationary expansion, for which $(\vec{F}_c)_L = (\vec{F}_c)_R$ but $\vec{W}_L \neq \vec{W}_R$. Furthermore, the so-called "carbuncle phenomenon" may occur, where a perturbation grows ahead of a strong bow shock along the stagnation line. The underlying difficulty is that the original scheme does not recognise the sonic point. In order to solve this problem, the modulus of the eigenvalues $|\Lambda_c| = |\tilde{V} \pm \tilde{c}|$ can be modified using Harten's entropy correction [11], [12]

$$|\Lambda_c| = \begin{cases} |\Lambda_c| & \text{if } |\Lambda_c| > \delta \\ \dfrac{\Lambda_c^2 + \delta^2}{2\delta} & \text{if } |\Lambda_c| \leq \delta\,, \end{cases} \tag{3.36}$$

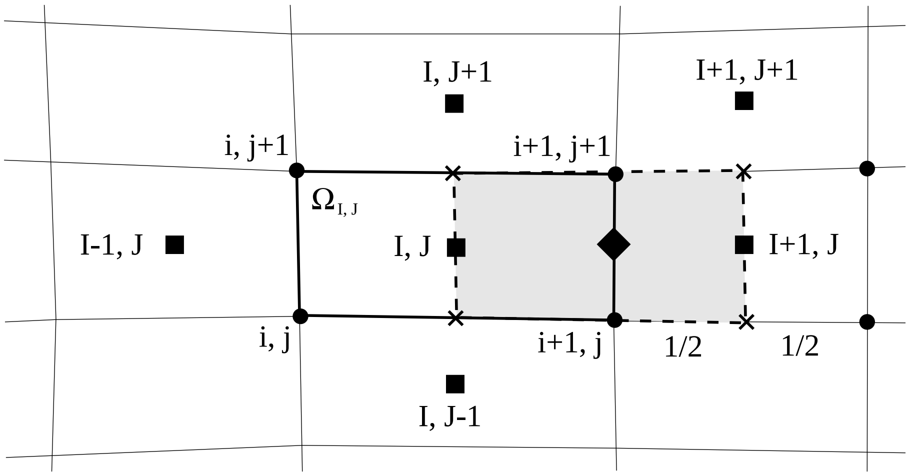


Figure 3.3: Auxiliary control volume Ω' (filled) for evaluation of first derivatives in two dimensions. The diamond symbol denotes location where first derivatives are to be evaluated.

where δ is a small value, which can be conveniently set equal to some fraction (e.g., 1/10) of the local speed of sound. In order to prevent the linear waves $|\Delta\vec{F}_{2,3,4}|$ from disappearing for $\tilde{V} \rightarrow 0$ (e.g., at stagnation points or for grid-aligned flow), the above modification can also be applied to $|\tilde{V}|$.

3.5 Viscous Fluxes

The viscous fluxes $\vec{F}_v$ in the governing equations (3.1) are evaluated from variables averaged at the faces of the control volume. This is in line with the elliptic nature of the viscous fluxes. Thus, values of the velocity components (u, v, w), the dynamic viscosity μ, and of the heat conduction coefficient k, which are required for the computation of the viscous terms (3.6), (3.7) and of the stresses (3.8), are simply averaged at the respective face. Hence, the values at the face $(I+1/2)$ of the control volume result from

$$U_{I+1/2} = \frac{1}{2}(U_I + U_{I+1})\,, \tag{3.37}$$

where U is any of the above flow variables.

The remaining task is the evaluation of the first derivatives (gradients) of the velocity components in Eq. (3.8) and of temperature in Eq. (3.7). The gradients are computed here using the Gauss' theorem. For this purpose, an auxiliary control volume is defined as shown in Fig. 3.3. The gradient of a variable U at the face centroid is obtained from

$$\frac{\partial U}{\partial x} = \frac{1}{\Omega'}\int_{\partial\Omega'} U\, dS'_x \approx \frac{1}{\Omega'}\sum_{m=1}^{m=6} U_m\, S'_{x,m}\,, \tag{3.38}$$

and similarly for the remaining coordinate directions. The volume Ω' and the components of the face vector $\vec{S}'_m$ can be computed as already presented in Section 3.2. However, in Rocflo they are averaged from the cell geometries. The face values U_m are obtained either directly as cell-centred values, or by averaging like on the upper and the lower face of Ω' in Fig. 3.3.

3.6 Time Stepping

Explicit and implicit time-stepping are available for the integration of the governing equations (3.1) in time. Two different explicit schemes are implemented. The first one is a 5-stage hybrid scheme [6], [13]. It is applied in the case of stationary flows. It is accelerated by local time stepping and implicit residual smoothing [14]. The second explicit time-stepping scheme is the classical 4-stage Runge-Kutta scheme, which is 4th-order accurate in time.

A distinguishing feature of the hybrid multistage scheme is that the viscous fluxes and the dissipation are not re-evaluated at each stage. This reduces the amount of computational work. Additionally, the dissipation terms from different stages can be blended to increase stability of the scheme. The (5,3)-scheme, which is programmed in Rocflo, reads

$$\begin{aligned}
\vec{W}_I^{(0)} &= \vec{W}_I^n \\
\vec{W}_I^{(1)} &= \vec{W}_I^{(0)} - \alpha_1 \frac{\Delta t_I}{\Omega_I}\left[\vec{R}_c^{(0)} - \vec{R}_d^{(0)}\right]_I \\
\vec{W}_I^{(2)} &= \vec{W}_I^{(0)} - \alpha_2 \frac{\Delta t_I}{\Omega_I}\left[\vec{R}_c^{(1)} - \vec{R}_d^{(0)}\right]_I \\
\vec{W}_I^{(3)} &= \vec{W}_I^{(0)} - \alpha_3 \frac{\Delta t_I}{\Omega_I}\left[\vec{R}_c^{(2)} - \vec{R}_d^{(2,0)}\right]_I \\
\vec{W}_I^{(4)} &= \vec{W}_I^{(0)} - \alpha_4 \frac{\Delta t_I}{\Omega_I}\left[\vec{R}_c^{(3)} - \vec{R}_d^{(2,0)}\right]_I \\
\vec{W}_I^{n+1} &= \vec{W}_I^{(0)} - \alpha_5 \frac{\Delta t_I}{\Omega_I}\left[\vec{R}_c^{(4)} - \vec{R}_d^{(4,2)}\right]_I ,
\end{aligned} \tag{3.39}$$

where

$$\begin{aligned}
\vec{R}_d^{(2,0)} &= \beta_3 \vec{R}_d^{(2)} + (1-\beta_3)\vec{R}_d^{(0)} \\
\vec{R}_d^{(4,2)} &= \beta_5 \vec{R}_d^{(4)} + (1-\beta_5)\vec{R}_d^{(2,0)} .
\end{aligned} \tag{3.40}$$

As indicated in Eq. (3.39), the scheme has 5 stages and the dissipation and diffusion terms $\vec{R}_d$ are evaluated at each odd stage. The stage coefficients α_m and the blending coefficients β_m in the above relations (3.39), (3.40) are given in Table 3.1.

Besides the explicit time-stepping, an implicit scheme is also implemented in the form of dual time-stepping for explicit multistage scheme. The method is based on an explicit multistage scheme accelerated by local time-stepping and multigrid [5]. The multigrid acceleration is however still under construction, only the simple form of dual time-stepping is

	central scheme $\sigma = 3.6$		upwind scheme $\sigma = 2.0$	
stage	α	β	α	β
1	0.2500	1.00	0.2742	1.00
2	0.1667	0.00	0.2067	0.00
3	0.3750	0.56	0.5020	0.56
4	0.5000	0.00	0.5142	0.00
5	1.0000	0.44	1.0000	0.44

Table 3.1: Hybrid multistage scheme: optimised stage (α) and blending (β) coefficients, as well as CFL numbers (σ) for central and upwind spatial discretisations.

readily used currently. In dual time-stepping, the solution vector is modified as follow, where t^* denotes a pseudo-time, β a parameter to stabilize the time-stepping scheme (its default value is 2), and R^* a modified unsteady residual defined as,

$$\vec{R}_I^*(\vec{W}_I^*) = R_I(\vec{W}_I^*) + \frac{3}{2\Delta t}(\Omega\overline{M})_I^{n+1}\vec{W}_I^* - \vec{Q}_I^*. \tag{3.41}$$

3.7 Boundary Conditions

Rocflo provides a variety of boundary conditions, which are applicable in internal and external flows. These are:

- Solid wall
- Injection
- Inflow / Outflow
- Far field
- Symmetry
- Block interface (conforming or not)
- Rotational or translational periodicity

The formulation of each of the above boundary conditions will be described in the next subsections.

3.7.1 Solid Wall

In the case of an inviscid flow, the fluid slips over the surface. Since there is no friction force, the velocity vector must be parallel to the surface. This is equivalent to the condition that there is no flow normal to the surface, i.e.,

$$\vec{v} \cdot \vec{n} = 0 \qquad \text{at the surface}\,, \tag{3.42}$$

where $\vec{n}$ denotes the unit normal vector at the surface. Hence, the contravariant velocity V (Eq. (3.4)) is zero at the wall. Consequently, the vector of convective fluxes Eq. (3.3) reduces to the pressure term alone, i.e,

$$(\vec{F}_c)_w = \begin{bmatrix} 0 \\ n_x\, p_w \\ n_y\, p_w \\ n_z\, p_w \\ 0 \end{bmatrix} \tag{3.43}$$

with p_w being the wall pressure. The wall pressure is linearly interpolated from the values at two interior cells

$$p_w = \frac{1}{2}(3p_2 - p_3)\,, \tag{3.44}$$

where 2 denotes the boundary cell and 3 the next cell layer.

For a viscous fluid which passes a solid wall, the relative velocity between the surface and the fluid directly at the surface is assumed to be zero. In the case of a stationary wall surface, the Cartesian velocity components become

$$u = v = w = 0 \qquad \text{at the surface.} \tag{3.45}$$

If the wall is adiabatic (i.e. there is no heat flux through the wall), the noslip boundary condition is implemented by copying the velocity components from the interior to the dummy cells and inverting the sign. Hence,

$$u_1 = -u_2\,, \quad v_1 = -v_2\,, \quad w_1 = -w_2\,. \tag{3.46}$$

Scalar values like density or total energy are just copied from the interior. The convective fluxes are evaluated as indicated in Eq. (3.43).

