HOPR Documentation

HOPR Developers

Apr 25, 2024

USER GUIDE

1 User Guide 3
1.1 Installation . . . 3
1.1.1 Executable download . . . 3
1.1.1.1 Docker . . . 3
1.1.1.2 AppImage . . . 3
1.1.2 Prerequisites . . . 4
1.1.2.1 Compilers . . . 5
1.1.3 Required Libraries . . . 5
1.1.3.1 Installing/setting up GCC . . . 5
1.1.3.2 Installing/setting up HOPR . . . 5
1.1.3.3 Installing/setting up HDF5 . . . 6
1.1.4 Troubleshooting . . . 7
1.1.4.1 Wrongly set HDF5_ROOT variable . . . 7
1.1.4.2 Pre-compiled HDF5 via Spack and/or cmake . . . 8
1.1.5 Testing HOPR . . . 8
1.2 HOPR HDF5 Curved Mesh Format . . . 8
1.2.1 Introduction and Main Idea Behind the Mesh Format . . . 9
1.2.2 Global Attributes . . . 9
1.2.3 Data Arrays . . . 10
1.2.3.1 Example 3D Mesh . . . 11
1.2.3.2 Element Information (ElemInfo) . . . 13
1.2.3.3 FEM Element Information (FEMElemInfo) . . . 13
1.2.3.4 Side Information (SideInfo) . . . 14
1.2.3.5 Edge Information (EdgeInfo) . . . 16
1.2.3.6 Vertex Information (VertexInfo) . . . 19
1.2.3.7 Node Coordinates and Global Index . . . 22
1.2.3.8 Boundary Conditions . . . 24
1.2.4 Parallel Read-in . . . 25
1.2.5 Element Definitions . . . 31
1.2.5.1 Element Types . . . 31
1.2.5.2 Element High Order Nodes . . . 31
1.2.5.3 Element Corners, Sides . . . 32
1.2.5.4 Element Connectivity . . . 32
1.2.6 Additional Extensions: Hanging Node Interface . . . 36
1.2.6.1 Changes to Existing Data Format . . . 36
1.2.6.2 Additional Information for Octrees . . . 39
1.3 Appendix . . . 41
1.3.1 Tested compiler combinations . . . 41
1.4 List of Parameters . . . 41

2 Developer Guide **45**
2.1 Github Workflow . . . 45
2.1.1 Issues & Milestones . . . 45
2.1.2 Merge Request . . . 46
2.1.3 Release and deploy . . . 47
2.1.3.1 Release Tag . . . 47
2.1.3.2 Collaborative Numerics Group . . . 47
2.1.3.3 GitHub . . . 49
2.2 Style Guide . . . 49
2.2.1 Header of Functions and Subroutines . . . 49
2.2.2 Variables . . . 50
2.2.3 Functions and Control Structures . . . 51
2.2.4 Workflow Description . . . 51
2.3 Building the Documentation . . . 53
2.4 Compiler Options . . . 54
2.5 Building the AppImage Executable . . . 56
2.6 Troubleshooting . . . 56
2.6.1 dlopen(): error loading libfuse.so.2 . . . 57
2.7 Markdown Examples . . . 57
2.7.1 hyperlinks . . . 57
2.7.2 Code environment . . . 57
2.7.3 Equations . . . 57
2.7.4 Bibtex, cite . . . 58
2.7.5 section references . . . 58
2.7.6 Figures, caption . . . 58
2.7.7 tables . . . 59
2.7.8 unnumbered section headings . . . 59
2.7.9 Code blocks for various languages . . . 59

3 Built-In Mesh Generators **61**
3.1 Straight-Edged Boxes . . . 61
3.1.1 Cartesian Box . . . 61
3.1.1.1 Cartesian Box: Description of Parameters . . . 61
3.1.1.2 Cartesian Box: Boundary Conditions and Sketch . . . 62
3.1.1.3 Cartesian Box: Output Visualization . . . 63
3.1.1.4 Cartesian Box: Exemplary Variations of Boundary Conditions . . . 63
3.1.2 Periodic Boundary Conditions . . . 64
3.1.2.1 Periodic Boundary Conditions: Description of Parameters . . . 64
3.1.2.2 Periodic Boundary Conditions: Boundary Conditions and Sketch . . . 65
3.1.3 Multiple Cartesian Boxes . . . 65
3.1.3.1 Multiple Cartesian Boxes: Definition of Multiple Cartesian Boxes . . . 65
3.1.3.2 Multiple Cartesian Boxes: Sketch . . . 66
3.1.3.3 Multiple Cartesian Boxes: Output Visualization . . . 67
3.1.4 Stretching Functions . . . 67
3.1.4.1 Stretching Functions: Definition of Stretching Functions . . . 67
3.1.4.2 Stretching Functions: Building a Cartesian Box with Stretched Elements . . . 67
3.1.4.3 Stretching Functions: Building Multiple Cartesian Boxes with Stretched Elements . 68
3.1.4.4 Stretching Functions: Sketch . . . 69
3.1.4.5 Stretching Functions: Output Visualization . . . 69
3.2 Curved Meshes . . . 69
3.2.1 Curved Structured Mesh . . . 69
3.2.1.1 Stretching Functions . . . 70
3.2.1.2 Examples . . . 71
3.2.2 Mesh Curving by Post-Deformation . . . 71

3.2.2.1 Post-Deformation from a box to a cylinder . . . 72
3.2.2.2 Parameter Variations . . . 72
3.2.3 Curved Torus . . . 72
3.2.3.1 Post-Deformation from a box to a torus . . . 72
3.2.4 Curved Sphere . . . 73
3.2.4.1 Post-Deformation from a box to a sphere . . . 73
3.2.4.2 Spherical shell . . . 73

4 External Meshes **75**
4.1 External Meshes without Curved Boundaries . . . 75
4.1.1 External Mesh . . . 75
4.1.2 Output Visualization . . . 77
4.2 External Meshes with Curved Boundaries . . . 77
4.2.1 Mesh Curving Techniques . . . 77
4.2.2 Curving Using Normal Vectors . . . 77
4.2.3 Curving Using Subdivided Surface Mesh . . . 78
4.2.4 Use of pre-curved meshes . . . 79

5 Agglomeration of Block-Structured Meshes **81**
5.1 Block-Structured Meshes . . . 81
5.1.1 Restrictions on the block-structured meshes . . . 82
5.1.2 Initial Meshes . . . 82
5.1.3 Description of Parameters . . . 82

6 Post-processing Meshes **85**
6.1 Mesh uncurving . . . 85
6.2 Mesh Refinement . . . 86
6.3 Generation of Hexahedral Meshes . . . 86

7 Visualization **87**
7.1 HOPR Output Parameter . . . 87
7.2 Visualization with Paraview . . . 88
7.2.1 Parameter Settings . . . 88
7.2.2 Recommended Settings . . . 88

8 References **89**

Bibliography **91**

HOPR HOPR is an open-source tool for the generation of three-dimensional unstructured high-order meshes. It is licensed under GPLv3 and written in Fortran.

CHAPTER

ONE

USER GUIDE

This user guide describes the installation procedure and the mesh format used in hopr and is intended for people using hopr as a mesh generator.

1.1 Installation

The following chapter describes the installation procedure on a Linux machine, possibly requiring root access. This may include the installation of required prerequisites, e.g., setting up HDF5. Please note that high-performance clusters usually have a module environment, where you have to load the appropriate modules instead of compiling them yourself.

1.1.1 Executable download

HOPR can be installed on a Linux machine without the need of compiling the source code. Currently there are two ways by which HOPR is distributed, as a *docker container* and as an *AppImage* executable.

1.1.1.1 Docker

Install the package via

```
docker pull ghcr.io/hopr-framework/docker-ubuntu20-hopr-exec:latest
```

1.1.1.2 AppImage

Download the pre-compiled (on Centos7) executable from the release tag assets or from the GitHub Actions builds: cmake-ninja to get a bleeding-edge version. Note that you need to be logged into Github (see button "Sign in to view logs") in order to be able to download any artefacts. After downloading the binary file, it has to be made executable via

```
chmod +x hopr-x86_64.AppImage
```

before being used.

The following table shows that there is no apparent drop in performance when using a pre-compiled executable:

Table 1.1: Performance test with pre-compiled executable: Cart-2D 665600 #Elements

Binary	Laptop	Hawk	Commit
	AMD Ryzen 7 4800H	AMD EPYC 7702	
hopr-x86_64.AppImage	22.087	29.274	0324ba3
hopr/master/gcc/12.2.0/openmpi/4.1.4/hdf5/1.12.2	27.064		afd6756

1.1.2 Prerequisites

HOPR supports Linux-based systems only, requires a x86_64 compliant platform and has been tested on the following platforms

- Linux Mint 17 or newer
- Red Hat Enterprise Linux 7.6 or newer
- SUSE Linux Enterprise Server 11 SP3 or newer
- Ubuntu 14.04 LTS, 16.04 LTS and 18.04 LTS, 20.04 LTS 20.10, 21.04 and 22.04 LTS

For **tested combinations** of prerequisites (HDF5, OpenMPI, CMake etc.) and known problems that may occur, visit Chapter *Appendix*.

The suggested packages in this section can be replaced by self compiled versions. The required packages for the Ubuntu Linux distributions are listed in Table 1.2. Under Ubuntu, they can be obtained using the apt environment:

```
sudo apt-get install git
```

Table 1.2: Debian/Ubuntu packages. x: required, o: optional, -: not available

Package	Ubuntu 14.04	Ubuntu 16.04	Ubuntu 18.04	Ubuntu 20.04
git	x	x	x	x
cmake	x	x	x	x
cmake-curses-gui	o	o	o	o
liblapack3	x	x	x	x
liblapack-dev	x	x	x	x
gfortran	x	x	x	x
g++	x	x	x	x
mpi-default-dev	x	x	x	x
zlib1g-dev	-	x	x	x
exuberant-ctags	o	o	o	o

On some systems it may be necessary to increase the size of the stack (part of the memory used to store information about active subroutines) in order to execute **HOPR** correctly. This is done using the command

```
ulimit -s unlimited
```

from the command line. For convenience, you can add this line to your `.bashrc`.

1.1.2.1 Compilers

HOPR requires a C and a Fortran 2003 compliant compiler, compilers tested with **HOPR** include

- GNU Compiler Collection 4.6 or newer
- Intel C/Fortran Compiler 12 or newer (recommended)
- CRAY Compiler Environment 8.1 or newer

HOPR furthermore requires CMake 3.5.2+ as a build system.

1.1.3 Required Libraries

The following libraries are required, if not mentioned otherwise, including their development headers. Libraries marked with a star (*) can alternatively be provided by HOPR.

- BLAS/LAPACK* (or compatible, e.g. ATLAS, MKL)
- CGNS*
- HDF5*
- libc6
- zlib
- Python 2.7 or newer (optional)

If not present on your system, **HOPR** can automatically download and compile these libraries

- HDF5 (1.12.0 if OpenMPI 4.0.0+ is detected, 1.10.6 otherwise)
- LAPACK (0.3.17)/OpenBLAS (3.10.0)
- CGNS (3.4.1)

For a list of tested library version combinations, see Chapter *Appendix*.

1.1.3.1 Installing/setting up GCC

Additional packages are required starting at specific versions of the GCC compiler suite.

GCC Version	Ubuntu 20.04 (and older)
9.3.0	libmpfr-dev libmpc-dev

1.1.3.2 Installing/setting up HOPR

HOPR supports CMake as a build system, which should be available on most systems. Ensure that your environment variables `CC` and `FC` (as well as their corresponding MPI counterparts `MPICC`and `MPIFC` if compiling with MPI support) point to the correct compiler.

For compiling HOPR, create a new sub-directory, e.g. “build” . Inside that directory execute

ccmake ..

Here you can specify library paths and options. If no preinstallied libraries for HDF5 and CGNS are found these libraries will be downloaded and built automatically. Press `c` to configure and `g` to create the Makefiles. Finally compile HOPR by typing `make`.

`LIBS_USE_CGNS`: If the user does not need the cgns library (i.e. HOPR mesh is not built via a cgns input meshfile), the cmake option `LIBS_USE_CGNS` can be set to `OFF`, which skips the installation of the cgns library. Note that the cmake tests that depend on CGNS will not be executed.

`HOPR_INSTRUCTION`: Processor instruction settings (mainly depending on the hardware on which the compilation process is performed or the target hardware where HOPR will be executed). This variable is set automatically depending on the machine where HOPR is compiled. CMake prints the value of this parameter during configuration

```
  -- Compiling Release/Profile with [GNU] (v12.2.0) fortran compiler using HOPR_
↪INSTRUCTION [-march=native] instruction
```

When compiling HOPR on one machine and executing the code on a different one, the instruction setting should be set to `generic`. This can be accomplished by running

```
  cmake -DHOPR_INSTRUCTION=-mtune=generic
```

To reset the instruction settings, run cmake again but with

```
  -DHOPR_INSTRUCTION=
```

which resorts to using the automatic determination depending on the detected machine.