If the wall temperature is given, the velocity components are still reversed as in Eq. (3.46). The temperature is linearly extrapolated from the interior field by using the specified wall temperature. Since the pressure gradient normal to the wall is zero, the pressure in the boundary element is prescribed also in the dummy cells (i.e., $p_1 = p_2$). The density and the total energy in the dummy cells are evaluated from the interpolated values.

3.7.2 Injection

The injection velocity and other flow variables are computed from the given mass flow rate $\dot{m}$ and the surface temperature T_s. The convective fluxes are evaluated using Eq. (3.3) with the contravariant velocity being equal to the injection velocity, i.e.

$$V = -\frac{\dot{m}}{\rho_s} \quad \text{and} \quad V_t = 0\,. \tag{3.47}$$

The density ρ_s in Eq. (3.47) is a function of T_s and of the pressure at the boundary cell. Values at the dummy cells are obtained by extrapolation from the interior together with the relative velocity of the surface (i.e. including V_t).

3.7.3 Inflow_TotAng

In the case of subsonic inflow, total pressure, total temperature, and two flow angles are prescribed. One characteristic variable has to be interpolated from the interior of the flow domain. One possibility is to employ the outgoing Riemann invariant [15], which is defined as

$$\mathcal{R}^- = \vec{v}_d \cdot \vec{n} - \frac{2\,c_d}{\gamma - 1}\,, \tag{3.48}$$

where the index d denotes the state inside the domain. The Riemann invariant is used to determine either the absolute velocity or the the speed of sound at the boundary. In practice, it was found that selecting the speed of sound leads to a more stable scheme, particularly for low Mach-number flows. Therefore,

$$c_b = \frac{-\mathcal{R}^-(\gamma-1)}{(\gamma-1)\cos^2\theta + 2}\left\{1 + \cos\theta\sqrt{\frac{[(\gamma-1)\cos^2\theta + 2]c_0^2}{(\gamma-1)(\mathcal{R}^-)^2} - \frac{\gamma-1}{2}}\right\} \tag{3.49}$$

with θ being the flow angle relative to the boundary, and c_0 denoting the stagnation speed of sound. Hence,

$$\cos\theta = -\frac{\vec{v}_d \cdot \vec{n}}{\|\vec{v}_d\|_2} \tag{3.50}$$

and

$$c_0^2 = c_d^2 + \frac{\gamma-1}{2}\|\vec{v}_d\|_2^2\,, \tag{3.51}$$

where $\|\vec{v}_d\|_2$ denotes the total velocity at an interior point d. The unit normal vector $\vec{n}$ in Eq. (3.50) was assumed to point outwards of the flow domain.

Quantities like static temperature, pressure, density, or the absolute velocity at the boundary are evaluated as follows

$$\begin{aligned} T_b &= T_0\left(\frac{c_b^2}{c_0^2}\right), \quad p_b = p_0\left(\frac{T_b}{T_0}\right)^{\gamma/(\gamma-1)} \\ \rho_b &= \frac{p_b}{RT_b}, \quad \|\vec{v}_b\|_2 = \sqrt{2\,c_p(T_0 - T_b)}\,, \end{aligned} \tag{3.52}$$

where T_0 and p_0 are the given values of total temperature and pressure, R and c_p represent the specific gas constant and the heat coefficient at constant pressure, respectively. The velocity components at the inlet are obtained by decomposing $\|\vec{v}_b\|_2$ according to the two prescribed flow angles. Flow variables in the dummy cells are obtained by linear extrapolation of the states at the boundary and at the interior point d.

In case the inflow is supersonic, all flow variables are prescribed by setting the values in the dummy cells. The implementation employs T_0, p_0, flow angles and a given Mach number to calculate all conservative and dependent variables.

3.7.4 Inflow_VelTemp

In the case of subsonic inflow, inlet velocity components ($u(\mathbf{x}_{inlet}$, $v(\mathbf{x}_{inlet}$, $w(\mathbf{x}_{inlet})$, and inlet temperature are prescribed. The pressure is found from a one dimensional characteristic wave extrapolation [16],

$$\frac{\partial p}{\partial x_n} - \rho c \frac{\partial u_n}{\partial x_n} = 0, \tag{3.53}$$

where u_n and x_n denote the velocity and coordinate normal to the inlet plane, ρ and c the local speed of sound at the first interior cell from the inlet plate, and the derivatives $\partial p/\partial x_n$ are taken using the pressure and velocity difference at the interior cell and dummy cell adjacent to the inlet plane. The velocity components and temperature at the first dummy layer are prescribed. From the computed pressure, and the prescribed velocities and temperature, all the conservative variables at the dummy layer adjacent to the inlet plane can be derived. The conservative variables at dummy layers further away from the plane are obtained by linear extrapolation.

In case the inflow is supersonic, all flow variables are prescribed by setting the values in the dummy cells. The implementation employs u, v, w, T, and$p$, to calculate all conservative and dependent variables.

If desired, turbulence fluctuations can be recycled to obtained an unsteady turbulent inflow. For this purpose, fluctuations are added to the mean inlet velocity, temperature and pressure obtained above. The fluctuations are computed from,

$$\mathbf{q'}(\mathbf{x}_f) = \mathbf{q}(\mathbf{x}_f) - < \mathbf{q}(\mathbf{x}_f) >, \tag{3.54}$$

where $\mathbf{q}$ is a vector containing velocity components and temperature, $\mathbf{x}_f$ the coordinate vector of a numerically determined location downstream of the inlet boundary where the fluctuations are extracted, the superscript (') denotes the fluctuations and $< . >$ the time averaged of the quantities to which they apply. These fluctuations are added to the prescribed inlet velocity and temperature $\mathbf{q}_b$, to generate unsteady flow at the inflow location $\mathbf{x}_i$,

$$\mathbf{q}(\mathbf{x}_i) = \mathbf{q}_b + \mathbf{q'}(\mathbf{x}_f) + (\mathbf{q}_b - < \mathbf{q}(\mathbf{x}_i) >). \tag{3.55}$$

The last term above is to converge the time averaged inflow to the prescribed mean inflow profiles.

3.7.5 Inflow_VelPress

In the case of subsonic flow, similar procedure as that in Inflow_VelTemp is followed. The difference is that the pressure is prescribed instead of temperature. The temperature is the computed from the caracteristics extrapolation [16],

$$\frac{\partial T}{\partial x_n} - \frac{c}{R}\frac{\partial u_n}{\partial x_n} = 0, \tag{3.56}$$

where R is the gas constant. The procedures for supersonic inflow and inlet turbulence generation are the same as that in Inflow_VelTemp, with $\mathbf{q}$ representing a vector containing velocity components and pressure, instead of temperature.

3.7.6 Outflow

In the case of subsonic flow, the static pressure is computed using a partially non-reflecting condition [16],

$$\frac{\partial p}{\partial t} - \rho c\frac{\partial u_n}{\partial t} + \beta(p - p_\infty) = 0. \tag{3.57}$$

The numerical implementation of this expression reads,

$$p_b = p^n + \rho^n c^n \mathbf{n}.(\mathbf{u}^n - \mathbf{u}^{n-1}) - dt\beta(p^n - p_\infty), \tag{3.58}$$

where p_b is the computed pressure at the first dummy layer, p_∞ the prescribed ambient pressure, and dt the local time step. The quantity p, ρ, c and $\mathbf{u}$ are the pressure, densty, speed of sound and velocity vector taken at the interior cells adjacent to the outflow plane, and $\mathbf{n}$ is the unit vector normal to the plane, directed outwards (leaving the computational domain). The superscript n denotes the latest time or stage within a multi-stage time stepping, and $n-1$ the previous time or stage. Setting the non-reflecting parameter $\beta = 0$ yields a fully non-reflecting condition, while setting it to $\beta > 0$ implies a partially non-reflecting condition. The higher the value of β the closer p_b fluctuates around p_∞.

The obtained pressure along with the velocity components and temperature extrapolated from the interior domain are used to compute the conservative variables at the first dummy layer from the outflow plane. The corresponding conservative variables at dummy layers further from the plane are obtained by extrapolation.

In the case of supersonic outflow, all flow variables are copied from the interior to the dummy cells.

3.7.7 Far Field

The approach of Whitfield and Janus [17] is based on the characteristic form of the one-dimensional Euler equations normal to the boundary. The two basic flow situations at the farfield boundary are sketched in Fig. 3.4. The flow can either enter or it can leave the domain. Therefore, depending on the local Mach number, four different types of farfield boundary conditions have to be treated: supersonic inflow, supersonic outflow, subsonic inflow, and subsonic outflow.

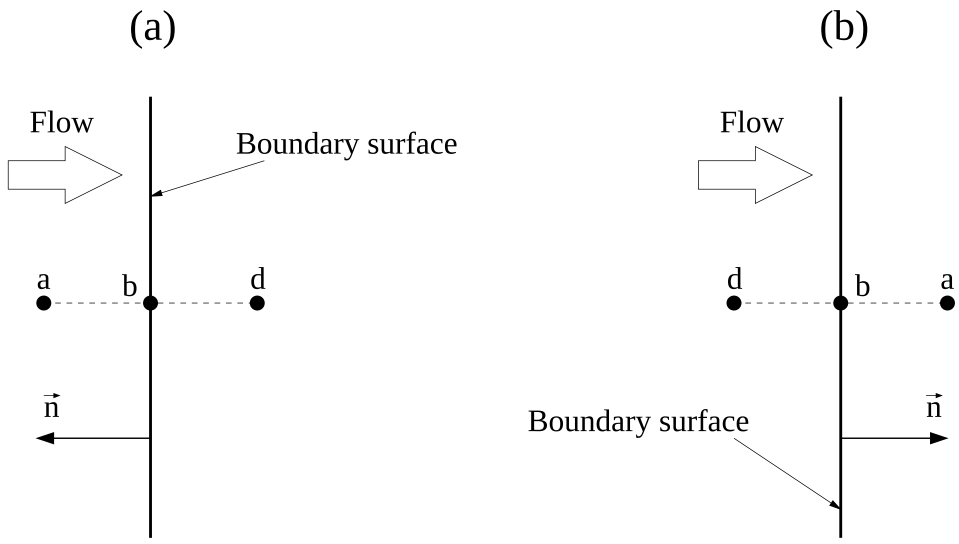


Figure 3.4: Farfield boundary: inflow (a) and outflow (b) situation. Position a is outside, b on the boundary, and position d is inside the physical domain. The unit normal vector $\vec{n} = [n_x,\, n_y,\, n_z]^T$ points out of the domain.

Supersonic Inflow

For supersonic inflow, all eigenvalues have the same sign. Since the flow is entering the physical domain, the conservative variables on the boundary (point b in Fig. 3.4) are determined by freestream values only. Thus,

$$\vec{W}_b = \vec{W}_a \,. \tag{3.59}$$

The values $\vec{W}_a$ are specified based on the given Mach number M_∞ and on two flow angles (angle of attack, side-slip angle).

Supersonic Outflow

In this case, all eigenvalues have also the same sign. However, the flow leaves now the physical domain and all conservative variables at the boundary must be determined from the solution inside the domain. This can be accomplished simply by setting

$$\vec{W}_b = \vec{W}_d \,. \tag{3.60}$$

Subsonic Inflow

Here, four characteristics enter and one leaves the physical domain. Therefore, four characteristic variables are prescribed based on the freestream values. One characteristic variable is extrapolated from the interior of the physical domain. This leads to the following set of boundary conditions [17]

$$
\begin{aligned}
p_b &= \frac{1}{2}\{p_a + p_d - \rho_0 c_0 [n_x(u_a - u_d) + n_y(v_a - v_d) + n_z(w_a - w_d)]\} \\
\rho_b &= \rho_a + (p_b - p_a)/c_0^2 \\
u_b &= u_a - n_x(p_a - p_b)/(\rho_0 c_0) \\
v_b &= v_a - n_y(p_a - p_b)/(\rho_0 c_0) \\
w_b &= w_a - n_z(p_a - p_b)/(\rho_0 c_0)\,,
\end{aligned}
\tag{3.61}
$$

where ρ_0 and c_0 represent reference state. The reference state is normally set equal to the state at the interior point (point d in Fig. 3.4). The values in point a are determined from the freestream state.

Subsonic Outflow

In the case of subsonic outflow, four flow variables (density and the three velocity components) have to be extrapolated from the interior of the physical domain. The remaining fifth variable (pressure) must be specified externally. The primitive variables at the farfield boundary are obtained from [17]

$$
\begin{aligned}
p_b &= p_a \\
\rho_b &= \rho_d + (p_b - p_d)/c_0^2 \\
u_b &= u_d + n_x(p_d - p_b)/(\rho_0 c_0) \\
v_b &= v_d + n_y(p_d - p_b)/(\rho_0 c_0) \\
w_b &= w_d + n_z(p_d - p_b)/(\rho_0 c_0)
\end{aligned}
\tag{3.62}
$$

with p_a being the prescribed static pressure.

Physical properties in the dummy cells are in all previous cases obtained by linear extrapolation from the states b and d.

3.7.8 Symmetry

If the flow is to be symmetrical with respect to a plane, the first condition which must be met is that there is no flux across the boundary. This is equivalent to the requirement that the velocity normal to the symmetry boundary is zero. Furthermore, the following gradients have to vanish:

- gradient normal to boundary of a scalar quantity,
- gradient normal to boundary of a tangential velocity,

- gradient along the boundary of the normal velocity (since $\vec{v} \cdot \vec{n} = 0$).

These conditions can be written as

$$\begin{aligned} \vec{n} \cdot \vec{\nabla} U &= 0 \\ \vec{n} \cdot \vec{\nabla}(\vec{v} \cdot \vec{t}) &= 0 \\ \vec{t} \cdot \vec{\nabla}(\vec{v} \cdot \vec{n}) &= 0 \, , \end{aligned} \tag{3.63}$$

where U stands for a scalar variable and $\vec{t}$ denotes a vector tangential to the symmetry boundary.

The implementation of the symmetry boundary condition is largely simplified by the use dummy cells. The flow variables in the dummy cells are obtained using the concept of reflected cells. This means that scalar quantities like density or pressure in the dummy cells are set equal to the values in the opposite interior cells. The velocity components are reflected with respect to the boundary

$$\vec{v}_1 = \vec{v}_2 - 2 \, V_2 \, \vec{n} \, , \tag{3.64}$$

where $V_2 = u_2 n_x + v_2 n_y + w_2 n_z$ is the contravariant velocity and $\vec{n} = [n_x, n_y, n_z]^T$ stands for the wall unit-normal vector. Further, index 1 denotes the first dummy cell and index 2 the first physical cell. The normal gradient of the normal velocity in the dummy cell equals to that in the opposite interior cell, but it has a reversed sign.

3.7.9 Block Interface

The first part of the data structure consists of the numbering of the block boundaries. The numbering scheme used in Rocflo is displayed in Fig. 3.5. The numbering strategy in Fig. 3.5 can be summarised as follows:

$$\begin{array}{ll} \text{boundary } 1: & i = IBEG \\ \text{boundary } 2: & i = IEND \\ \text{boundary } 3: & j = JBEG \\ \text{boundary } 4: & j = JEND \\ \text{boundary } 5: & k = KBEG \\ \text{boundary } 6: & k = KEND \, . \end{array}$$

It is important that all blocks employ the same numbering scheme. The indices i, j, k of the grid points in the computational space are defined in the ranges

$$IBEG \le i \le IEND \, , \quad JBEG \le j \le JEND \, , \quad KBEG \le k \le KEND \, .$$

The cell indices I, J, K, which are required by the cell-centered scheme are defined in a similar way. Due to the dummy cells/points, the physical cells/points will have a certain offset from the start or the end of each range.

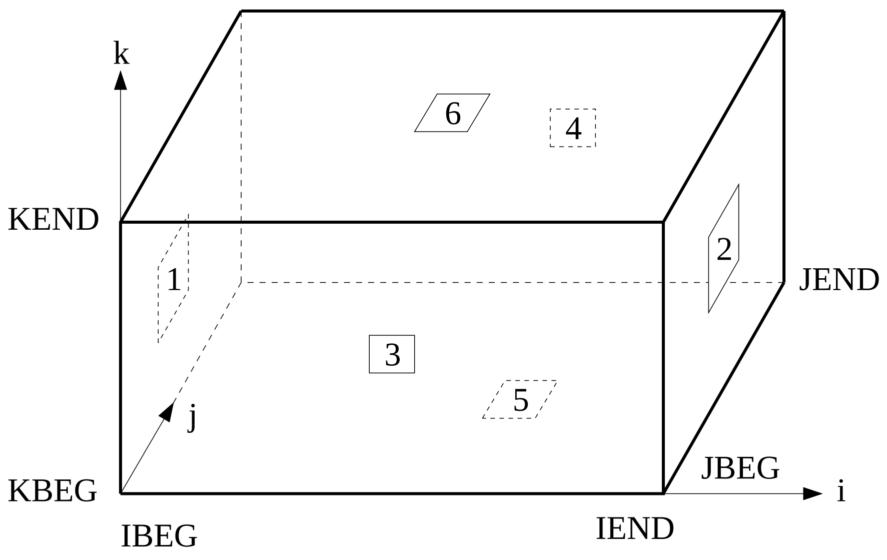


Figure 3.5: Numbering of the sides of the computational space and of the block boundaries.

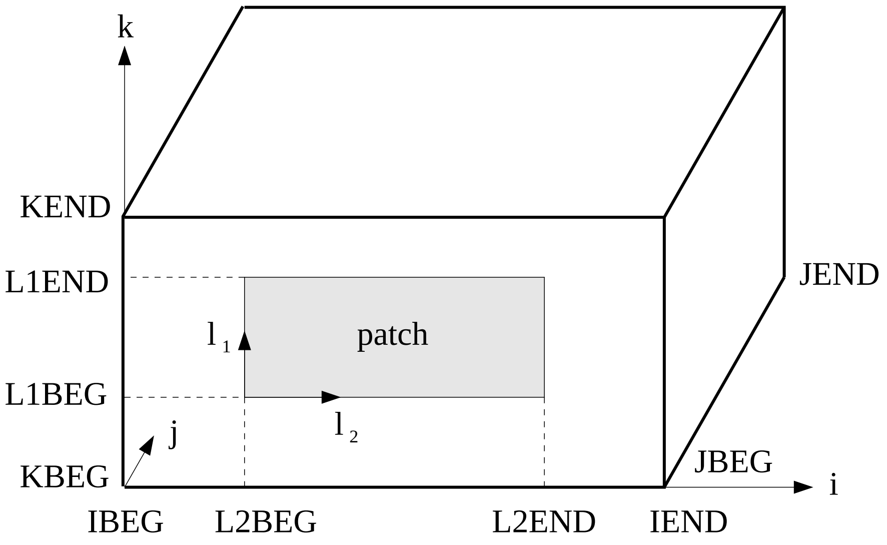


Figure 3.6: Coordinates of a boundary patch in computational space. The patch has its own local coordinate system l_1, l_2.

The boundary of each block is in general divided into a number of non-overlapping patches. This allows the specification of different boundary conditions on the same block boundary. The situation is depicted in Fig. 3.6. For a unique identification of each patch it is necessary to store the number of the corresponding block and the number of the block boundary. Furthermore, the origin, the height and the width of the patch must be stored. For this purpose, the coordinates $L1BEG$, $L1END$, $L2BEG$ and $L2END$ are used in Fig. 3.6. The local coordinate system of the patch l_1, l_2 is oriented according to the cyclic directions. This means, that if we consider the i-coordinate, j and k will be the first and the second cyclic direction. In the case of the j-coordinate, the cyclic directions will become k and i, respectively. Therefore, since the patch in Fig. 3.6 is on the $j = JBEG$ boundary, the l_1-coordinate is oriented in the k-direction and l_2 in the i-direction. The application of the cyclic directions allows for a unique definition of the patch orientation.