1.1.3.3 Installing/setting up HDF5

An available installation of HDF5 can be utilized with **HOPR**. This requires properly setup environment variables and the compilation of HDF5 during the **HOPR** compilation has to be turned off (`LIBS_BUILD_HDF5 = OFF`). If this option is enabled, HDF5 will be downloaded and compiled. However, this means that every time a clean compilation of **HOPR** is performed, HDF5 will be recompiled. It is preferred to either install HDF5 on your system locally or utilize the packages provided on your cluster.

The recommended HDF5 version to use with **HOPR** is **hdf5-1_12_0**. In the following a manual installation of HDF5 is described, if HDF5 is already available on your system you can skip to the next section *Setting environment variables*.

Manual HDF5 installation

First, download HDF5 from HDFGroup (external website) and extract it

```
tar xvf hdf5-version.tar.gz
```

Then create a build folder

```
cd hdf-version && mkdir -p build
```

and configure HDF5 to install into "/opt/hdf5/1.X.X" (your choice, should be writable)

```
cmake -DBUILD_TESTING=OFF -DHDF5_BUILD_FORTRAN=ON -DHDF5_BUILD_CPP_LIB=OFF -DHDF5_BUILD_
↪EXAMPLES=OFF -DHDF5_ENABLE_PARALLEL=ON -DHDF5_BUILD_HL_LIB=ON -DHDF5_BUILD_TOOLS=ON -
↪DHDF5_ENABLE_F2003=ON -DBUILD_SHARED_LIBS=OFF -DCMAKE_BUILD_TYPE=Release -DCMAKE_
↪INSTALL_PREFIX=/opt/hdf5/1.X.X ..
```

Make and install (if you chosen a folder required root access)

```
make && make install
```

Setting environment variables

Depending whether HDF5 was installed using *configure* or *CMake*, different settings for the HDF5_DIR variable are required

- Configure

  ```
  export HDF5_DIR = /opt/hdf5/1.X.X
  ```

- CMake

  ```
  export HDF5_DIR = /opt/hdf5/1.X.X/shared/cmake/XXX
  ```

If your CMake version is above 3.9.X, CMake uses a new `findPackage` routine, requiring that `HDF5_ROOT` is set

```
export HDF5_ROOT=/opt/hdf5/1.X.X
```

For convenience, you can add these lines to your `.bashrc`.

IMPORTANT: Note that `HDF5_ROOT` must be cleared or set to the correct path when using `LIBS_BUILD_HDF5 = ON` to prevent cmake from compiling hopr and cgns with different HDF5 versions. Otherwise, an error might occur, see *Wrongly set HDF5_ROOT variable*.

1.1.4 Troubleshooting

Sometimes errors occur during installation, for which standard fixes may apply.

1.1.4.1 Wrongly set HDF5_ROOT variable

Requirements: The cmake options `LIBS_BUILD_HDF5 = ON` and `LIBS_BUILD_CGNS = ON` have been set.

The output error might look like this during compilation

```
[ 6%] Building C object src/CMakeFiles/cgns_static.dir/cgns_internals.c.o
/hdf5/hdf5-1.12.2/include/H5public.h:68:10: fatal error: mpi.h: No such file or directory
68 | #include <mpi.h>
| ^~~~~~~
compilation terminated.
```

or the build test might fail with the following message

```
WRITING THE DEBUGMESH...
   #Elements           113
   WRITE DATA TO CGNS FILE... SPHERE_CURVED_Debugmesh.cgnsWarning! ***HDF5 library␣
↪version mismatched error***
...
...
...
#16  0x149524759d8f in __libc_start_call_main
     at ../sysdeps/nptl/libc_start_call_main.h:58
```

(continues on next page)

(continued from previous page)

```
#17  0x149524759e3f in __libc_start_main_impl
     at ../csu/libc-start.c:392
#18  0x4047d4 in ???
#19  0xffffffffffffffff in ???
```

The cause of the problem is that `export HDF5_ROOT=/opt/hdf5/vX.X.X/...` sets the `HDF5_ROOT` environment variable, which leads to HDF5 being built with possibly a different version (or compiler settings) for HOPR and CGNS. The variable is also exposed in cmake

```
HDF5_ROOT                        /opt/hdf5/1.X.X
```

Note that it does not matter if the correct path is exported via `export HDF5_DIR=...` if the variable **HDF5_ROOT** is also set as CGNS automatically searches for the latter.

To fix this problem, set `export HDF5_ROOT=` in the installation terminal.

1.1.4.2 Pre-compiled HDF5 via Spack and/or cmake

Requirements: The cmake options `LIBS_BUILD_HDF5 = OFF` and `LIBS_BUILD_CGNS = ON` have been set. Furthermore, it is not clear whether the pre-installed HDF5 library is installed using Spack **AND/OR** built via cmake.

The output error might look like this during compilation

```
CMake Error at CMakeLists.txt:210 (add_executable):
add_executable cannot create imported target "h5dump" because another
target with the same name already exists.
```

To fix this problem, pre-compile HDF5 using **configure** instead of **cmake AND/OR** do not use Spack. Otherwise compile HDF5 using `LIBS_BUILD_HDF5 = ON` and do not forget to clear the `HDF5_ROOT` variable, see *Wrongly set HDF5_ROOT variable*.

1.1.5 Testing HOPR

After compiling, tests are automatically run for each parameterfile provided in sub-directories of the `tutorials` directory. The runs are pre-built by cmake in the `build/buildTests` directory and executed there.

1.2 HOPR HDF5 Curved Mesh Format

Authors: Florian Hindenlang, Thomas Bolemann, Tobias Ott, Stephen Copplestone, Marcel Pfeiffer, Patrick Kopper

Last modified: November 23, 2023

1.2.1 Introduction and Main Idea Behind the Mesh Format

The High Order Preprocessor (HOPR) is able to generate high order unstructured 3D meshes, including tetrahedra, pyramids, prisms and hexahedra. The HDF5 library (http://www.hdfgroup.org/) allows to use parallel MPI-I/O, thus the mesh format is designed for a fast parallel read-in, using large arrays. There is also the GUI *HDFView* to browse h5 files.

An important feature is that the elements are ordered along a space-filling curve. This allows a simple domain decomposition during parallel read-in, where one simply divides the number of elements by the number of domains, so that each domain is associated with a contiguous range of elements. That means one can directly start the parallel computation with an arbitrary number of domains ($\geq$ number of elements) and always read the same mesh file.

For each element, the neighbor connectivity information of the element sides and the element node information (index and position) are stored as a package per element, allowing to read contiguous data blocks for a given range of elements. To enable a fast parallel read-in, the coordinates of the same physical nodes are stored several times, but can be still associated by a unique global node index.

Notes:

- Array-indexing starts at 1! (Fortran/Matlab Style)
- Element connectivity is based on CGNS unstructured mesh standard (CFD general notation system, http://cgns.sourceforge.net), see Section *Element Corners, Sides*
- The polynomial degree N_{geo} of the curved element mappings is globally defined. Straight-edged elements are found for $N_{geo} = 1$.
- Only the nodes for the volume element mapping and no surface mappings are stored.
- Curved node positions in reference space are uniform for all element types (see Section *Element High Order Nodes*).
- Data types: we use 32bit INTEGER and 64bit REAL (double precision), if not stated differently.

HOPR generates **_mesh.h5* files. You can find examples of the mesh file by executing the tutorials in HOPR, and you can browse the files using *HDFView*.

1.2.2 Global Attributes

These attributes are defined globally for the whole mesh as given in table Table 1.3. For a mesh with elements having only straight edges, the polynomial degree of the element mapping is *Ngeo*$= N_{geo} = 1$. A mesh with curved elements has a fixed polynomial degree $N_{geo} > 1$ for all elements.

Table 1.3: Mesh File attributes.

Attribute	Data type	Description
Version	REAL	Mesh File Version
Ngeo ≥ 1	INTEGER	Polynomial degree N_{geo} of element mapping, used to determine the number of nodes per element
nElems	INTEGER	Total number of elements in mesh
nSides	INTEGER	Total number of sides (or element faces) in mesh
nNodes	INTEGER	Total number of nodes in mesh
nUniqueSides	INTEGER	Total number of geometrically unique sides in the mesh
nUniqueNodes	INTEGER	Total number of geometrically unique nodes in the mesh
nBCs	INTEGER	Size of the Boundary Condition list
FEMconnect	STRING "ON"/"OFF"	"ON" if FEM edge and vertex connection have been built and written to file.
only for **FEMconnect="ON"**:		
nEdges	INTEGER	Total number of entries in the EdgeInfo array (=sum over elements of nEdge(ElemType))
nVertices	INTEGER	Total number of entries in the VertexInfo array (=sum over elements of nVertices(ElemType))
nUniqueEdges	INTEGER	Total number of geometrically unique edges in the mesh
nFEMSides	INTEGER	Total number of topologically (includes periodicity) unique sides in the mes (needed for a FEM solver)
nFEMEdges	INTEGER	Total number of topologically (includes periodicity) unique edges in the mesh (needed for a FEM solver)
nFEMEdgeConnections	INTEGER	Size of **EdgeConnectInfo**
nFEMVertices	INTEGER	Total number of topologically (includes periodicity) unique vertices in the mesh (needed for a FEM solver)
nFEMVertexConnections	INTEGER	Size of **VertexConnectInfo**

1.2.3 Data Arrays

The mesh information is organized in arrays. The data is always stored in blocks for each element, which results in storing it multiple times. However, this way, each processor has a defined, non overlapping, range of **geometry and connectivity information**, where it can perform IO operations, minimizing the need of communication between processors.

The **ElemInfo** array is the first to read, since it contains the data range of each element in the **SideInfo**, **EdgeInfo**, **VertexInfo** and **NodeCoords / GlobalNodeIDs** arrays.

Table 1.4: List of all data arrays in mesh file. Dimensions marked with * will be distributed in parallel read mode.

Array Name	Description	Type	Size
ElemInfo	Start\End positions of element data in **SideInfo /NodeCoords**	INTEGER	(1:6,1:**nElems***)
SideInfo	Side Data / Connectivity information	INTEGER	(1:5,1:**nSides***)
EdgeInfo	Element Edge information and offsets in **EdgeConnectInfo**	INTEGER	(1:3,1:**nEdges***)
NodeCoords	Node Coordinates	REAL	(1:3,1:**nNodes***)
GlobalNodeIDs	Globally unique node index	INTEGER	(1:**nNodes***)
BCNames	List of user-defined boundary condition names (max. 255 Characters)	STRING	(1:**nBCs**)
BCType	Four digit boundary condition code	INTEGER	(1:4,1:**nBCs**)
ElemBarycenters	Barycenter location of each element	REAL	(1:3,1:**nElems***)
ElemWeight	Element Weights for domain decomposition (=1 by default)	REAL	(1:**nElems***)
ElemCounter	mesh statistics (no. of elements of each element type)	INTEGER	(1:2,1:11)
only for **FEMconnect="ON"**:			
FEMElemInfo	Start\End positions of element data in **EdgeInfo/VertexInfo**	INTEGER	(1:4,1:**nElems***)
EdgeConnectInfo	Connectivity information for each element edge (needed for a FEM solver)	INTEGER	(1:2,1:nFEMEdgeConnections)
VertexInfo	Element Vertex Data information and and offsets in **VertexConnectInfo**	INTEGER	(1:3,1:**nVertices***)
VertexConnectInfo	Connectivity information for each element vertex (needed for a FEM solver)	INTEGER	(1:2,1:nFEMVertexConnections)

1.2.3.1 Example 3D Mesh

In the following sections, we explain the array definitions and show an example, which refers to the mesh in Fig. 1.1 with straight-edges, so $N_{geo} = 1$. There is one element of each type, a tetrahedron, a pyramid, a prism and a hexahedron, four elements in total. Corner nodes and element sides have unique indices.

The global attributes of the mesh are shown in Table 1.5.

Table 1.5: Global attributes for example 3D mesh with 4 elements.

Ngeo	1	nElems	4 (Prism,Hex,Tet,Pyra)	nBCs	4
nSides	20 (=5+6+4+5)	nUniqueSides	16	nFEMSides	16
nEdges	35	nUniqueEdges	22	nFEMEdges	22
nNodes	23 (=6+8+4+5)	nUniqueNodes	11	nFEMVertices	11

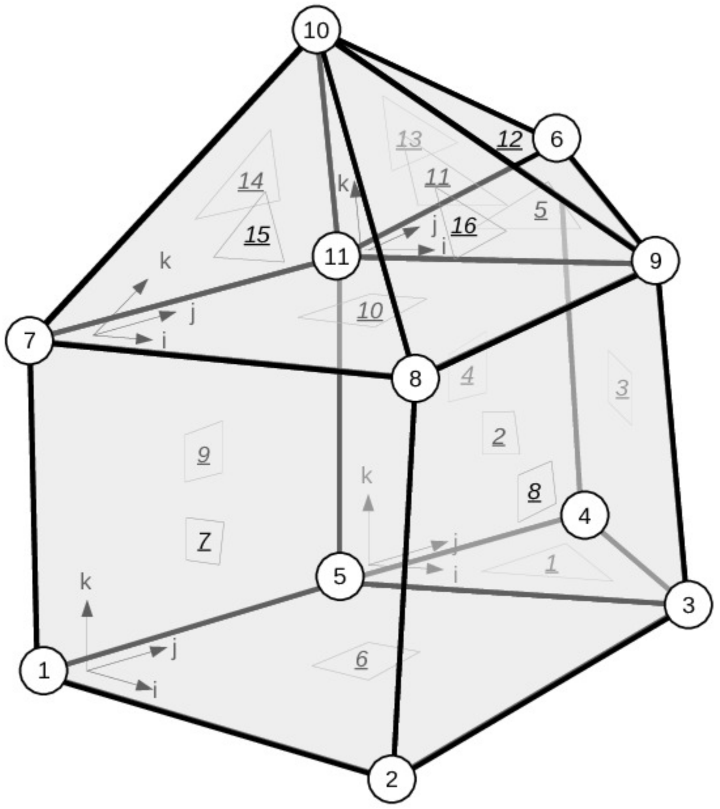


Fig. 1.1: Example 3D mesh with unique node IDs (circles) and unique side IDs (underline) and element-local coordinate system.

1.2.3.2 Element Information (ElemInfo)

Table 1.6: Element Information

Name in file:	**ElemInfo**
Type:	INTEGER, Size: Array(1:6,1:**nElems***)
Description:	Array containing elements, one element per row, **row number is elemID**.

The example mesh Fig. 1.1 with 4 elements is summarized in table Table 1.7. The example shows the four different elements (prism/hexahedron/tetrahedra/pyramid), the prism and hexa are in zone 1 and the tet and the pyramid in zone 2. A detailed list of the element type encoding is found in Section *Element Types*.

Table 1.7: **ElemInfo** array for example 3D mesh with 4 elements.

	Element Type	*Zone*	*offsetIndSIDE*	*lastIndSIDE*	*offsetIndNODE*	*lastIndNODE*
1	116	1	0	5	0	6
2	118	1	5	11	6	14
3	104	2	11	15	14	18
4	115	2	15	20	18	23

Table 1.8: **ElemInfo** definitions.

Element Type:	Encoding for element type, see Section *Element Types*.
Zone:	Element group number.
offsetInd-SIDE/lastIndSIDE:	Each element has a range of sides in the **SideInfo** array.
off-setIndNODE/lastIndNO	Each element has a range of node coordinates in the **NodeCoords** array and **GlobalN-odeIDs** array for unique indices.

The range and the size are always defined as: *Range=[offset+1,last], Size=last-offset*

1.2.3.3 FEM Element Information (FEMElemInfo)

This array will only exist if `FEMConnect="ON"` (hopr parameterfile flag `generateFEMconnectivity=T`).

Table 1.9: FEM Element Information

Name in file:	**FEMElemInfo**
Type:	INTEGER, Size: Array(1:4,1:**nElems***)
Description:	Array containing elements, one element per row, **row number is elemID**.

The example mesh Fig. 1.1 with 4 elements is summarized in table Table 1.7. The example shows the four different elements (prism/hexahedron/tetrahedra/pyramid), the prism and hexa are in zone 1 and the tet and the pyramid in zone

2. A detailed list of the element type encoding is found in Section *Element Types*.

Table 1.10: **FEMElemInfo** array for example 3D mesh with 4 elements.

	offsetIndEDGE	*lastIndEDGE*	*offsetIndVERTEX*	*lastIndVERTEX*
1	0	9	0	6
2	9	21	6	14
3	21	27	14	18
4	27	35	18	23

Table 1.11: **FEMElemInfo** definitions.

offsetIndEDGE/lastIndEDGE:	Each element has a range of edges in the **EdgeInfo** array.
offsetIndVERTEX/lastIndVERTEX:	Each element has a range of edges in the **VertexInfo** array.

The range and the size are always defined as: *Range=[offset+1,last], Size=last-offset*

1.2.3.4 Side Information (SideInfo)

Table 1.12: Side Information

Name in file:	**SideInfo**
Type:	INTEGER, Size: Array(1:6,1:**nSides***)
Description:	Side array, all information of one element is stored continuously (CGNS ordering, \rf{fig:CGNS}) in range 'offsetIndSIDE+1:lastIndSIDE' from **ElemInfo**.

The **SideInfo** array for the example mesh Fig. 1.1 with 4 elements is given in table Table 1.13.

Table 1.13: **SideInfo** array for example 3D mesh with 4 elements.

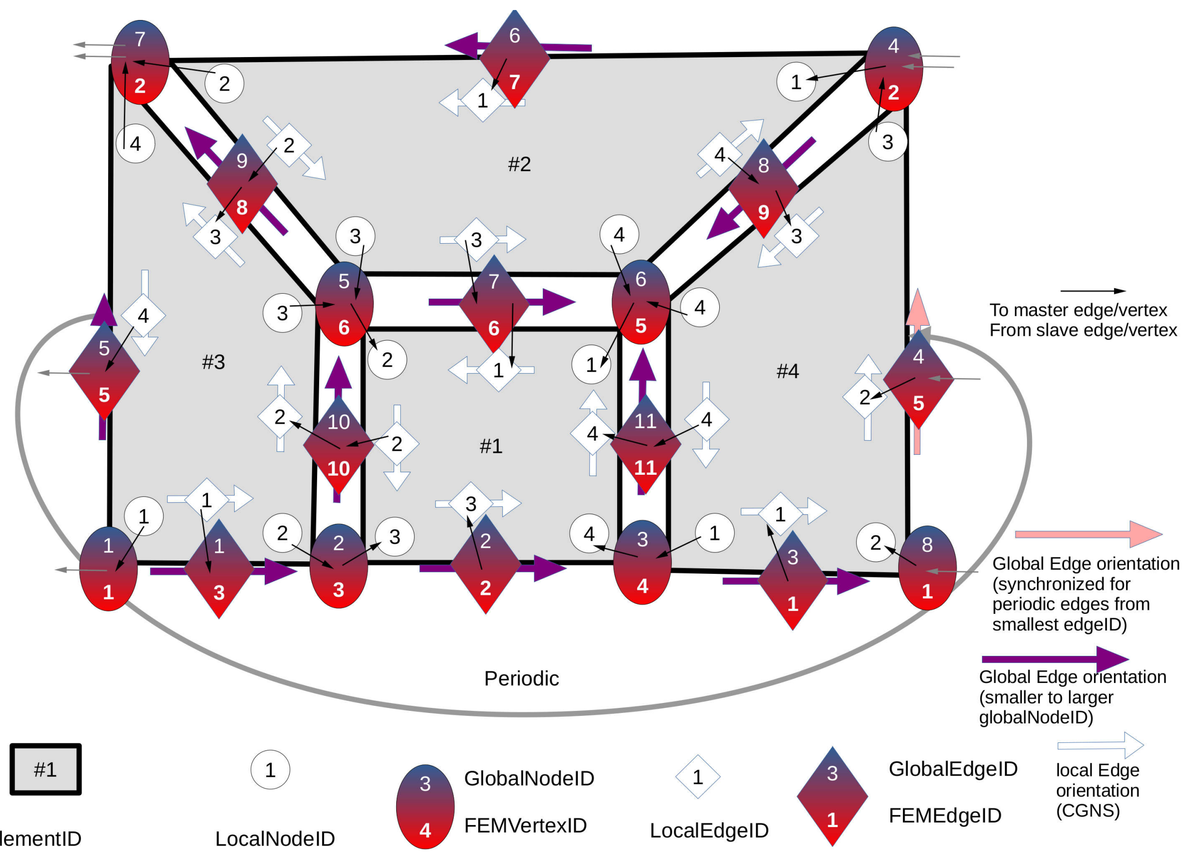
	Side-Type	Global-SideID	nbElemID	10*nbLoc-Side+Flip	BCID	[#El-emID,locSideID]	in **ElemInfo**
1	3	1	0	0	1	[#1,1]	[(offsetInd-SIDE,1)+1]
2	14	2	2	43	0	[#1,2]	
3	14	3	0	0	3	[#1,3]	
4	14	4	0	0	4	[#1,4]	
5	3	5	3	12	0	[#1,5]	[(lastIndSIDE,1)]
6	14	6	0	0	1	[#2,1]	[(offsetInd-SIDE,2)+1]
7	14	7	0	0	2	[#2,2]	
8	14	8	2	50	3	[#2,3]	
9	14	-2	1	23	0	[#2,4]	
10	14	9	2	30	4	[#2,5]	
11	14	10	4	14	0	[#2,6]	[(lastIndSIDE,2)]
12	3	-5	1	52	0	[#3,1]	[(offsetInd-SIDE,3)+1]
13	3	11	4	42	0	[#3,2]	
14	3	12	0	0	3	[#3,3]	
15	3	13	0	0	4	[#3,4]	[(lastIndSIDE,3)]
16	14	-10	2	61	0	[#4,1]	[(offsetInd-SIDE,4)+1]
17	3	15	0	0	2	[#4,2]	
18	3	16	0	0	3	[#4,3]	
19	3	-11	3	22	0	[#4,4]	
20	3	14	0	0	4	[#4,5]	[(lastIndSIDE,4)]

Table 1.14: **SideInfo** definitions

Side-Type:	Side type encoding, the number of corner nodes is the last digit (triangle/quadrangle), more details see Section *Element Types*.
Glob-al-SideID:	Unique global side identifier, can be directly used as MPI tag: it is negative if the side is a slave side (a master and a slave side is defined for side connections).
nbElemIl	ElemID of neighbor element ($= 0$ for no connection). This helps to quickly build up element connections, for local (inside local element range) as well as inter-processor element connections.
10*nbLoc·Side+Fli*	first digit : local side of the connected neighbor element$\in [1, \ldots, 6]$, last digit: Orientation between the sides (flip $\in [0, \ldots, 4]$), see Section *Element Connectivity*.
BCID:	Refers to the row index of the Boundary Condition List in **BCNames/BCType** array ($\in [1, \ldots$ `nBCs`]). $= 0$ for inner sides. Note that $\neq 0$ for periodic and inner boundary conditions, while nbElemID and nbLocSide+Flip are given, see Section *Boundary Conditions*.

1.2.3.5 Edge Information (EdgeInfo)

These arrays will only exist if `FEMConnect="ON"` (hopr parameterfile flag `generateFEMconnectivity=T`).



#1
#2
#3
#4
Periodic
To master edge/vertex
From slave edge/vertex
Global Edge orientation (synchronized for periodic edges from smallest edgeID)
Global Edge orientation (smaller to larger globalNodeID)
ElementID
LocalNodeID
GlobalNodeID
FEMVertexID
LocalEdgeID
GlobalEdgeID
FEMEdgeID
local Edge orientation (CGNS)

Fig. 1.2: Example 2D mesh with periodic BC, local, unique node IDs and **FEMVertexID** (circles,ellipses) and local, unique edge IDs and their FEMEdgeIDs (trapezoid) Arrows for edge orientation

The **EdgeInfo** array includes the `FEMEdgeID` of each local element edge in the same order as the CGNS edges as well as the `offsetIndEDGEConnect` and the `lastIndEDGEConnect` which refer to the corresponding position on the additional `EdgeConnectInfo` array. Here, the `nbElemID` as well as the `localEdgeID` in the corresponding `nbElemID` are saved.

Therefore, the multiplicity is given as `multiplicity=lastEdgeConnect - offSetEdgeConnect+1`.

Table 1.15: Edge Information

Name in file:	**EdgeInfo**
Type:	INTEGER, Size: Array(1:3,1:**nEdges***)
Description:	Edge array, all information of one element is stored continuously (CGNS ordering, \rf{fig:CGNS}) in the range ‘offsetIndEDGE+1:lastIndEDGE’ from **FEMElemInfo**.

The **EdgeInfo** array for the example mesh Fig. 1.2 with 4 elements is given in table Table 1.16.

Table 1.16: **EdgeInfo** array for example 2D mesh with 4 elements.