The exchange of flow quantities between two blocks is sketched in Fig. 3.7. If the blocks are on different processors, the procedure consists of two steps. In the first step, variables from the part of the domain, which is overlapped by the dummy layers of the adjacent patch

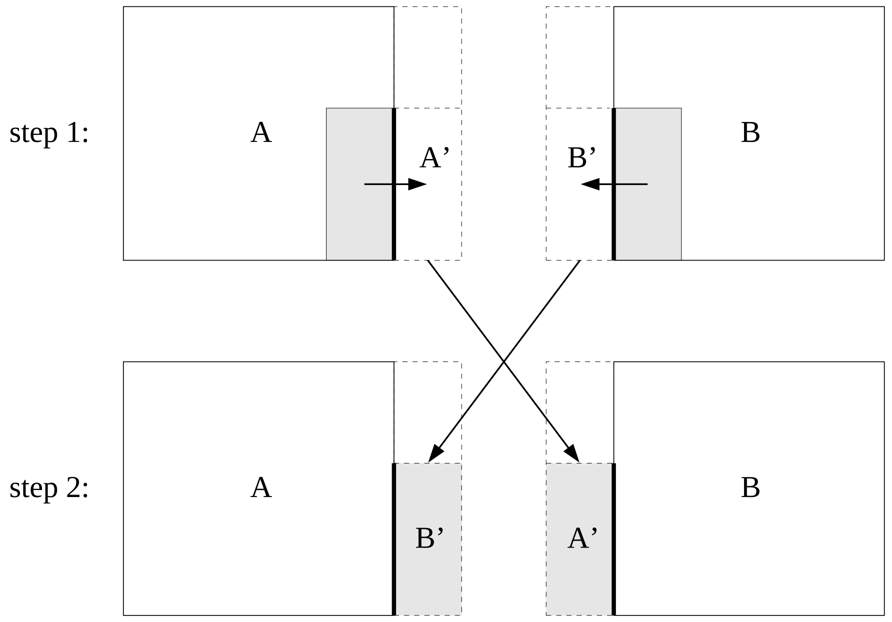


Figure 3.7: Exchange of flow variables (in shaded regions A', B') between two blocks A and B. Dummy layers are denoted by a dashed line.

are written to a temporary storage (A' and B' in Fig. 3.7). This is done for all blocks. In the second step, the data in A' and B' is exchanged between both blocks. This means that A' is written to the dummy layers of block B and B' to the dummy layers of block A. If the two patches have a different orientation (i.e. l_1, l_2 of the current and the source block are not aligned), the data must be transformed accordingly. In the case both blocks are on the same processor, the data is exchanged using a transformation matrix.

The dummy cells at the corners and the edges of the computational space (for the numbering see Fig. 3.1 and Fig. 3.2, respectively) require a separate treatment since they are always connected to multiple patches. In the case of Rocflo, a data structure is implemented which directly relates each corner or edge cell to its source cell in another block. The search algorithm employs the logical relations between the blocks instead of geometrical data. The advantage is that no data has to exchanged between the processors in a parallel run. However, the current approach has the following limitations:

- the cells must be attached to at least one inter-block or periodic boundary,
- the source cell has to be within the physical domain of the source block,
- in the case of multiple source cells only the first one is used,
- only the conserved variables are exchanged (no geometry).

3.7.10 Periodicity

In the case of translational periodicity, the implementation is identical to the block interface in Subsection 3.7.9. Data from the physical cells at one periodic boundary are simply copied to the dummy cells of the other boundary.

In the case of rotational periodicity, the data are copied between the two boundaries as before. However, since the periodicity condition is based on a rotation of the coordinate system, all vector quantities like velocity or gradients of scalars have to be transformed accordingly. Scalar quantities like pressure or density, which are invariant with respect to coordinate rotation, remain unchanged. Assuming the rotational axis is parallel to the x-axis (see Fig. 3.8), the rotation matrix becomes

$$\bar{\mathcal{R}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, \tag{3.65}$$

where the angle ϕ between the periodic boundaries A and B is positive in the clockwise direction. Hence, for example, the velocity vector transformed from boundary A to B reads

$$\vec{v}_B = \bar{\mathcal{R}}\,\vec{v}_A\,. \tag{3.66}$$

It is easy to show that the x-component of $\vec{v}_A$ (i.e., u_A) is not changed by the rotation. Thus, $u_B = u_A$. The gradients of all flow quantities are transformed in a similar way.

3.8 Grid Motion

Rocflo relies on 5 different schemes for moving the grid according to the deformation of the boundaries and/or the burn-back of the propellant:

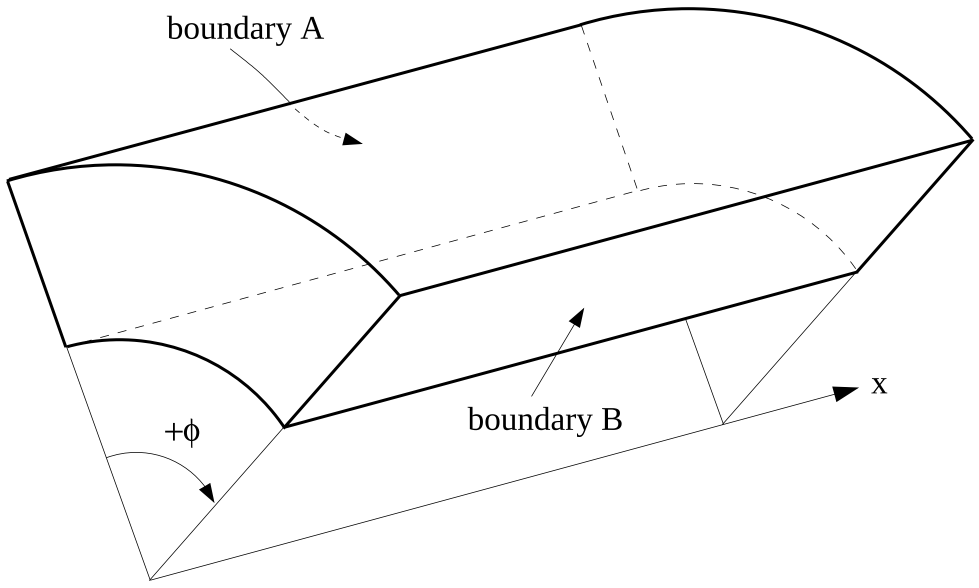


Figure 3.8: Rotationally periodic boundaries (A and B). The rotational axis is assumed to coincide with the x-axis.

- `Block Transfinite Interpolation (TFI)`
- `Block Weighted Laplacian Smoothing`
- `Global Weighted Laplacian Grid Motion`
- `Global NuLaplacian Grid Motion`
- `Global Elliptic PDE Grid Motion`

Which type of grid motion can best be used in certain fluid-structure interaction case depends on the degree of the solid deformation incurred. For small deformation the Block TFI is appropriate, while for large and complex deformations the Global Elliptic PDE method is probably more suitable.

3.8.1 Block Transfinite Interpolation

The first algorithm initially moves the grid in each block separately by using the linear Transfinite Interpolation (TFI) method [18] with the blending functions of Soni [19]. TFI is employed to interpolate the deformation of the block boundaries onto the grid from the previous time step. After this initial step, deformations are exchanged between the blocks. This is important in cases where a block shares only a point or a line with the moving surface, as the block 2 in Fig. 3.9. As the last step, grids are regenerated inside those blocks, whose boundaries were moved by adjacent blocks.

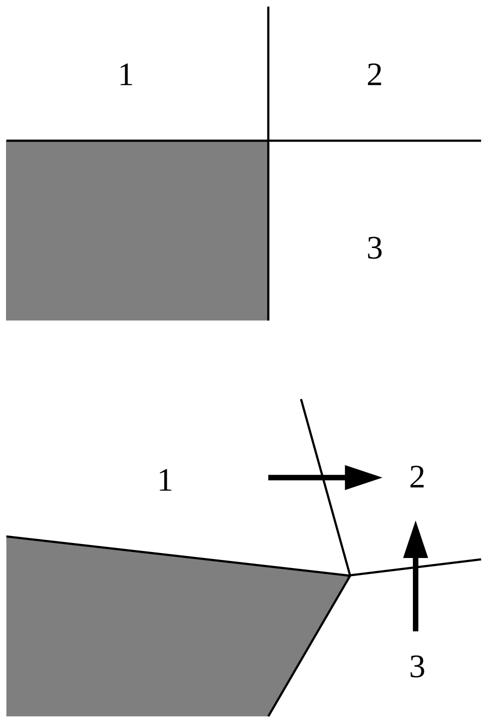


Figure 3.9: Adjustment of the boundaries of a block (2) which does not share a cell face with the deforming body (gray). Boundary movement is communicated through the blocks 1 and 3. Top: undeformed, bottom: deformed configuration.

3.8.2 Block Weighted Laplacian Smoothing

In cases with a sufficiently large surface movement, it becomes necessary to deform and move grid blocks even far away from the surface. This situation is handled by a second algorithm. This scheme globally smoothes the grid and the block boundaries by solving a Laplace equation for the deformations. The derivatives are weighted by the inverse distance between points, so that the stifness of the grid can be adjusted. Jacobi iteration is employed for the solution of the system of equations since it can be easily parallelized. Grid inside the blocks is then regenerated using TFI as in the first method. The effect of the global Laplacian smoothing is demonstrated in Fig. 3.10, where it is compared to the block TFI motion.

3.8.3 Global Weighted Laplacian Grid Motion

For a very large and complex surface movements, such as those involving rotational motions, even the block Laplacian smoothing is not sufficiently robust. This is because the Laplacian smoothing propagates the motion very slowly to the other faces of the block. To tackle this shortcoming, in Global Weighted Laplacian grid motion, the block corners are moved

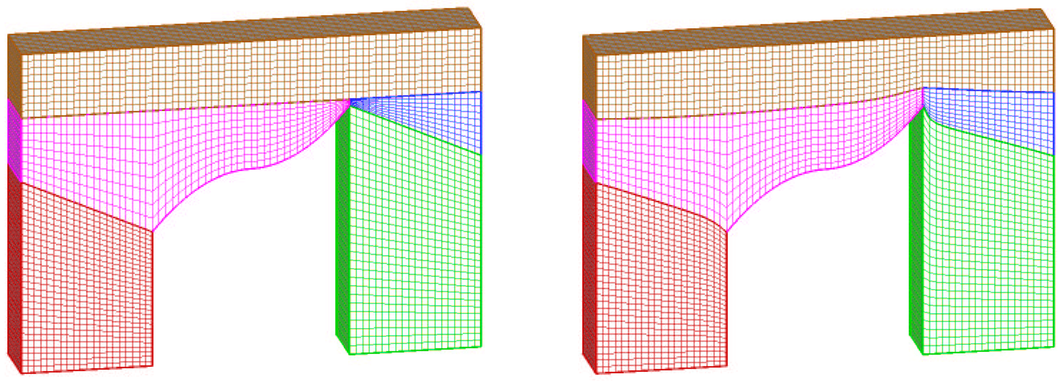


Figure 3.10: Local (left) versus global (right) grid motion for a multi-block test case.

directly by averaging the motions of the neighboring block corners which are selected based on their distance to the block corner being moved. The neigboring block corners can include those from the same block and neigboring blocks. Once the block corners are moved, usually by iterating the averagin process by 10 iterations, the following procedures are the same as those employed in the Block Laplacian smoothing. Specifically, the block edge, surface, and volume grid are moved using TFI employing the block corner motions as the initial drivers, then followed by the Laplacian smoothing for the volume grid. The word 'Global' in this method and the following methods refer to the computation of block-corners motions.

3.8.4 Global NuLaplacian Grid Motion

This type of grid motion is a variant of the Global Weighted Laplacian grid motion. The difference lays in the method of determining the weights of the smoothing. Instead of using the inverse distances to compute the weights, NuLaplacian employes constant but non-uniform weights (off from 1/4 in 2D and 1/6 in 3D) and applys a corrective procedure by orthogonally projecting the resulting grid movements to the cell faces. Therefore the name NuLaplacian wich stands for Non-Uniform Laplacian. In contrast to what one may expect from the orthogonal projection procedure, NuLaplacian does not guarantee boundary orthogonality. The seemingly orthogonal grid resulting from this method is a side effect of the choice of non-uniform weights, and the orthogonality in general occurs in the interior grid, rather than at block boundaries. Selecting uniform weights, NuLaplacian will give the same result as TFI.

3.8.5 Global Elliptic PDE Grid Motion

The main drawback of the Weighted Laplacian and NuLaplacian methods is their inability to force orthogonality at block boundaries. This boundary orthogonality is crucial for the robustness of the grid motion on boundary fitted grid, especially for alternating or periodic or fluctuating motions, either caused by solid vibrations or flexing motions. Grid motions involving one direction of motion as opposed to alternating motion, like a simple propellant regression, is usually easier to handle. To produce grids that has the property of boundary orthogonality, we resort to solving Elliptic Partial Differential Equations that emerge from the composite map of a grid control mapping from the computational space C to the parameter space P, and an inverse harmonic mapping from the parameter space P to the physical space D. The resulting equations are Poisson equations that need to be solved using an iterative method such as Successive Overrelaxation method. Specifically, we compute the block edges motion using the linear TFI, followed by a 2D Poisson smoothing for block surfaces with the edges grid as Dirichlet boundary condition. After all the surface grid are moved, we apply a 3D Poisson smoothing for the block interior grid. Variants of 2D and 3D Poisson smoothing are available, depending on the type of the grid control mapping used. Here, we employ Hermite cubic interpolation for the grid control mapping in 2D and bi-linear interpolation in 3D. The detail of the surface and volume Poisson smoothing procedures implemented in Rocflocan be found in the [20], Chapter 4.

Chapter 4

Source Code Organization

The source codes of Rocflo, of the physical modules, and of the utility programs are located in the `Rocstar/RocfluidMP/Codes` directory of the CVS repository. The code can be compiled either as a stand alone version or as a library to be linked with GENx. The user can also specify which physical modules are to be included during the compilation.

In order to check out the source codes, type:

```
% cvs co -d RocfluidMP Rocstar/RocfluidMP/Codes
```

This creates the directory `RocfluidMP` in the current path. Besides the source codes, the directory will also contain a number of test cases and the documentation. The contents of the directory `RocfluidMP` is the following:

- `CODING_RULES` – coding rules for the fluids codes
- `Makefile` – main makefile used to compile Rocflo, the physical modules and utility programs
- `Makefile.AIX` – machine dependent makefile for IBM-SP
- `Makefile.IRIX64` – machine dependent makefile for SGI
- `Makefile.Linux` – machine dependent makefile for Linux
- `Makefile.SunOS` – machine dependent makefile for SUN
- `Makefile.common` – makefile with machine independent settings
- `Makefile.dep` – makefile for the generation of file dependencies
- `README` – quick overview of the fluid codes
- `TEMPLATE.F90` – template for subroutines
- `build_lib` – directory where the library is build

- `build_std` – directory where the standalone code is build
- `build_util` – directory where the utilities are build
- `calcs` – test cases for the verification and validation of Rocflo, cases for benchmarking the code
- `docs` – documentation including the user's and developer's manuals, and the description of code versions
- `genx` – routines required for the coupling to GENx
- `libflo` – library of routines specific to the Rocflo base solver
- `libfloflu` – library of routines common to Rocflo and the unstructured code
- `libflu` – library of routines specific to the unstructured code
- `modflo` – F90 modules specific to Rocflo
- `modfloflu` – F90 modules common to both fluids codes
- `modflu` – F90 modules specific to the unstructured code
- `rocflo` – routines of the base Rocflo solver
- `rocflu` – routines of the base unstructured solver
- `rocinteract` – routines of the interaction module
- `rocpart` – routines specific to the Lagrangian particles module
- `rocperi` – routines specific to specific periodic flows
- `rocrad` – routines specific to the radiation module
- `rocsmoke` – routines specific to the Eulerian particles module
- `rocspecies` – routines specific to the species module
- `rocturb` – routines specific to the turbulence module
- `standalone` – routines specific to the stand alone code
- `utilities` – various utility programs (see User's Manual)

In order to distinguish between the structured and the unstructured solver, as well as the physical modules, prefixes are added to module and subroutine names:

- RFLO – structured flow solver

- RFLU – unstructured flow solver
- PEUL – Eulerian particles (smoke)
- PLAG – Lagrangian particles
- RADI – radiation
- SPEC – chemical species
- TURB – turbulence
- PERI – periodic flows

The same names are also utilized as compiler switches (see User's Manual). Generic subroutines, which are common to both codes and are also employed by the physical modules, have no prefix (directories `modfloflu` and `libfloflu`).

Chapter 5

Data Structures

The grid can consist of one or multiple *regions* with structured grid. Thus, a region is the basic data entity. Structured regions are composed of hexahedra, although the hexahedra can be degenerated, for example, into prisms. Each region can again consist of a number of grid *levels* – either in geometrical sense or, in the case of algebraic multigrid, as a union of equations. Besides the above two basic data structures, which are named as **regions** and **levels** in the code, there are variables identical to all regions. They are related to time stepping, multigrid, MPI, etc. These variables are gathered in a structure called **global**. The variables **regions**, **levels**, **global** are defined as local variables within the solver.

5.1 F90 Modules

Within `RocfluidMP`, there are three directories with F90 modules:

- `modflo`
- `modfloflu`
- `modflu`

The directory `modflo` contains modules used only by Rocflo:

- `Indexing.h` – preprocessor directive for mapping from i, j, k space to a single pointer
- `RFLO_ModIndexing.F90` – mapping from a single pointer to the i, j, k space, pointer to source region using patch mapping
- `RFLO_ModInterfacesExternal.F90` – interfaces to all external subroutines called either from Rocflo or GenX
- `RFLO_ModInterfacesLibrary.F90` – interfaces to all subroutines contained in the Rocflo library (i.e. `libflo`)

- `RFLO_ModInterfacesSolver.F90` – interfaces to all subroutines called only from Rocflo
- `RFLO_ModBoundaryConditions.F90` – collection of boundary conditions routines
- `RFLO_ModDegenerateCornEdge.F90` – collection of degenerate edges/corners identification routines
- `RFLO_ModElliptSmoothing.F90` – collection of grid smoothing routines based on solving Poisson equations
- `RFLO_ModExtrapolation.F90` – collection of 2D patch and 3D block extrapolation routines
- `RFLO_ModFiniteDifference.F90` – collection of 2D and 3D finite differencing routines
- `RFLO_ModGridControlMap.F90` – collection of routines needed for 2D and 3D grid control map matrices
- `RFLO_ModLaplaceSmoothing.F90` – collection of grid smoothing routines based on weighted Laplacian and NuLaplacian techniques
- `RFLO_ModMoveGridFrame.F90` – collection of global (block and grid level) grid motion routines
- `RFLO_ModVectorTensor.F90` – collection of vector and tensor operations routines
- `RFLO_ModStatsBoundaryConditions.F90` – collection of statistics boundary conditions routines

The directory `modfloflu` contains modules which are common to Rocflo and Rocflu:

- `ModBndPatch.F90` – data related to boundary patches and boundary conditions (data types *t_patch*, *t_bcvalues* and *t_tbcvalues*)
- `ModDataStruct.F90` – definitions of main derived data types (*t_level, t_region, t_dCell, t_dCellSrc, t_dCellTransf*)
- `ModDataTypes.F90` – parameters to define floating point accuracy and the maximum length of character constants
- `ModError.F90` – error codes and error function
- `ModGenx.F90` – global variables for coupling with GenX
- `ModGlobal.F90` – global variables in the data type *t_global*
- `ModGrid.F90` – geometrical data and grid speeds (data type *t_grid*)
- `ModInteract.F90` – variables of the interaction module

- `ModInterfaces.F90` – interfaces to all subroutines common to both flow solvers (in `libfloflu`); wrapper to solver specific interfaces and interfaces of physical modules
- `ModInterfacesBcond.F90` – interfaces to all subroutines related to boundary conditions
- `ModInterfacesEulerian.F90` – interfaces to all subroutines of Rocsmoke
- `ModInterfacesIO.F90` – interfaces to all I/O subroutines
- `ModInterfacesInteract.F90` – interfaces to all subroutines of the interaction module
- `ModInterfacesLagrangian.F90` – interfaces to all subroutines of Rocpart
- `ModInterfacesMixt.F90` – interfaces to all subroutines related to the mixture
- `ModInterfacesPeriodic.F90` – interfaces to all subroutines of Rocperi
- `ModInterfacesRadiation.F90` – interfaces to all subroutines of Rocrad
- `ModInterfacesSpecies.F90` – interfaces to all subroutines of Rocspecies
- `ModInterfacesStatistics.F90` – interfaces to all subroutines of the statistics tool
- `ModInterfacesTurbulence.F90` – interfaces to all subroutines of Rocturb
- `ModInterfacesUtil.F90` – interfaces to all subroutines of utility functions
- `ModMPI.F90` – parameters defining the master process and the MPI include file
- `ModMaterials.F90` – parameters of the injected species
- `ModMixture.F90` – user input and flow variables for the core solver (data types *t_mixt_input* and *t_mixt*)
- `ModParameters.F90` – definitions of various integer parameters
- `ModPartEul.F90` – user input and flow variables for the Eulerian particles (data types *t_peul_input* and *t_peul*)
- `ModPartLag.F90` – user input and flow variables for the Lagrangian particles (data types *t_plag_input* and *t_plag*)
- `ModPeriodic.F90` – user input and variables for the Rocperi module
- `ModRadiation.F90` – user input and flow variables for the radiation module (data types *t_radi_input* and *t_radi*)
- `ModRandom.F90` – functions and variables of the portable random number generator

- `ModSortSearch.F90` – used only by Rocflu
- `ModSpecies.F90` – user input and flow variables for the chemical species (data types *t_spec_input* and *t_spec*)
- `ModTools.F90` – used only by Rocflu
- `ModTurbulence.F90` – user input and flow variables for the turbulence module (data types *t_turb_input* and *t_turb*)

Other modules existing in directory modfloflu not mentioned here are used only by Rocflu. The directory `modflu` contains modules used only by the unstructured solver.