	(+/- orientation)FEMEdgeID	offsetIndEDGEConnect	LastIndEDGEConnect	[#ElemID,locEdgeID]	[in **FEMElemInfo**]
1	- 6	0	1	[#1,1]	[(offsetIndEDGE,1)+1]
2	- 10	1	2	[#1,2]	
3	+ 2	2	2	[#1,3]	
4	+ 11	2	3	[#1,4]	[(lastIndEDGE,1)]
5	+ 7	3	3	[#2,1]	[(offsetIndEDGE,2)+1]
6	- 8	3	4	[#2,2]	
7	+ 6	4	5	[#2,3]	
8	- 9	5	6	[#2,4]	[(lastIndEDGE,2)]
9	+ 3	6	6	[#3,1]	[(offsetIndEDGE,3)+1]
10	+ 10	6	7	[#3,2]	
11	+ 8	7	8	[#3,3]	
12	- 5	8	9	[#3,4]	[(lastIndEDGE,3)]
13	+ 1	9	9	[#4,1]	[(offsetIndEDGE,4)+1]
14	+ 5	9	10	[#4,2]	
15	+ 9	10	11	[#4,3]	
16	- 11	11	12	[#4,4]	[(lastIndEDGE,4)]

Table 1.17: **EdgeInfo** definitions

FEMEdgeID:	Topologically unique global edge ID, includes periodicity. Sign refers to the local to global edge orientation (+ is same / – is opposite)
offsetIndEDGEConnect/lastIndEDGEConnect:	Each local element edge has a range of neighbor element edges in the **EdgeConnectInfo** array

Table 1.18: **EdgeConnectInfo** array for example 2D mesh with 4 elements.

	(+/- master/slave) nbElemID	(+/- orientation)nbLocEdgeID	[#ElemID,locEdgeID,FEMEdge	[in **EdgeInfo**]
1	+ 2	+ 3	[#1,1,6]	[(offsetIndEDGEConnect, 1)+1]
2	- 3	+ 2	[#1,2,10]	[(offsetIndEDGEConnect, 2)+1]
3	+ 4	- 4	[#1,4,11]	[(offsetIndEDGEConnect, 4)+1]
4	- 3	+ 3	[#2,2,8]	[(offsetIndEDGEConnect, 6)+1]
5	- 1	- 1	[#2,3,6]	[(offsetIndEDGEConnect, 7)+1]
6	- 4	+ 3	[#2,4,9]	[(offsetIndEDGEConnect, 8)+1]
7	+ 1	- 2	[#3,2,10]	[(offsetIndEDGEConnect,10)+1]
8	+ 2	- 2	[#3,3,8]	[(offsetIndEDGEConnect,11)+1]
9	- 4	+ 2	[#3,4,5]	[(offsetIndEDGEConnect,12)+1]
10	+ 3	- 4	[#4,2,5]	[(offsetIndEDGEConnect,14)+1]
11	+ 2	- 4	[#4,3,9]	[(offsetIndEDGEConnect,15)+1]
12	- 1	+ 4	[#4,4,11]	[(offsetIndEDGEConnect,16)+1]

Table 1.19: **EdgeConnectInfo** definitions

nbElemID:	element ID of connected element via the edge. Sign refers if the neighbor edge is master or slave (+ master / – slave)
	from the master slave information, the master/slave of the elements' edge can be deduced
nbLocEdgeID:	local Edge ID in neighbor element. Sign refers to the local to global edge orientation of neighbor edge (+ is same / – is opposite)

1.2.3.6 Vertex Information (VertexInfo)

These arrays will only exist if `FEMConnect="ON"` (hopr parameterfile flag `generateFEMconnectivity=T`).

The **VertexInfo** array includes the `FEMVertexID` of each local element vertex in the same order as the CGNS corners as well as the `offsetIndVERTEXConnect` and the `lastIndVERTEXConnect` which refer to the corresponding position in the additional **VertexConnectInfo** array. Here, the `nbElemID` as well as the `localNodeID` in the corresponding `nbElemID` are saved.

Therefore, the multiplicity is given as `multiplicity = lastIndVERTEXConnect- offsetIndVERTEXConnect + 1.`

```
{table} Vertex Information
---
name: tab:vertex_info
---
|               |                                                                  ␣
↪                                  |
|      :---     |                                                    :---          ␣
↪                                  |
| Name in file: |                                              **VertexInfo**       ␣
↪                                  |
|     Type:     |                               INTEGER, Size: Array(1:3,
↪1:**nVertices**$^*$)                             |
|  Description: | Vertex array, all information of one element is a  stored continuously␣
↪(CGNS ordering, \rf{fig:CGNS})    |
|               |    in the  range 'offsetIndVERTEX+1:lastIndVERTEX' from␣
↪**FEMElemInfo**.                                  |
```

Table 1.20: **VertexInfo** array for example 2D mesh with 4 elements.

	FEMVertexID	offsetIndVERTEXConnect	lastIndVERTEXConnect	[#ElemID,locVertexID]	[*in FEMElemInfo*]
1	5	0	2	[#1,1]	[(offsetIndVERTEX,1)+1]
2	6	2	4	[#1,2]	
3	3	4	5	[#1,3]	
4	4	5	6	[#1,4]	[(lastIndVERTEX,1)]
5	2	6	9	[#2,1]	[(offsetIndVERTEX,2)+1]
6	2	9	12	[#2,2]	
7	6	12	14	[#2,3]	
8	5	14	16	[#2,4]	[(lastIndVERTEX,2)]
9	1	16	17	[#3,1]	[(offsetIndVERTEX,3)+1]
10	3	17	18	[#3,2]	
11	6	18	20	[#3,3]	
12	2	20	23	[#3,4]	[(lastIndVERTEX,3)]
13	4	23	24	[#4,1]	[(offsetIndVERTEX,4)+1]
14	1	24	25	[#4,2]	
15	2	25	28	[#4,3]	
16	5	28	30	[#4,4]	[(lastIndVERTEX,4)]

Table 1.21: **VertexInfo** definitions

FEMVertexID:	Topologically unique global vertex ID, includes periodicity (needed for a FEM solver)
offsetIndVERTEXConnect/lastIndVERTEXConnect:	Each local element vertex has a range of neighbor element edgvertices in the **VertexConnectInfo** array.

Table 1.22: VertexConnect Information

Name in file:	**VertexConnectInfo**
Type:	INTEGER, Size: Array(1:2,1:nFEMVertexConnections)
Description:	Array of connected vertices, all information of one vertex is stored continuously in the range `offsetIndVERTEXConnect+1:lastIndVERTEXConnect` in **VertexInfo**

Table 1.23: **VertexConnecInfo** array for example mesh with 4 elements.

	(+/- master/slave) nbElemID	localNodeID	[#ElemID,locVertexID,FEMVertexID]	[in **VertexInfo**]
1	- 4	4	[#1,1,5]	[(offsetIndVERTEXConnect,1)+1]
2	- 2	4	[#1,1,5]	[(lastIndVERTEXConnect,1)]
3	- 2	3	[#1,2,6]	[(offsetIndVERTEXConnect,2)+1]
4	- 3	3	[#1,2,6]	[(lastIndVERTEXConnect,2)]
5	- 3	2	[#1,3,3]	[(offsetIndVERTEXConnect,3)+1]
6	- 4	1	[#1,4,4]	[(offsetIndVERTEXConnect,4)+1]
7	- 4	3	[#2,1,2]	[(offsetIndVERTEXConnect,5)+1]
8	- 3	4	[#2,1,2]	
9	- 2	2	[#2,1,2]	[(lastIndVERTEXConnect,5)]
10	- 4	3	[#2,2,2]	[(offsetIndVERTEXConnect,6)+1]
11	- 3	4	[#2,2,2]	
12	+ 2	1	[#2,2,2]	[(lastIndVERTEXConnect,6)]
13	+ 1	2	[#2,3,6]	[(offsetIndVERTEXConnect,7)+1
14	- 3	3	[#2,3,6]	[(lastIndVERTEXConnect,7)]
15	+ 1	1	[#2,4,5]	[(offsetIndVERTEXConnect,8)+1
16	- 4	4	[#2,4,5]	[(lastIndVERTEXConnect,8)]
17	+ 4	2	[#3,1,1]	[(offsetIndVERTEXConnect,9)+1
18	+ 1	3	[#3,2,3]	[(offsetIndVERTEXConnect,10)+1
19	+ 1	2	[#3,3,6]	[(offsetIndVERTEXConnect,11)+1
20	- 2	3	[#3,3,6]	[(lastIndVERTEXConnect,11)]
21	- 2	2	[#3,4,2]	[(offsetIndVERTEXConnect,12)+1
22	+ 2	1	[#3,4,2]	
23	- 4	3	[#3,4,2]	[(lastIndVERTEXConnect,12)]
24	+ 1	4	[#4,1,4]	[(offsetIndVERTEXConnect,13)+1
25	- 3	1	[#4,2,1]	[(offsetIndVERTEXConnect,14)+1
26	+ 2	1	[#4,3,2]	[(offsetIndVERTEXConnect,15)+1
27	- 2	2	[#4,3,2]	
28	- 3	4	[#4,3,2]	[(lastIndVERTEXConnect,15)]

continues on next page

Table 1.23 – continued from previous page

	(+/- master/slave) nbElemID	localNodeID	[#ElemID,locVertexID,FEMVertexID]	[in **VertexInfo**]
29	+ 1	1	[#4,4,5]	[(offsetIndVERTEXConnect,16)+1
30	- 2	4	[#4,4,5]	[(lastIndVERTEXConnect,16)]

Table 1.24: **VertexConnectInfo** definitions

nbElemID:	element ID of connected element via the vertex. Sign refers if the neighbor vertex is master or slave (+ master / - slave)
	from the master slave information, the master/slave of the elements' vertex can be deduced
nbLocVertexID:	local vertex ID in neighbor element.

1.2.3.7 Node Coordinates and Global Index

Table 1.25: NodeCoords

Name in file:	**NodeCoords**
Type:	REAL \quad Size: Array(1:3,1:**nNodes***)
Description:	The coordinates of the nodes of the element, as a set for each element.
	offsetIndNODE/lastIndNODE in **ElemInfo** refers to the row index of one set of element nodes.

Table 1.26: GlobalNodeIDs

Name in file:	**GlobalNodeIDs**
Type:	INTEGER \quad Size: Array(1:**nNodes***)
Description:	The unique global node identifier corresponding to the node at the same array position in **NodeCoords**.

The node list contains the high order nodes of the element, so the number of nodes per element depends on the polynomial degree of the element mapping N_{geo}. From this list, the corner nodes can be extracted. The details of the node ordering are explained in Section *Element High Order Nodes*. It is important to note that in the case of $N_{geo} = 1$, our node ordering does NOT correspond to the CGNS corner node ordering for pyramids and hexahedra. Note that the nodes are multiply stored because of the parallel I/O, and therefore the GlobalNodeID is needed for a unique node indexing.

The **NodeCoords**and **GlobalNodeIDs** array for the example mesh Fig. 1.1 with 4 elements is given in table Table 1.27. The node ordering is explained in Section *Element High Order Nodes*.

Table 1.27: **NodeCoords** and **GlobalNodeIDs** array for the example mesh.

NodeCoords	**GlobalNodeIDs**	in **ElemInfo**
$(x,y,z)_5$	5	(offsetIndNODE,1)+1
$(x,y,z)_3$	3	
$(x,y,z)_4$	4	
$(x,y,z)_{11}$	11	
$(x,y,z)_9$	9	
$(x,y,z)_6$	6	(lastIndNODE,1)
$(x,y,z)_1$	1	(offsetIndNODE,2)+1
$(x,y,z)_2$	2	
$(x,y,z)_5$	5	
$(x,y,z)_3$	3	
$(x,y,z)_7$	7	
$(x,y,z)_8$	8	
$(x,y,z)_{11}$	11	
$(x,y,z)_9$	9	(lastIndNODE,2)
$(x,y,z)_{11}$	11	(offsetIndNODE,3)+1
$(x,y,z)_9$	9	
$(x,y,z)_6$	6	
$(x,y,z)_{10}$	10	(lastIndNODE,3)
$(x,y,z)_7$	7	(offsetIndNODE,4)+1
$(x,y,z)_8$	8	
$(x,y,z)_{11}$	11	
$(x,y,z)_9$	9	
$(x,y,z)_{10}$	10	(lastIndNODE,4)

1.2.3.8 Boundary Conditions

Table 1.28: BCNames

Name in file:	**BCNames**
Type:	STRING, \quad Size: Array(1:*nBCs*)
Description:	User-defined list of boundary condition names.

Table 1.29: BCType

Name in file:	**BCType**
Type:	INTEGER, \quad Size: Array(1:4,1:**nBCs**)
Description:	User-defined array of 4 integers per boundary condition.

The boundary conditions are completely defined by the user. Each BCID from the **SideInfo** array refers to the **position** of the boundary condition in the **BCNames list**. An additional 4 integer code in **BCType** is available for user-defined attributes.

Table 1.30: **BCNames** and **BCType** array for the example mesh, representing a list of boundary condition names.

Ind	Boundary Conditions Name:	**BCType**
1	lowerWall	(4,0,0,0)
2	Inflow	(2,0,0,0)
3	OutflowRight	(10,0,0,0)
4	OutflowLeft	(8,0,0,0)

The **BCType** array consists of the following entries, of which some are specific to HOPR:

Table 1.31: **BCType** = (\emph{ BoundaryType, CurveIndex, StateIndex, PeriodicIndex})

Boundc ary-Type:	Actual type of boundary condition (e.g. inflow, outflow, periodic). {**Reserved values**:} BoundaryType=1 is reserved for periodic boundaries and BoundaryType=100 is reserved for "inner" boundaries or "analyze sides". For these two cases the sides in the SideInfo array will have a neighbor side/element/flip specified, all other sides with BCs are not connected!
Curve dex:	Geometry tag used to distinguish between multiple BCs of the same type, e.g. to specify the original CAD surface belonging to the mesh side. Also used to control some mesh curving features, sides with CurveIndex>0 are curved, while sides with CurveIndex=0 are mostly (bi-) linear.
StateI dex:	Specifies the index of a reference state to be used inside the solver. This value is completely used-defined and will not be used/checked/modified by HOPR.
Pe-ri-od-icIn-dex:	Only relevant for periodic sides, ignored for others. For periodic connections two boundary conditions are required, having the same absolute PeriodicIndex, one with positive, the other with negative sign.

1.2.4 Parallel Read-in

The overall parallel read-in process is depicted in Fig. 1.3. The Algorithms Fig. 1.4, Fig. 1.5, Fig. 1.6 describe how to open and close a HDF5 file and read the file attributes.

Each parallel process (MPI rank) has to read a contiguous element range, which will be basically defined by dividing the total number of elements by the number of domains, already leading to the domain decomposition. The element distribution is computed locally on each rank. Follow Fig. 1.7 for an equal distribution of an arbitrary number of elements on an arbitrary number of domains/ranks. The algorithm is easy to extend to account for different element weights. The element distribution is saved in the *offsetElem* array of size *0:nDomains*. The element range for each domain (*mydom*∈*[0:nDomains-1]*) is then

ElementRange(myDom)=[offsetElem(myDom)+1;offsetElem(myDom+1)]

Note that the *offsetElem* array will have the information of all element ranges of all ranks, which is very helpful for building the inter-domain mesh connectivity to quickly find neighbor elements on other domains/ranks.

Using the number of local elements and the offset, we read the non-overlapping sub-arrays of the **ElemInfo** array in parallel (using hyperslab HDF5 commands, see Fig. 1.8), which will assign a continuous sub-array of element informations for each rank. With the local element informations, we easily compute the offset and size of sub-arrays for the side data (**SideInfo**) and node data (**NodeCoords**), by computing

firstElem =	*offsetElem(myDom)+1*
lastElem =	*offsetElem(myDom+1)*
firstSide =	**ElemInfo** *(offsetIndSIDE,firstElem)+1*
lastSide =	**ElemInfo** *(lastIndSIDE,lastElem)*
firstNode =	**ElemInfo** *(offsetIndNODE,firstElem)+1*
lastNode =	**ElemInfo** *(lastIndNODE,lastElem)*

and again read the non-overlapping sub-arrays in parallel. Now element geometry is easily built locally. Local element connectivities would only have neighbor element indices inside the local element range and can directly be assigned. The overall read-in process is summarized in Fig. 1.9.

For the inter-domain connectivity, we have to find the domain containing the neighbor element. A quick search is done with a bisection of the *offsetElem* array, since element ranges are monotonically increasing, see Fig. 1.10.

Finally, we group the sides connected to each neighbor domain and sort the sides along the global side index (known from **SideInfo**). This creates the same side list on both domains without any communication. If an orientation of the side link is needed, the side is always marked either master or slave (positive or negative global side index).

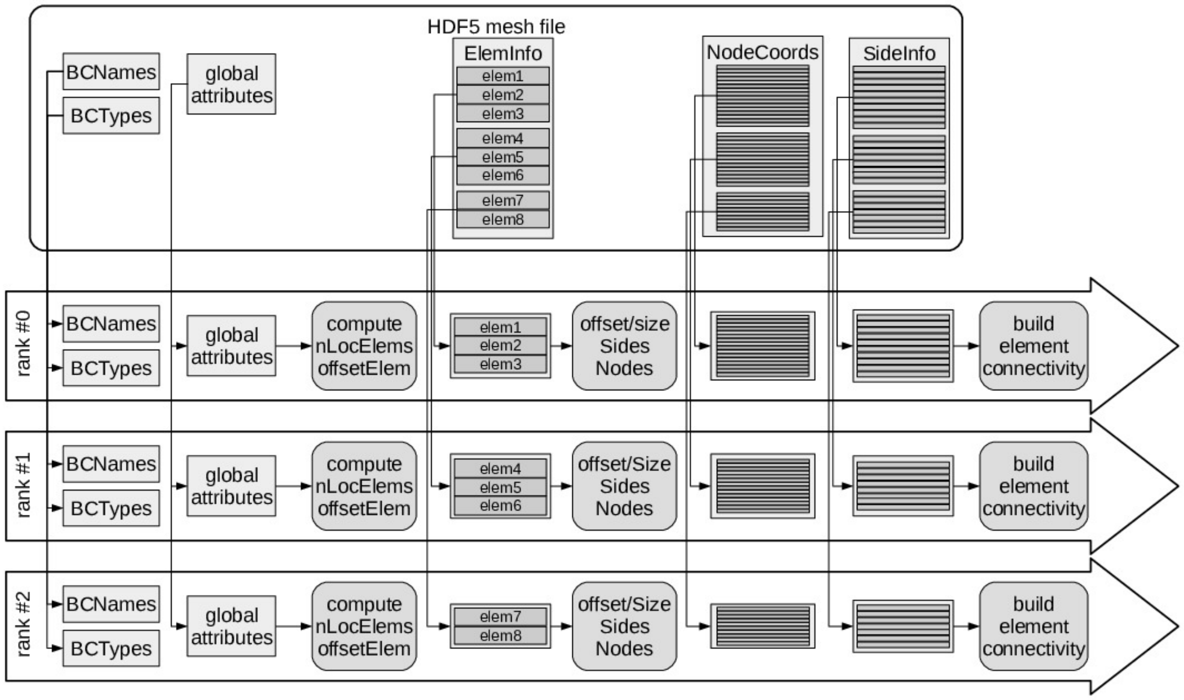
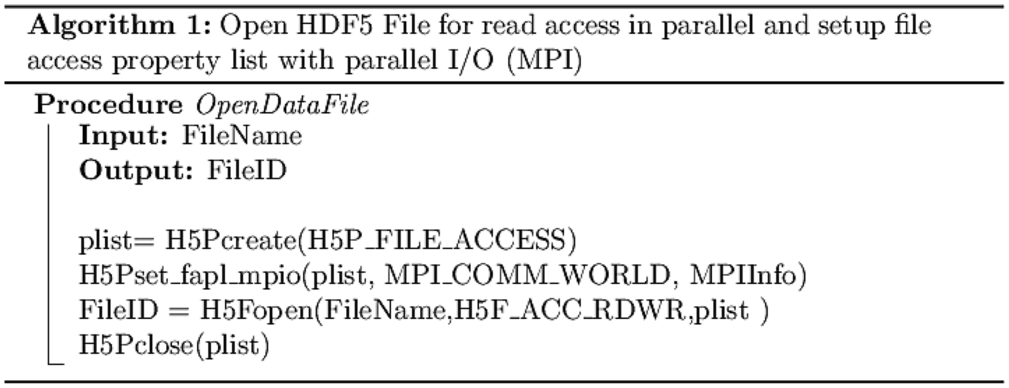


Fig. 1.3: Parallel read-in process of the HDF5 mesh file, exemplary with 8 elements on 3 MPI ranks **domains**.



```
Algorithm 1: Open HDF5 File for read access in parallel and setup file
access property list with parallel I/O (MPI)

Procedure OpenDataFile
    Input: FileName
    Output: FileID

    plist= H5Pcreate(H5P_FILE_ACCESS)
    H5Pset_fapl_mpio(plist, MPI_COMM_WORLD, MPIInfo)
    FileID = H5Fopen(FileName,H5F_ACC_RDWR,plist )
    H5Pclose(plist)
```

Fig. 1.4: Algorithm 1

```
Algorithm 2: Close HDF5 File
Procedure CloseDataFile
    Input: FileID
    Output:

    H5Close(FileID)
```

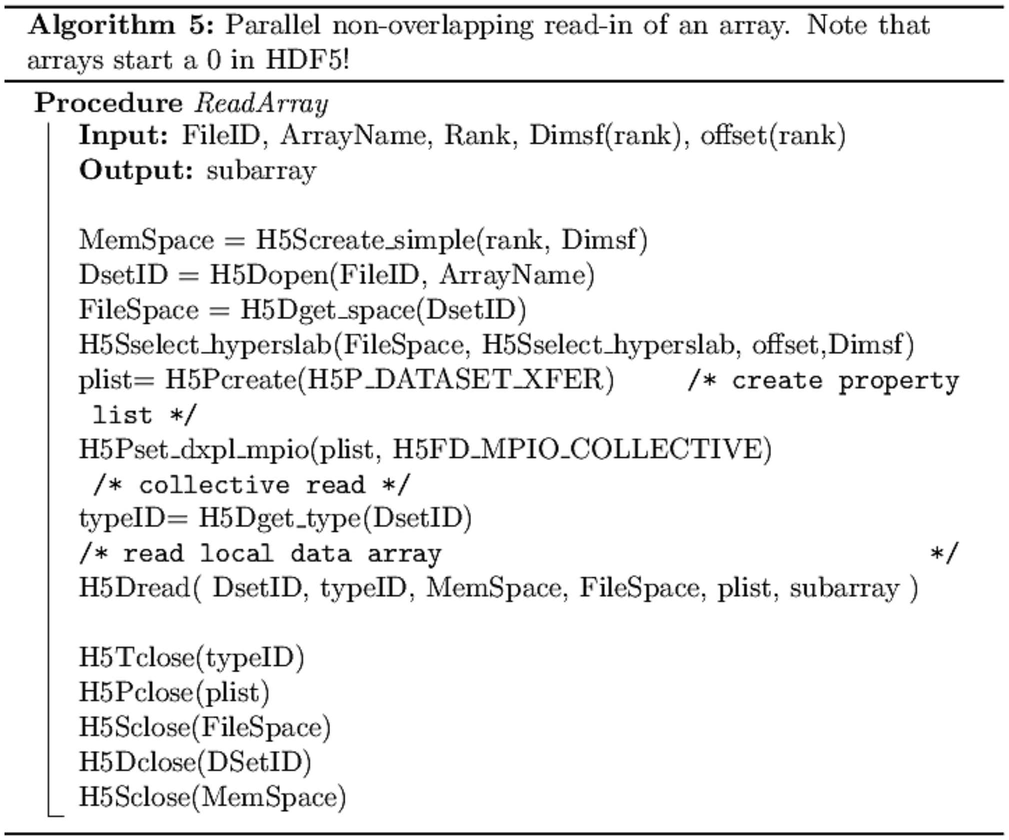
Fig. 1.5: Algorithm 2

```
Algorithm 3: Read attribute from File
Procedure ReadAttribute
    Input: FileID, AttributeName, Dimsf(1)
    Output: attribute

    AttrID = H5Aopen(FileID, AttributeName)
    typeID= H5Aget_type(DsetID)
    H5Dread( AttrID, typeID, attribute )

    H5Tclose(typeID)
    H5Aclose(AttrID)
```

Fig. 1.6: Algorithm 3

```
Algorithm 4: Simple Domain Decomposition: Assign local number of
elements of domain $myDom \in [0; nDomains - 1]$ and element ranges for
all domains
Procedure DomainDecomp
    Input: nElems, nDomains, myDom
    Output: offsetElem(0:nDomains)

    nLocalElems ← nElems/nDomains
    remainElems ← nElems-nLocalElems * nDomains
    for iDom = 0 to nDomains-1 do
        offsetElem(iDom)← iDom * nLocalElems +
         MIN(iDom,remainElems)
    offsetElem(nDomains)←nElems
```