5.2 Data Types and Variables

The following subsections describe all variables of Rocflo contained in the derived data types. The variables are grouped accordingly to the derived types they belong to. Since the basic flow variables have the same appearance for all physical modules and the core solver, they will be explained first.

5.2.1 Types of Flow Variables

There are the following types of flow variables:

- *cv* - conservative variables
- *dv* - dependent variables
- *tv* - transport variables
- *gv* - gas variables
- *tav* - time averaged (statistics) variables

The flow variables always have two dimensions. The first dimension represents the equations, which are addressed using parameters provided in `modfloflu/ModParameters.F90`. The second index corresponds to the grid cells. This means in the case of Rocflo that the computational coordinates i, j, k have to be transformed into a single index. The F90 module `modflo/RFLO_ModIndexing.F90` provides mapping functions for 2-D and 3-D cases. It also contains an inverse mapping function. The precise meaning of these variables will depend on the physical module they are used in.

5.2.2 ModBndPatch: Type t_bcvalues

This derived type is related to boundary condition values of a patch. It contains the following variables:

- *bcSet* – boundary condition set for this patch (true or false)
- *setMotion* – internal motion set for this patch (true or false)
- *nSendBuff* – dimension of the send buffer (MPI)
- *nRecvBuffer* – dimension of the receive buffer (MPI)
- *iRequest* – number of the MPI request to send data
- *distrib* – determines whether the BC values are constant for the patch (=0) or varying over the patch (=1)
- *nData* – number of BC values
- *nSwitches* – number of BC options (e.g. like sub- or supersonic)
- *switches* – values of BC options
- *vals* – BC values
- *maxChange* – max. relative change of ρ, ρE or T before the extrapolation to the dummy cells is switched from 1st- to 0-th order (slip walls or injection boundaries only)
- *bndVel* – velocity components of patch internal motion
- *sendBuff* – send buffer
- *recvBuff* – receive buffer
- *tbcs* – time dependent boundary conditions (see Subsection 5.2.4)

5.2.3 ModBndPatch: Type t_patch

This derived type is related to a boundary patch. It contains the following variables:

- *bcType* – type of boundary condition:
  - 10–19 = inflow
  - 20–29 = outflow
  - 30–39 = block boundary, conforming grid

  - 40–49 = block boundary, non-conforming grid with integer ratio
  - 50–59 = block boundary, irregular non-conforming grid
  - 60–69 = slip wall
  - 70–79 = noslip wall
  - 80–89 = far field
  - 90–99 = injection
  - 100–109 = symmetry
  - 110–119 = translational periodicity
  - 120–129 = rotational periodicity

- *bcCoupled* – determines whether the patch receives BC values from an external source (e.g. from Rocburn); the values can be either *BC_INTERNAL* or *BC_EXTERNAL*
- *bcMotion* – determines whether the patch receives grid deformation from an external source (e.g. from Rocsolid/frac/prop); the values can be either *BC_INTERNAL* (e.g. from Rocflo internal slipwall motion) or *BC_EXTERNAL* (e.g. from Rocsolid/frac/prop)
- *periodAngle* – angle between periodic boundaries
- *srcRegion* – number of source region (other side of block interface)
- *srcPatch* – number of the source patch
- *lbound* – side of the computational space (see Fig. 3.5)
- *l1beg/end* – coordinates of the patch in the l_1-direction (cf. Fig. 3.6)
- *l2beg/end* – coordinates of the patch in the l_2-direction
- *srcLbound* – side of the computational space for the source region
- *srcL1beg/end* – coordinates of the source patch in the l_1-direction
- *srcL2beg/end* – coordinates of the source patch in the l_2-direction
- *corns(4)* – node-index of patch corners
- *position(4)* – position code of patch corners (0 at interacting surface, 1 interior, 2 non interacting surface)
- *align* – determines whether l_1 and l_2 directions (see Fig. 3.6) on the current and the adjacent patch are aligned with each other (true or false)
- *thrustCalc* – determines whether the patch is used for calculating the thruct

- *l1VecSrc* – vector in the l_1-direction of the source patch (used to determine the orientation of patches if only one cell in common in the respective direction)
- *l1VecSrc* – vector in the l_2-direction of the source patch
- *surfCoord* – coordinates of the surface of the patch
- *st* – control parameter grid (in Elliptic PDE grid motion)
- *stOld* – control parameter grid from previous iteration (Elliptic PDE patch grid motion)
- *sti,j* – first derivative of st w.r.t i- and j-coordinate
- *stii, jj* – second derivative of st w.r.t i and j-coordinate
- *stij* – mixed derivative of st w.r.t i- and j-coordinate
- *pfun* – control function matrix (EPDE patch grid motion)
- *aij-aipjp* – matrix coefficients in Poisson system (EPDE patch grid motion)
- *bcFlag* – determines whether the patch is ignitable (=1) or not (=0)
- *bFlag* – determines whether the patch is burning (=1) or not (=0)
- *duAlp* – boundary movement
- *mdotAlp* – mass flow rate
- *rhovfAlp* – density times relative flow velocity
- *tflmAlp* – flame temperature
- *pf* – static pressure
- *qc* – convective heat flux
- *qr* – radiative heat flux
- *rhofAlp* – density
- *nfAlp* – normal vector
- *tracf* – tractions
- *tempf* – fluids temperature
- *valMixt* – BC values for the mixture (the core solver)
- *valBola* – BC values for the mixture wall modeling

- *valTurb* – BC values for the turbulence
- *valSpec* – BC values for the species
- *valPeul* – BC values for the smoke (Eulerian particles)
- *valRadi* – BC values for the radiation

5.2.4 ModBndPatch: Type t_tbcvalues

This derived type is related to time-dependent modification of a boundary condition data. It contains the following variables:

- *tbcType* – type of the modification of the boundary condition
- *switches* – values of options
- *params* – user input parameters
- *mean* – mean values
- *svals* – space independent tbc
- *bvals* – spatially distributed tbc

5.2.5 ModDataStruct: Type t_dCell

This derived type is related to dummy cells at the corners (see Fig. 3.1) and at the edges (cf. Fig. 3.2) of the computational domain. It contains the following variables:

- *interact* – at least some of the cells interact with another domain (true/false)
- *interType* – typ of interaction, full or partial
- *degenrt* – degeneration type of edge/corner (0=non-degenerate)
- *cells* – data associated with each dummy cell

5.2.6 ModDataStruct: Type t_dCellSrc

This derived type is related to the data of a particular edge or corner cell. It contains the following variables:

- *rotate* – the source cell is rotationally periodic (true/false)
- *srcRegion* – index of the source region
- *srcCell* – index of the source cell
- *srcIndexMapMat(3,4)* – mapping matrix from current patch to source patch (used only by plag)

5.2.7 ModDataStruct: Type t_dCellTransf

This derived type is related to the transfer of solution variables between edge and/or corner cells and their respective source cells (only if on a different processor). It contains the following variables:

- *nCells* – number of cell data to send/receive
- *iRequest* – number of the MPI request to send data
- *buff* – send/receive buffer
- *iRequestMetrics* – number of the MPI request to send metrics data (used only by plag)
- *buffMetrics* – send/receive buffer for metrics data (used only by plag)

5.2.8 ModDataStruct: Type t_level

This derived type is related to a multigrid level. It contains the following variables:

- *dt* – local time step
- *grid* – grid data
- *gridOld* – grid data before movement
- *gridOld2* – grid data for third order dual (implicit) timestepping
- *mixt* – data of the mixture (core solver)
- *turb* – data of the turbulence
- *radi* – data of the radiation
- *peul* – data of the smoke (eulerian particles)
- *plag* – data of the lagrangian particles
- *peri* – data of the periodic flows
- *patches* – data of the boundary patches
- *edgeCells* – data of the 12 edge-cell blocks (see Fig. 3.2)
- *cornerCells* – data of the 8 corner-cell blocks (see Fig. 3.1)
- *sendEcCells* – send buffer and MPI requests for the edge and corner cells
- *recvEcCells* – receive buffer for the edge and corner cells

5.2.9 ModDataStruct: Type t_region

This derived type is related to a grid region (block). It contains the following variables:

- *localNumber* – local number of the region on the current processor
- *active* – region active (*ACTIVE* or *OFF*)
- *procid* – processor ID
- *iRegionGlobal* – reserved for future use
- *nDumCells* – number of dummy cells (equal or larger than two)
- *nEdgeCells* – number of parts of edges
- *nPatches* – number of boundary patches
- *nGridLevels* – number of grid levels
- *startLevel* – grid level to start (FMG or successive grid refinement)
- *currLevel* – current grid level
- *irkStep* – current Runge-Kutta stage
- *dimWork1D* – dimension of the 1-D work array
- *dimWork2D* – dimension of the 2-D work array
- *work1D* – 1-D work array
- *work2D* – 2-D work array
- *mixtInput* – user input for the core solver
- *turbInput* – user input for the turbulenc
- *peulInput* – user input for the smoke
- *plagInput* – user input for the lagrangian particles
- *radiInput* – user input for the radiation
- *periInput* – user input for the periodic flows
- *inrtInput* – user input for the interaction module
- *levels* – data related to the grid levels
- *global* – pointer to global variables (strange isn't it?)