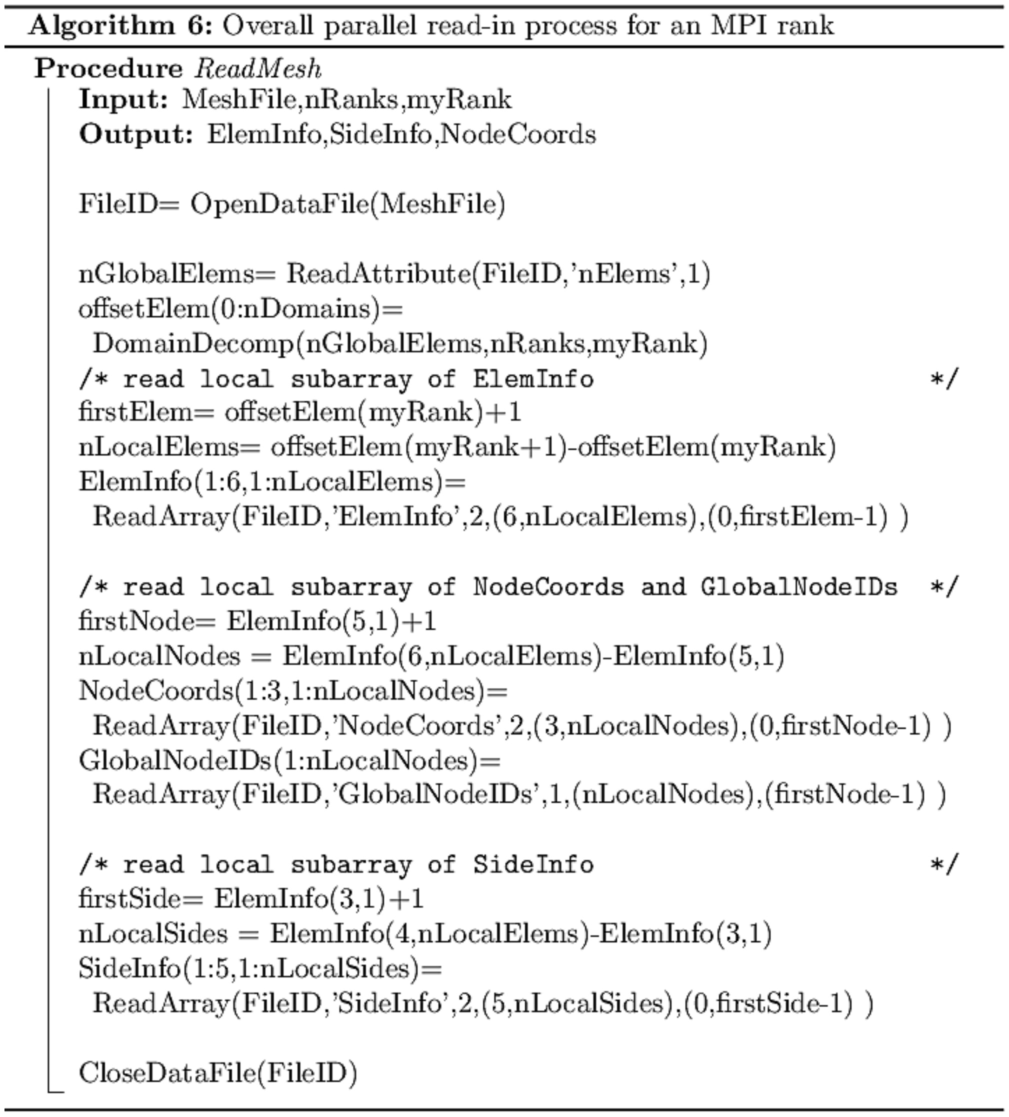
Fig. 1.7: Algorithm 4



Algorithm 5: Parallel non-overlapping read-in of an array. Note that arrays start a 0 in HDF5!

```
Procedure ReadArray
    Input: FileID, ArrayName, Rank, Dimsf(rank), offset(rank)
    Output: subarray

    MemSpace = H5Screate_simple(rank, Dimsf)
    DsetID = H5Dopen(FileID, ArrayName)
    FileSpace = H5Dget_space(DsetID)
    H5Sselect_hyperslab(FileSpace, H5Sselect_hyperslab, offset,Dimsf)
    plist= H5Pcreate(H5P_DATASET_XFER)      /* create property
     list */
    H5Pset_dxpl_mpio(plist, H5FD_MPIO_COLLECTIVE)
     /* collective read */
    typeID= H5Dget_type(DsetID)
    /* read local data array                                    */
    H5Dread( DsetID, typeID, MemSpace, FileSpace, plist, subarray )

    H5Tclose(typeID)
    H5Pclose(plist)
    H5Sclose(FileSpace)
    H5Dclose(DSetID)
    H5Sclose(MemSpace)
```

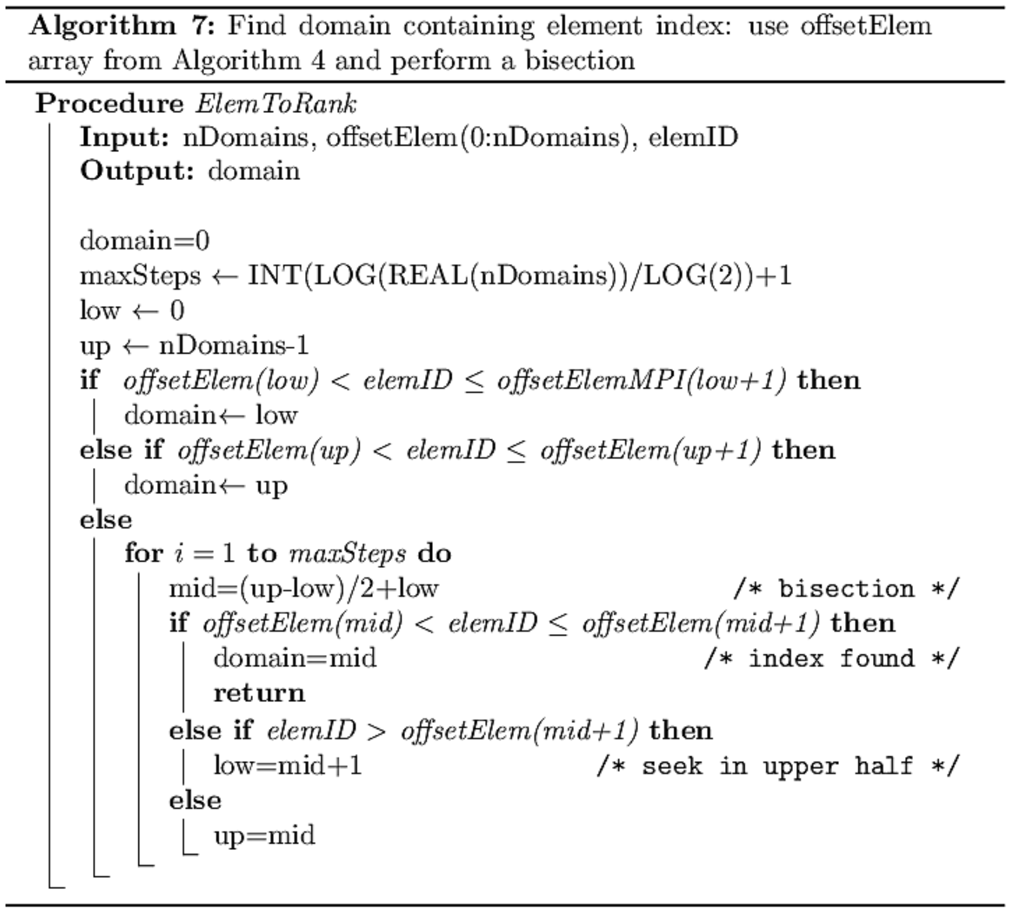
Fig. 1.8: Algorithm 5



Algorithm 6: Overall parallel read-in process for an MPI rank

```
Procedure ReadMesh
    Input: MeshFile,nRanks,myRank
    Output: ElemInfo,SideInfo,NodeCoords

    FileID= OpenDataFile(MeshFile)

    nGlobalElems= ReadAttribute(FileID,'nElems',1)
    offsetElem(0:nDomains)=
     DomainDecomp(nGlobalElems,nRanks,myRank)
    /* read local subarray of ElemInfo                        */
    firstElem= offsetElem(myRank)+1
    nLocalElems= offsetElem(myRank+1)-offsetElem(myRank)
    ElemInfo(1:6,1:nLocalElems)=
     ReadArray(FileID,'ElemInfo',2,(6,nLocalElems),(0,firstElem-1) )

    /* read local subarray of NodeCoords and GlobalNodeIDs    */
    firstNode= ElemInfo(5,1)+1
    nLocalNodes = ElemInfo(6,nLocalElems)-ElemInfo(5,1)
    NodeCoords(1:3,1:nLocalNodes)=
     ReadArray(FileID,'NodeCoords',2,(3,nLocalNodes),(0,firstNode-1) )
    GlobalNodeIDs(1:nLocalNodes)=
     ReadArray(FileID,'GlobalNodeIDs',1,(nLocalNodes),(firstNode-1) )

    /* read local subarray of SideInfo                        */
    firstSide= ElemInfo(3,1)+1
    nLocalSides = ElemInfo(4,nLocalElems)-ElemInfo(3,1)
    SideInfo(1:5,1:nLocalSides)=
     ReadArray(FileID,'SideInfo',2,(5,nLocalSides),(0,firstSide-1) )

    CloseDataFile(FileID)
```

Fig. 1.9: Algorithm 6



Algorithm 7: Find domain containing element index: use offsetElem array from Algorithm 4 and perform a bisection

```
Procedure ElemToRank
    Input: nDomains, offsetElem(0:nDomains), elemID
    Output: domain

    domain=0
    maxSteps ← INT(LOG(REAL(nDomains))/LOG(2))+1
    low ← 0
    up ← nDomains-1
    if offsetElem(low) < elemID ≤ offsetElemMPI(low+1) then
        domain← low
    else if offsetElem(up) < elemID ≤ offsetElem(up+1) then
        domain← up
    else
        for $i = 1$ to maxSteps do
            mid=(up-low)/2+low                          /* bisection */
            if offsetElem(mid) < elemID ≤ offsetElem(mid+1) then
                domain=mid                              /* index found */
                return
            else if elemID > offsetElem(mid+1) then
                low=mid+1                               /* seek in upper half */
            else
                up=mid
```

Fig. 1.10: Algorithm 7

1.2.5 Element Definitions

1.2.5.1 Element Types

The classification of the element types is given in Table 1.32. The last digit is always the number of corner nodes. The classification is geometrically motivated. The element has a linear mapping if $N_{geo} = 1$ and the corner nodes are an affine transformation of the reference element corner nodes, whereas bilinear stands for the general straight-edged element with $N_{geo} = 1$, and non-linear for the high order case $N_{geo} \geq 1$.

For mesh file read-in, only the number of element corner nodes is important to distinguish the 3D elements, since the polynomial degree N_{geo} is globally defined.

Table 1.32: Element type encoding.

ElementType	Index	ElementType	Index	ElementType	Index
Triangle, linear	3	Tetrahedron, linear	104	Prism, bilinear	116
Quad, linear	4	Pyramid, linear	105	Hexahedron, bilinear	118
		Prism, linear	106	Tetrahedron, non-linear	204
Quad, bilinear	14	Hexahedron, linear	108	Pyramid, non-linear	205
Triangle, non-linear	23			Prism, non-linear	206
Quad, non-linear	24	Pyramid, bilinear	115	Hexahedron, non-linear	208

1.2.5.2 Element High Order Nodes

In the arrays **NodeCoords** and **GlobalNodeIDs** (Section *Node Coordinates and Global Index*), the element high order nodes are found as a node list, $1, \ldots, \ell, \ldots M_{\text{elem}}$. The number of nodes for each element is defined by the element type and the polynomial degree N_{geo} of the mapping and is listed in Table 1.33. See Section *Element Corners, Sides* if one needs only the corner nodes of the linear mesh.

Table 1.33: Element node count.

Element Type:	#Corner nodes	#HO nodes (M_{elem})
Triangle	3	$\frac{1}{2}(N_{geo}+1)(N_{geo}+2)$
Quad	4	$(N_{geo}+1)^2$
Tetrahedron	4	$\frac{1}{6}(N_{geo}+1)(N_{geo}+2)(N_{geo}+3)$
Pyramid	5	$\frac{1}{6}(N_{geo}+1)(N_{geo}+2)(2N_{geo}+3)$
Prism	6	$\frac{1}{2}(N_{geo}+1)^2(N_{geo}+2)$
Hexhedron	8	$(N_{geo}+1)^3$

The mapping from the node list to the node position

$$\ell \mapsto (i,j,k) \qquad \ell \in [1; M_{\text{elem}}] \qquad 0 \leq i,j,k \leq N_{geo}$$

is defined by Fig. 1.12 and an example is shown for quadratic mapping in Fig. 1.11. The high order node positions are regular in reference space $-1 \leq (\xi,\eta,\zeta) \leq 1$ and therefore can be easily computed from the (i,j,k) index of the node ℓ by

$$(\xi,\eta,\zeta)_\ell = -1 + \frac{2}{N}(i,j,k)_\ell$$

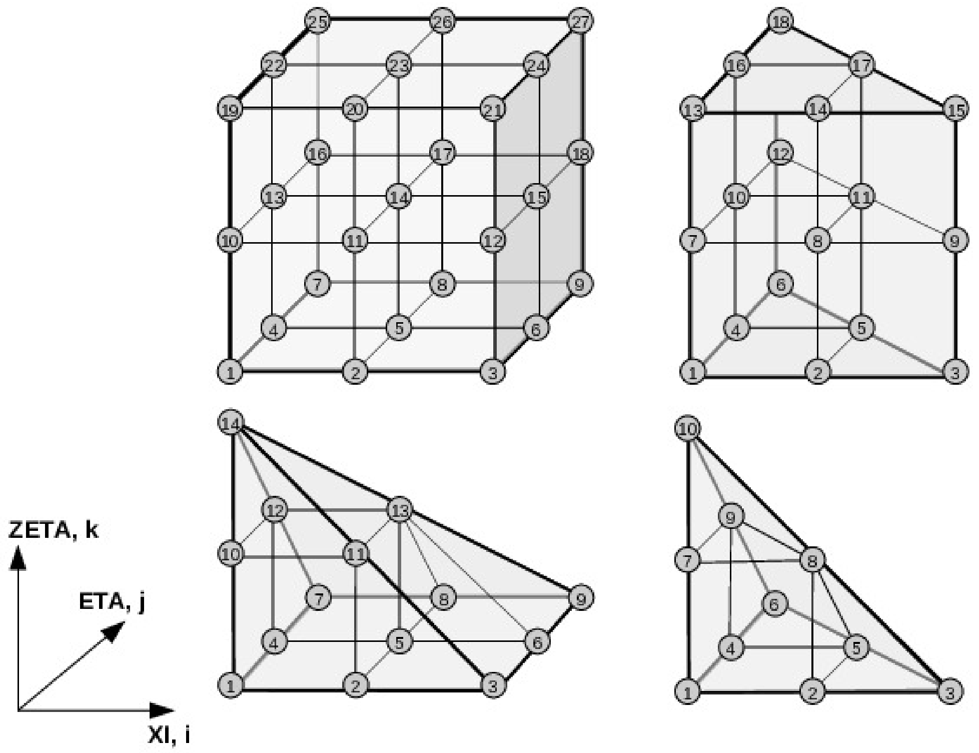


Fig. 1.11: Example of the element high order node sorting from Fig. 1.12 for a quadratic mapping ($N_{geo} = 2$).

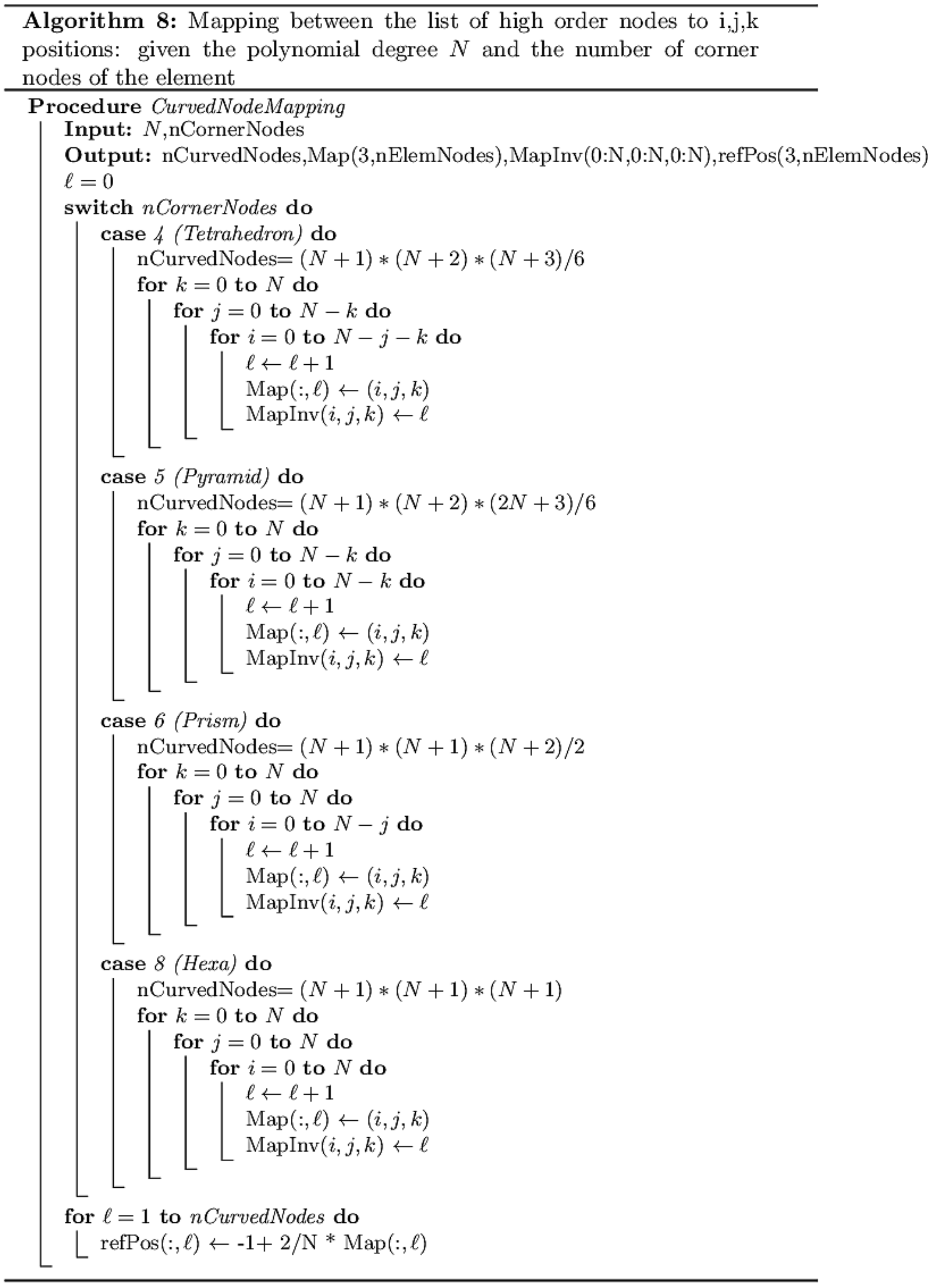
1.2.5.3 Element Corners, Sides

To define the element corner nodes, the side order and side connectivity, we follow the standard from CGNS SIDS (CFD General Notation System, Standard Interface Data Structures, [http://cgns.sourceforge.net/][http://cgns.sourceforge.net/]). The definition is sketched in Fig. 1.13. To get the CGNS corner nodes from the high order node list, follow Fig. 1.14. Note that in the case of $N_{geo} = 1$, the node ordering does **not** correspond to the CGNS corner node ordering for pyramids and hexahedra!

Especially, the CGNS standard defines a local coordinate system of each element side. The side's first node will be the origin, and the remaining nodes are ordered in the direction of the outward pointing normal.

1.2.5.4 Element Connectivity

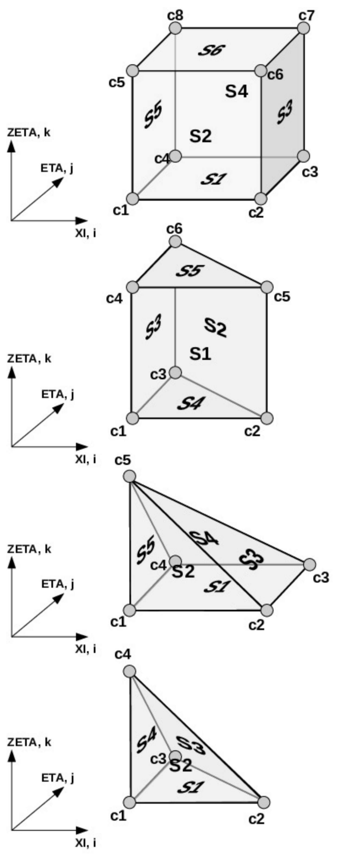
In the **SideInfo** array, we explicitly store the side-to-side connectivity information between elements, consisting of the neighbor element ID, the local side of the neighbor and the orientation, encoded with the variable *flip*. Using the local side system, the orientation between elements boils down to three cases for a triangular element side and four for a quadrilateral element side. The definition is given in Fig. 1.15. Also note that the flip in Table 1.34 is symmetric, having the same value if seen from the neighbor side.



Algorithm 8: Mapping between the list of high order nodes to i,j,k positions: given the polynomial degree N and the number of corner nodes of the element

```
Procedure CurvedNodeMapping
    Input: $N$,nCornerNodes
    Output: nCurvedNodes,Map(3,nElemNodes),MapInv(0:N,0:N,0:N),refPos(3,nElemNodes)
    $\ell = 0$
    switch nCornerNodes do
        case 4 (Tetrahedron) do
            nCurvedNodes= $(N+1)*(N+2)*(N+3)/6$
            for $k = 0$ to $N$ do
                for $j = 0$ to $N - k$ do
                    for $i = 0$ to $N - j - k$ do
                        $\ell \leftarrow \ell + 1$
                        Map$(:,\ell) \leftarrow (i,j,k)$
                        MapInv$(i,j,k) \leftarrow \ell$
        case 5 (Pyramid) do
            nCurvedNodes= $(N+1)*(N+2)*(2N+3)/6$
            for $k = 0$ to $N$ do
                for $j = 0$ to $N - k$ do
                    for $i = 0$ to $N - k$ do
                        $\ell \leftarrow \ell + 1$
                        Map$(:,\ell) \leftarrow (i,j,k)$
                        MapInv$(i,j,k) \leftarrow \ell$
        case 6 (Prism) do
            nCurvedNodes= $(N+1)*(N+1)*(N+2)/2$
            for $k = 0$ to $N$ do
                for $j = 0$ to $N$ do
                    for $i = 0$ to $N - j$ do
                        $\ell \leftarrow \ell + 1$
                        Map$(:,\ell) \leftarrow (i,j,k)$
                        MapInv$(i,j,k) \leftarrow \ell$
        case 8 (Hexa) do
            nCurvedNodes= $(N+1)*(N+1)*(N+1)$
            for $k = 0$ to $N$ do
                for $j = 0$ to $N$ do
                    for $i = 0$ to $N$ do
                        $\ell \leftarrow \ell + 1$
                        Map$(:,\ell) \leftarrow (i,j,k)$
                        MapInv$(i,j,k) \leftarrow \ell$
    for $\ell = 1$ to nCurvedNodes do
        refPos$(:,\ell) \leftarrow$ -1+ 2/N * Map$(:,\ell)$
```

Fig. 1.12: Algorithm 8



CGNS side	Corner Nodes
S1	c1 c4 c3 c2
S2	c1 c2 c6 c5
S3	c2 c3 c7 c6
S4	c3 c4 c8 c7
S5	c1 c5 c8 c4
S6	c5 c6 c7 c8

CGNS Edge	Corner Nodes	CGNS Edge	Corner Nodes
E1	c1,c2	E7	c3,c7
E2	c2,c3	E8	c4,c8
E3	c3,c4	E9	c5,c6
E4	c4,c1	E10	c6,c7
E5	c1,c5	E11	c7,c8
E6	c2,c6	E12	c8,c5

CGNS side	Corner Nodes
S1	c1 c2 c5 c4
S2	c2 c3 c6 c5
S3	c3 c1 c4 c6
S4	c1 c3 c2
S5	c4 c5 c6

CGNS Edge	Corner Nodes	CGNS Edge	Corner Nodes
E1	c1,c2	E6	c3,c6
E2	c2,c3	E7	c4,c5
E3	c3,c1	E8	c5,c6
E4	c1,c4	E9	c6,c4
E5	c2,c5		