5.2.10 ModGlobal: Type t_global

This derived type is related to global variables (not related to regions or grid levels). It contains the following variables:

- *winName* – name of fluids window (required for GenX)
- *caseName* – name of the flow case (determines names of all I/O files)
- *inDir* – directory for input data
- *outDir* – directory for output data
- *gridFormat* – format of the grid file (*FORMAT_ASCII* or *FORMAT_BINARY*)
- *solutFormat* – format of the solution file
- *gridSource* – used only by Rocflu
- *initFlowFlag* – used only by Rocflu
- *refVelocity* – reference velocity
- *refPressure* – reference static pressure
- *refDensity* – reference density
- *refCp* – reference c_p value
- *refGamma* – reference γ value
- *refLength* – reference length
- *refREnum* – Reynolds number
- *refVisc* – reference viscosity μ
- *prLam* – reference laminar Prandtl number
- *prTurb* – reference turbulent Prandtl number
- *scnLam* – reference laminar Schmidt number
- *scnTurb* – reference turbulent Schmidt number
- *accelOn* – switch for acceleration terms
- *accelX/Y/Z* – acceleration in x-, y-, z-direction
- *dualTstSource* – compute source terms of dual-tst (true/false)

- *predCorrIter* – predictor-corrector iteration (if within GENx)
- *predictSol* – guess start solution for subiterations in dual-tst
- *flowType* – type of flow (*FLOW_EULER* or *FLOW_NAVST*)
- *nrkSteps* – number of Runge-Kutta stages
- *currentIter* – current iteration number
- *maxIter* – maximum number of iterations
- *writeIter* – number of iterations between output of solution file
- *printIter* – number of iterations between output of convergence and probe data
- *currentTime* – current time
- *dTimeSystem* – system time step (used when coupled to GenX)
- *dtImposed* – specified time step Δt
- *maxTime* – maximum physical time of simulation (stand alone code)
- *writeTime* – time between output of solution file
- *printTime* – time between output of convergence and probe data
- *timeStamp* – initial time (to read the restart solution)
- *timeStampPrep* – used only by Rocflu
- *resTol* – tolerance of the density residual for a steady flow
- *tolSubIter* – tolerance of subiterations in dual-tst
- *stopRun* – run is to be stopped if ¿ 0
- *startLevel* – initial multigrid level
- *cycleType* – type of multigrid cycle (*MGCYCLE_NO*, *MGCYCLE_V*, or *MGCYCLE_W*)
- *refineIter* – number of iterations before switching to next finer grid (FMG scheme)
- *moveGridScheme* – type of grid motion scheme (MOVEGRID_BLOCKS or MOVEGRID_GLOBAL)
- *moveGridNiter* – number of iterations of the global grid motion
- *moveGridNbour* –

- *moveGridRegNc* –
- *moveGridNsMatch* –
- *moveGridNshareMax* –
- *moveGridWeight* – edge weighting within the global grid motion
- *moveGridPower* –
- *moveGridAmplifX,Y,Z* –
- *moveGridOrthWght* –
- *moveGridOrthCell* –
- *nProbes* – number of probes in the flow field
- *probePos* – location of a probe (region, (i, j, k)-index)
- *probeXYZ* – location of a probe (region, (x, y, z)-coordinate)
- *probeOpenClose* – swith for opening and closing the probe file each time new values are being written
- *probeSaveIter* – number of iterations between output of the probe data
- *probeSaveTime* – time between output of the probe data
- *thrustType* – type of thrust calculated (THRUST_NONE=none, THRUST_MOM= momentum, THRUST_MOMP=momentum and pressure)
- *thrustPlane* – thrust plane (1=x, 2=y, 3=z)
- *thrustSaveIter* – number of iterations between output of the thrust data
- *thrustOpenClose* – swith for opening and closing the thrust file each time new values are being written
- *thrustCoord* – coordinate of the thrust plane
- *thrustPamb* – ambient pressure
- *thrustSaveTime* – ime between output of the thrust data
- *thrustMom* – momentum thrust
- *thrustPress* – pressure thrust
- *thrustTotal* – total thrust

- *infloNijk* –
- *internDeform* –
- *infloPlanEdges* –
- *xyzMinmax* –
- *mixtStatNm* –
- *doStat* –
- *reStat* –
- *mixtNStat* –
- *mixtStatId* –
- *integrTime* –
- *verbLevel* – verbosity level (0=none, 1=moderate, 2=full)
- *mpiComm* – MPI communicator
- *nProcAlloc* – number of processors allocated
- *myProcid* – ID of the processor
- *nRegionsProc* – number of regions per processor; if set to zero, mapping will be done automatically
- *nRequests* – total number of concurrent send requests (MPI)
- *requests* – list of send requests
- *nRegions* – total number of regions
- *dtMin* – largest possible time step (unsteady flow); it already includes the CFL number
- *currentTime* – current physical time
- *resInit* – density residual at the first iteration (used for normalization)
- *residual* – density residual at the current iteration (steady flow only)
- *error* – error flag (0=no error)
- *mpierr* – MPI error flag (0=no error)
- *nFunTree* – current depth of the function tree (used by the error function)

- *functionTree* – names of functions in the function tree (i.e. functions being previously called)
- *forcesOn* – calculation of forces (can be *FORCES_NONE*, *FORCES_PRESS*, or *FORCES_VISC*)
- *forceX/Y/Z* – force in x-, y- and z-direction
- *massIn* – total incomming mass flow
- *massOut* – total outgoing mass flow
- *pi* – π
- *rad* – $\pi/180$
- *genxHandleBc* – GenX handle for boundary conditions
- *genxHandleGm* – GenX handle for geometry

5.2.11 ModGenx: Type t_globalGenx

This derived type is related to the coupling with GenX. It contains the following variables:

- *isDummy* – determines whether the flow solver is run in dummy mode (=1) or not (=0)
- *global* – pointer to structure with global variables
- *regions* – pointer to structure with data of all regions

5.2.12 ModGrid: Type t_grid

This derived type is related to the grid data. It contains the following variables:

- *indSvel* – determines whether grid speeds are calculated (if grid is moving or deforming) or not; 0 means no and 1 means yes; it is also used to multiply the index when accessing the grid speed vectors
- *i/j/kpc* – number of physical cells in the i-, j-, and k-direction
- *boundMoved* –
- *allExternal* –
- *edgeMoved* –
- *ijkDgen* –

- *c2fCoI/J/K* –
- *c2eCoI/J/K* –
- *si/j/k* – face vectors in the *i*-, *j*-, and *k*-direction
- *si/j/kVel* – grid speeds in the *i*-, *j*-, and *k*-direction
- *arcLen12* –
- *arcLen34* –
- *arcLen56* –
- *xyz* – grid coordinates
- *xyzOld* – grid coordinates of previous iteration
- *vol* – cell volumes
- *cofg* – centroids of grid cells
- *cfcI/J/K* –
- *nCorns* –
- *ijkCorn* –
- *nghbor* –
- *nShared* –
- *cShared* –
- *regCorn* –
- *regCornOld* –
- *regCornOrig* –
- *regCornBuff* –
- *regCornOrth* –
- *xyzOrth* –
- *xyzTemp* –
- *stu* –

- *stuOld* –
- *stui/j/k* –
- *stuii/jj/kk* –
- *stuij/ik/jk* –
- *pmat* –
- *aijk-aijpkp* –
- *tofluLoc2g* –

5.2.13 ModMixture: Type t_mixt

This derived type is related to the mixture of gases. It contains the following variables:

- *nDv* – number of dependent variables; currently there are 3: pressure, temperature and speed of sound
- *nTv* – number of transport variables like viscosity or heat conduction coefficient
- *nGv* – number of gas variables; currently there are 2: c_p and the Mol mass
- *nGrad* – number of gradients of scalars
- *indCp* – determines whether c_p is constant over the flow field (=0) or not (=1); it is used to multiply the index when accessing the *gv* vector
- *indMol* – determines whether the Mol mass is constant over the flow field (=0) or not (=1); it is used to multiply the index when accessing the *gv* vector
- *prLam* – laminar Prandtl number
- *prTurb* – turbulent Prandtl number
- *scnLam* – laminar Schmidt number
- *scnTurb* – turbulent Schmidt number
- *cv* – conservative variables
- *cvOld* – conservative variables from previous time step
- *cvn* –
- *cvn1* –

- *cvn2* –
- *dv* – dependent variables
- *tv* – transport variables
- *gv* – gas variables
- *tav* – time averaged values
- *rhs* – right-hand side (residual)
- *rhsOld* –
- *rhsSum* – sum of residuals (required for the classical Runge-Kutta scheme)
- *diss* – numerical dissipation and viscous fluxes
- *fterm* – forcing term (required for multigrid)
- *gradi/j/k* – gradients at the *i*-, *j*-, and *k*-faces of the control volumes
- *srad* – spectral radii in the three computational coordinates
- *epsIrs* – scaled coefficients of implicit residual smoothing

The data type is located in the module `ModMixture.F90`.

5.2.14 ModMixture: Type t_mixt_input

This derived type is related . It contains the following variables:

- *flowModel* – type of flow model (can be either *FLOW_EULER* or *FLOW_NAVST*)
- *fluidModel* –
- *moveGrid* – grid moving (true or false)
- *externalBc* – some of the boundary patches communicate with an external program (true or false)
- *turbModel* – type of turbulence model (*NONE* and *flowModel*=*FLOW_NAVST* means laminar flow
- *spaceDiscr* – type of spatial discretization (central or upwind)
- *spaceOrder* – order of spatial discretization (1 or 2)
- *pSwitchType* – type of pressure switch (see Section 3.3)

- *timeScheme* – type of the time-stepping scheme
- *ldiss* – determines whether the dissipation (numerical, physical) is computed at a certain stage of the multi-stage scheme
- *cfl* – CFL number
- *smoocf* – coefficient of implicit residual smoothing; if ¡0 no smoothing is applied
- *vis2* – coefficient $k^{(2)}$ of the numerical dissipation (see Section 3.3)
- *vis4* – coefficient $k^{(4)}$ of the numerical dissipation
- *pSwitchOmega* – parameter ω in Eq. (3.26); it is related to the TVD-type pressure switch
- *limfac* – limiter coefficient (upwind scheme)
- *epsentr* – entropy correction coefficient (upwind scheme)
- *ark* – stage coefficients
- *grk* – coefficients to multiply the residual before it is added to the sum of residuals from previous stages of the explicit Runge-Kutta scheme
- *trk* – used to multiply Δt to get the actual physical time *at the end* of a Runge-Kutta stage
- *betrk* – blending coefficient between new and old values of the dissipation (numerical and physical) at a stage of the explicit scheme
- *faceEdgeAvg* –
- *iniVelX/Y/Z* –
- *iniPress* –
- *iniDens* –
- *iniXsplit* –
- *iniVelX2/Y2/Z2* –
- *iniVelPress2* –
- *iniVelDens2* –

Chapter 6

Input/Output Files

The execution of Rocflo depends on the following five files:

- User input file
- Input file with boundary values
- Topology file (description of the boundary patches)
- Grid file
- Solution file (must always be present - flow solver does not generate an initial solution)

It should be noted that there is one grid and one solution file for all grid regions. Also the other files are common to all regions. There can be additional files, which store distributions of flow quantities for boundary patches. Furthermore, Rocflo can generate so called probe files, where values of the density, the velocity components, the static pressure and temperature are stored along with the physical time or the iteration number for a given location within the flow field. Rocflo can also read in or generate file which contains a time averaged flow quantities such as the velocity components (statistics file). Finally, Rocflo also stores the convergence history, probe data and the thrust into separate files.

The formats of the grid, solution and statistics file can be either native ASCII or binary. All other files are in a native ASCII format. All files are denoted by the name of the flow case (specified on the command line when invoking the flow solver) and an additional extension:

- `.inp` for user input
- `.bc` for boundary values
- `.top` for topology
- `.grda` for grid file in ASCII format, `.grdb` for binary format
- `.sola` for solution file in ASCII format, `.solb` for binary format

- `.prb` for probe file(s)
- `.thr` for thrust
- `.stata` for statistics file in ASCII format, `.statb` for binary format
- `.con` for convergence history
- `.degec` for degenerate edges and corners

The files with the user input and the boundary values are explained in detail in the User's Manual.

6.1 Grid

The grid file has the extension `.grda[_stamp]` for the ASCII format and `.grdb[_stamp]` for the binary format, respectively. The additional key `_stamp` is added when the grid is moving and/or deforming (thus only for unsteady flow). In this case, `_stamp` denotes the physical time (in seconds) at which the grid file was stored. It has the format 1PE11.5. Hence,

```
casename.grdb_1.23000E-01
```

represents the grid at the time $t = 0.123$ sec. A grid file can be generated by converting a 2-D grid into Rocflo's format using the `rflo2dto3d` utility program. Another possibility is to generate the grid by Gridgen and to convert it into Rocflo's format using `makeflo`, as described in the User's Manual.

6.2 Topology

The topology file has the extension `.top`. The file is automatically generated either by the `makeflo` utility when converting grid files from Gridgen's format into Rocflo's format, or by the `rflosplit` program (see the User's Manual for further details).

6.3 Solution

The solution file has the extension `.sola[_stamp]` for the ASCII format and `.solb[_stamp]` for the binary format, respectively. The additional key `_stamp` can take two different forms. In the case of a steady flow, it is a six digit integer (with all leading zeros - format I6.6) which represents the iteration number. Hence,

```
casename.sola_000123
```

represents the solution at the 123rd iteration.

In the case of an unsteady flow, `_stamp` denotes the physical time (in seconds) at which the solution file was stored. It has the format 1PE11.5. Hence,

```
casename.solb_1.23000E-01
```

represents the solution at the time $t = 0.123$ sec.

The initial solution file (zeroth iteration or $t = 0$) is generated by the pre-processor `rfloprep` using the input file described in the User's Manual.

6.4 Probe

The extension of the probe file(s) is `.prb[_number]`, where `_number` is an integer in the format I3.3 representing the number of the probe. The file contains for each iteration or time step a single line with the iteration number or time and the values of the density, the velocity components, the static pressure and temperature. The number of probes and their location can be specified in the user input file. The offset (in terms of iterations or physical time) at which the data is written to the probe file can also be set in the user input file (see the User's Manual).

6.5 Thrust

The extension of the thrust file is `.thr`. The file contains for each iteration or time step a single line with the iteration number or time and the values of the momentum, pressure, and the total thrust. The location of the thrust plane can be specified in the user input file. The offset (in terms of iterations or physical time) at which the data is written to the thrust file can also be set in the user input file (see the User's Manual).

6.6 Convergence

The convergence file has the extension `.con`. For each iteration or time step, it contains a single line with with the iteration number or time, the density residual as $log(\Delta\rho/\Delta\rho_{initial})$, the forces in all three coordinate directions (in [N]), the inflow mass (in [kg/s]) and the outflow mass (also in [kg/s]). The offset (in terms of iterations or physical time) at which the data is written to the convergence file can be set in the user input file (see the User's Manual).

6.7 Degenerate Edges and Corners

The information regarding degenerate edges and corners can be found in the file with the extension `.degec`. The file contains the region numbers in which degenerate edges and corners occur, and the degenerate edge and corner numbers, along with the degeneration type codes. The degeneration type codes are defined as follows.

Edge degeneration type:

- 1 – EDGE IN PATCH (¡ 4 interior cells meet at the edge)
- -1 – EDGE DETACHED (¿ 4 interior cells meet at the edge)

Corner degeneration type:

- 1 – CORNER IN EDGE (6 interior cells meet at the corner)
- 2 – CORNER IN PATCH (4 interior cells meet at the corner)
- -1 – CORNER DETACHED (¿ 8 interior cells meet at the corner)

Chapter 7

Coding Rules

A template for a subroutine or function is provided in the file `TEMPLATE.F90`. It has the following form:

```
!******************************************************************
!
! Purpose:
!
! Description:
!
! Input:
!
! Output:
!
! Notes:
!
!******************************************************************
!
! $Id: dguide.tex,v 1.9 2005/04/15 18:14:39 wasistho Exp $
!
! Copyright: (c) 2001 by the University of Illinois
!
!******************************************************************

SUBROUTINE
FUNCTION

  USE ModDataTypes
  USE ModError
  USE ModGlobal, ONLY     : global
  USE ModInterfaces, ONLY :
```

```
  IMPLICIT NONE

! ... parameters


! ... loop variables


! ... local variables
  CHARACTER(CHRLEN) :: RCSIdentString

!******************************************************************************

  CALL RegisterFunction( global,'function name',__FILE__ )

! comment ---------------------------------------------------------------------

! comment

! - comment

! --- comment

  CALL DeregisterFunction( global )

END SUBROUTINE
END FUNCTION

!******************************************************************************
!
! RCS Revision history:
!
! $Log: dguide.tex,v $
! Revision 1.9  2005/04/15 18:14:39  wasistho
! modified files
!
! Revision 1.8  2003/12/11 00:22:11  jblazek
! More changes.
!
!
!******************************************************************************
```

Names of subroutines, functions or F90 modules, which are specific to a particular code or physical module, have to contain one of the prefixes:

- *RFLO_* for Rocflo,
- *RFLU_* for Rocflu,
- *PEUL_* for Eulerian particles,
- *PFEU_* for Fast Eulerian particles,
- *PLAG_* for Lagrange particles,
- *RADI_* for radiation,
- *SPEC_* for species,
- *TURB_* for turbulence,
- *PERI_* for specific periodic flows.

Sources which are common to both codes and to all physical modules are without any prefix. If a routine of a physical module is called from Rocflo an *S* is to be added before the prefix (e.g., *SPEUL_*).

If parts of a source code should be compiled only for either Rocflo or Rocflu, or for a particular physical module, preprocessor directives, i.e.,

```
#ifdef KEYWORD
  ...
  some specific source code
  ...
#endif
```

have to be utilized. There are the following keywords:

- `RFLO` for Rocflo,
- `RFLU` for Rocflu,
- `PEUL` for Eulerian particles,
- `PFEU` for Fast Eulerian particles,
- `PLAG` for Lagrange particles,
- `RADI` for radiation,
- `SPEC` for species,

- `TURB` for turbulence,
- `PERI` for specific periodic flows,
- `MPI` for parallelization.

They are identical to the switches used during compilation of the code, as it is described in Rocflo User's Guide.

Bibliography

[1] Blazek, J.: *Computational Fluid Dynamics: Priciples and Applications.* Elsevier Science Ltd., Oxford, GB, 2001.

[2] Thomas, P.D.; Lombard, C.K.: *Geometric Conservation Law and Its Application to Flow Computations on Moving Grids.* AIAA Journal, 17 (1979), pp. 1030-1037.

[3] Bruner, C.W.S.: *Geometric Properties of Arbitrary Polyhedra in Terms of Face Geometry.* AIAA Journal, 33 (1995), p. 1350.

[4] Jameson, A.; Schmidt, W.; Turkel, E.: *Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes.* AIAA Paper 81-1259, 1981.

[5] Jameson, A.: *Time-Dependent Calculations Using Multigrid with Applications to Unsteady Flows Past Airfoils and Wings.* AIAA Paper 91-1596, 1991.

[6] Martinelli, L.: *Calculation of Viscous Flows with a Multigrid Method.* Ph.D. Thesis, Dept. of Mech. and Aerospace Eng., Princeton University, 1987.

[7] Martinelli, L.; Jameson, A.: *Validation of a Multigrid Method for Reynolds Averaged Equations.* AIAA Paper 88-0414, 1988.

[8] Turkel, E.; Swanson, R.C.; Vatsa, V.N.; White, J.A.: Multigrid for Hypersonic Viscous Two- and Three-Dimensional Flows. AIAA Paper 91-1572, 1991.

[9] Roe, P.L.: *Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes.* J. Computational Physics, 43 (1981), pp. 357-372.

[10] Van Leer, B.: *Towards the Ultimate Conservative Difference Scheme V. A second Order Sequel to Godunov's method.* J. Computational Physics, 32 (1979), pp. 101-136.

[11] Harten, A.; Lax, P.D.; Van Leer, B.: *On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws.* Soc. Industrial and Applied Mathematics Rew., 25 (1983), No. 1.

[12] Harten, A.; Hyman, J.M.: *Self Adjusting Grid Methods for One-Dimensional Hyperbolic Conservation Laws.* J. Computational Physics, 50 (1983), pp. 235-269.

[13] Mavriplis, D.J.; Jameson, A.: *Multigrid Solution of the Navier-Stokes Equations on Triangular Meshes.* AIAA Journal, 28 (1990), pp. 1415-1425.

[14] Jameson, A.; Baker, T.J.: *Solution of the Euler Equations for Complex Configurations.* AIAA Paper 83-1929, 1983.

[15] Holmes, D.G.: *Inviscid 2D Solutions on Unstructured, Adaptive Grids.* Numerical Methods for Flows in Turbomachinery, VKI-LS 1989-06, 1989.

[16] Rudy, D.H.; Strikwerda, J.C.: *Boundary Conditions for Subsonic Compressible Navier-Stokes Calculations.* Computers and Fluids, 9 (1981), pp. 327-338.

[17] Whitfield, D.L.; Janus, J.M.: *Three-Dimensional Unsteady Euler Equations Solution Using Flux Vector Splitting.* AIAA Paper 84-1552, 1984.

[18] Gordon, W.N.; Hall, C.A.: *Construction of Curvilinear Coordinate Systems and Application to Mesh Generation.* Int. J. Num. Methods in Engineering, 7 (1973), pp. 461-477.

[19] Soni, B.K.: *Two- and Three-Dimensional Grid Generation for Internal Flow Applications of Computational Fluid Dynamics.* AIAA Paper 85-1526, 1985.

[20] Thompson, J.F.; Soni, B.K.; Weatherill, N.P. (eds.): *Handbook of Grid Generation.* CRC Press, Boca Raton, 1999.