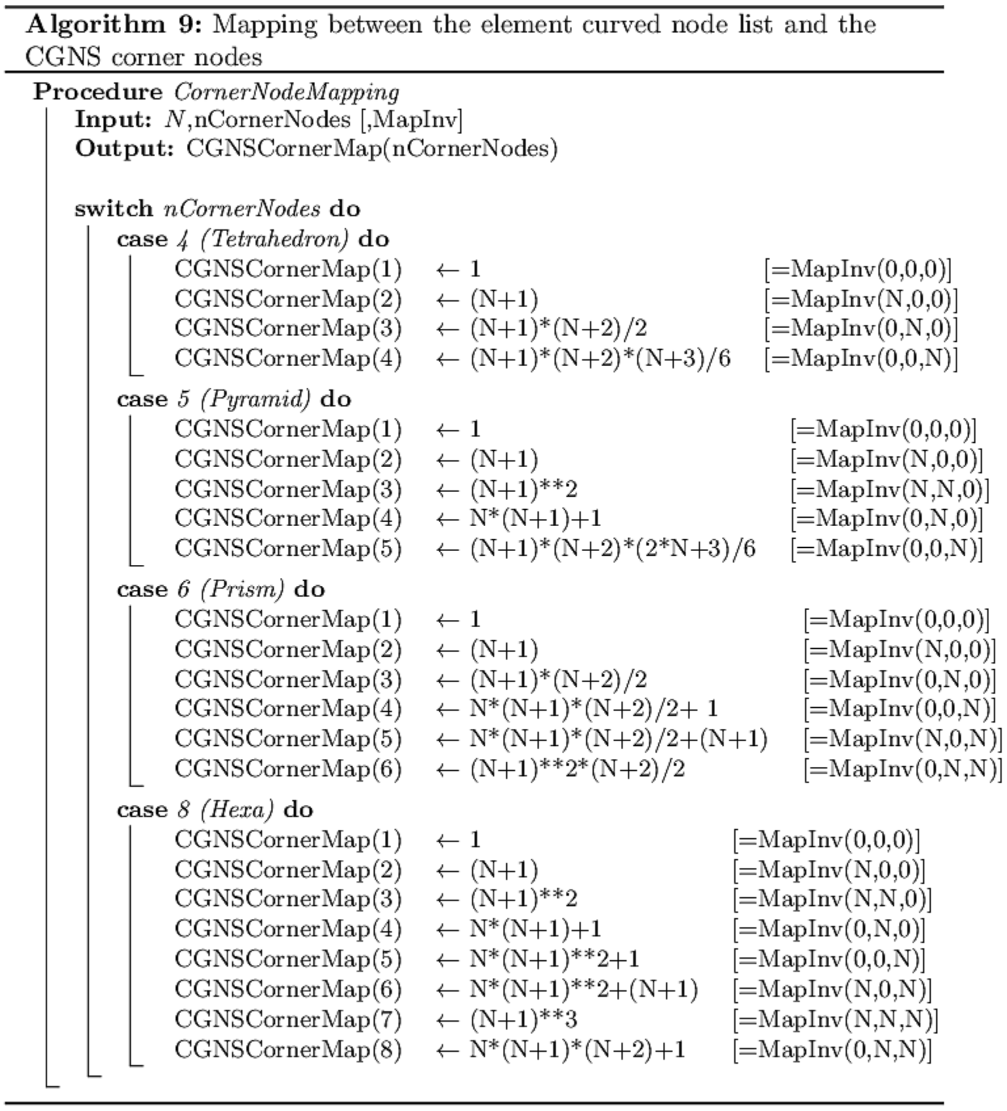
CGNS side	Corner Nodes
S1	c1 c4 c3 c2
S2	c1 c2 c5
S3	c2 c3 c5
S4	c3 c4 c5
S5	c4 c1 c5

CGNS Edge	Corner Nodes	CGNS Edge	Corner Nodes
E1	c1,c2	E5	c1,c5
E2	c2,c3	E6	c2,c5
E3	c3,c4	E7	c3,c5
E4	c4,c1	E8	c4,c5

CGNS side	Corner Nodes
S1	c1 c3 c2
S2	c1 c2 c4
S3	c2 c3 c4
S4	c3 c1 c4

CGNS Edge	Corner Nodes	CGNS Edge	Corner Nodes
E1	c1,c2	E4	c1,c4
E2	c2,c3	E5	c2,c4
E3	c3,c1	E6	c3,c4

Fig. 1.13: Definition of corner nodes, side order and side orientation, from CGNS SIDS.



Algorithm 9: Mapping between the element curved node list and the CGNS corner nodes

```
Procedure CornerNodeMapping
    Input: N,nCornerNodes [,MapInv]
    Output: CGNSCornerMap(nCornerNodes)

    switch nCornerNodes do
        case 4 (Tetrahedron) do
            CGNSCornerMap(1)   ← 1                          [=MapInv(0,0,0)]
            CGNSCornerMap(2)   ← (N+1)                      [=MapInv(N,0,0)]
            CGNSCornerMap(3)   ← (N+1)*(N+2)/2              [=MapInv(0,N,0)]
            CGNSCornerMap(4)   ← (N+1)*(N+2)*(N+3)/6        [=MapInv(0,0,N)]
        case 5 (Pyramid) do
            CGNSCornerMap(1)   ← 1                          [=MapInv(0,0,0)]
            CGNSCornerMap(2)   ← (N+1)                      [=MapInv(N,0,0)]
            CGNSCornerMap(3)   ← (N+1)**2                   [=MapInv(N,N,0)]
            CGNSCornerMap(4)   ← N*(N+1)+1                  [=MapInv(0,N,0)]
            CGNSCornerMap(5)   ← (N+1)*(N+2)*(2*N+3)/6      [=MapInv(0,0,N)]
        case 6 (Prism) do
            CGNSCornerMap(1)   ← 1                          [=MapInv(0,0,0)]
            CGNSCornerMap(2)   ← (N+1)                      [=MapInv(N,0,0)]
            CGNSCornerMap(3)   ← (N+1)*(N+2)/2              [=MapInv(0,N,0)]
            CGNSCornerMap(4)   ← N*(N+1)*(N+2)/2+ 1         [=MapInv(0,0,N)]
            CGNSCornerMap(5)   ← N*(N+1)*(N+2)/2+(N+1)      [=MapInv(N,0,N)]
            CGNSCornerMap(6)   ← (N+1)**2*(N+2)/2           [=MapInv(0,N,N)]
        case 8 (Hexa) do
            CGNSCornerMap(1)   ← 1                          [=MapInv(0,0,0)]
            CGNSCornerMap(2)   ← (N+1)                      [=MapInv(N,0,0)]
            CGNSCornerMap(3)   ← (N+1)**2                   [=MapInv(N,N,0)]
            CGNSCornerMap(4)   ← N*(N+1)+1                  [=MapInv(0,N,0)]
            CGNSCornerMap(5)   ← N*(N+1)**2+1               [=MapInv(0,0,N)]
            CGNSCornerMap(6)   ← N*(N+1)**2+(N+1)           [=MapInv(N,0,N)]
            CGNSCornerMap(7)   ← (N+1)**3                   [=MapInv(N,N,N)]
            CGNSCornerMap(8)   ← N*(N+1)*(N+2)+1            [=MapInv(0,N,N)]
```

Fig. 1.14: Algorithm 9

Table 1.34: Side-to-side connection (*flip*)

flip= 1:	1^{st}	node of neighbor side = 1^{st} node of side
flip= 2:	2^{nd}	node of neighbor side = 1^{st} node of side
flip= 3:	3^{rd}	node of neighbor side = 1^{st} node of side
flip= 4:	4^{th}	node of neighbor side = 1^{st} node of side

1.2.6 Additional Extensions: Hanging Node Interface

For complex geometries it is often desirable to use elements with hanging nodes to provide more geometric flexibility when meshing. As geometric restrictions are most severe for pure hexahedral meshes, the HOPR format supports a limited octree-like topology with hanging nodes for purely hexahedral meshes. The octree topology is implemented as extension to the existing mesh structure. While full octrees permit an element-side to have an arbitrary number of neighbors on various octree levels, our format supports only one octree level difference between element sides with two (anisotropic) and four neighbors, the single types are depicted in Fig. 1.16.

For the connection of two elements over an octree level additional interfaces are required, which are termed mortar interfaces and are depicted in Fig. 1.17. Thereby the big side is denoted big mortar master side, the intermediate sides are denoted small mortar master sides. The sides of the small elements are denoted slave sides. Note that they do not require any information about the mortar interface and therefore the interface is only represented from the big element side in the data format.

The following differences are present for the **ElemInfo** and the **SideInfo** array:

1.2.6.1 Changes to Existing Data Format

- **ElemInfo**: The range of sides defined by *offsetIndSIDE* and *lastIndSIDE* now includes the small mortar master sides for the element that owns the big mortar side.
- **SideInfo**: The field *nbElemID* of the big mortar side defines no connection to the neighbor element, but contains the type of the mortar interface (=1/2/3) from Fig. 1.16 **with negative sign**, to mark that the following sides belong to a mortar interface. The type of the interface defines the number of the small mortar master sides (Type 1 has 4 and Type 2&3 have 2 small master sides).
- **SideInfo**: The list of sides belonging to an element includes the small mortar master sides sorted as exemplified in Table 1.35 and Fig. 1.17.
- **SideInfo**: Only the small mortar masters have a valid *nbElemID*, defining the connection to the adjacent small elements.
- **SideInfo**: (Mortar) Master sides always have flip=0, thus the small element sides are always slave sides.
- **SideInfo**: If the element side belongs to a mortar but with the small mortar slave side, it is marked as such using a *SideType* **with negative sign**.

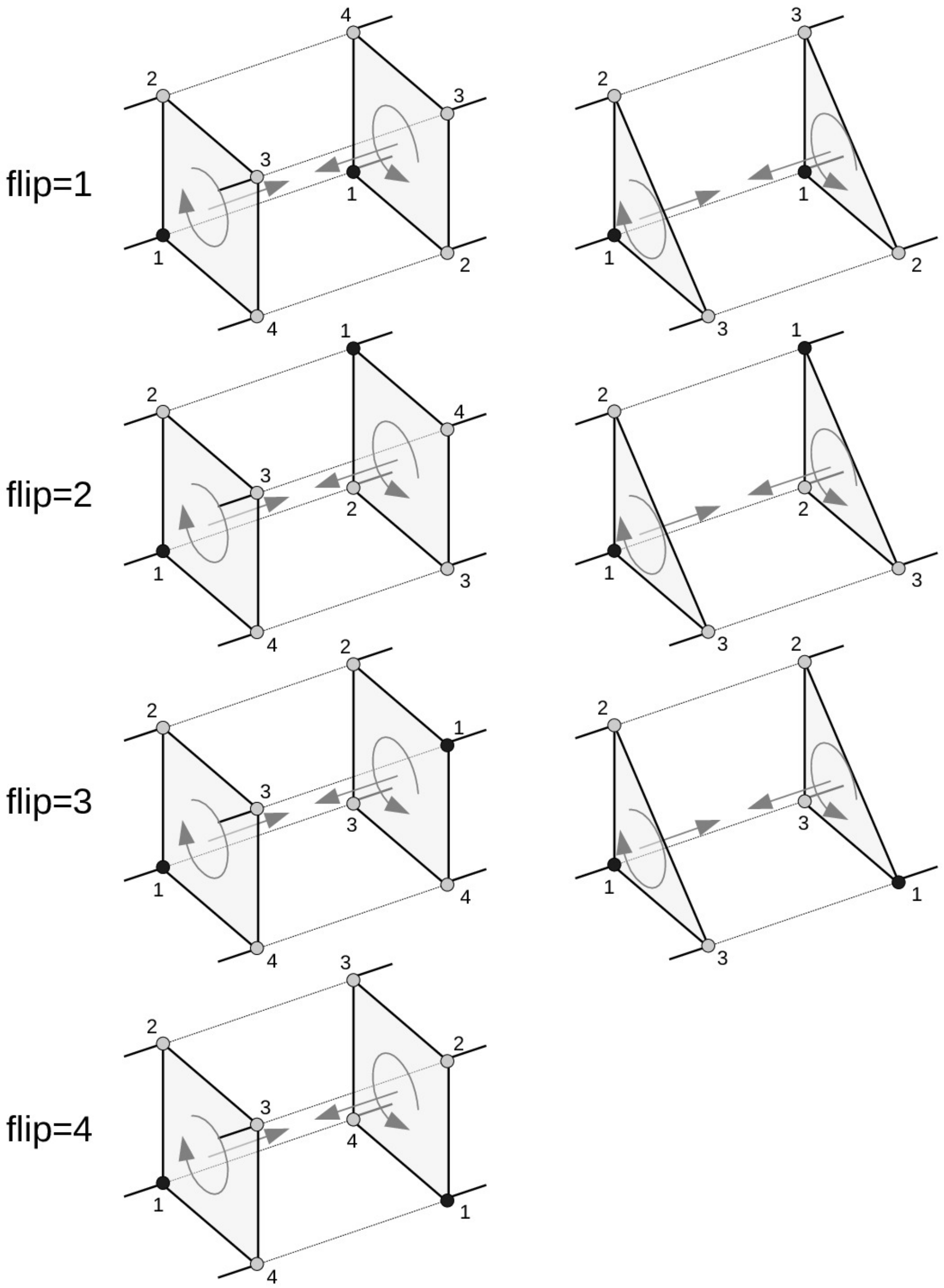


Fig. 1.15: Definition of the orientation of side-to-side connection (*flip*) for quadrilateral and triangular element sides, the numbers are the local order of the element side nodes, as defined in Section *Element Corners, Sides*.

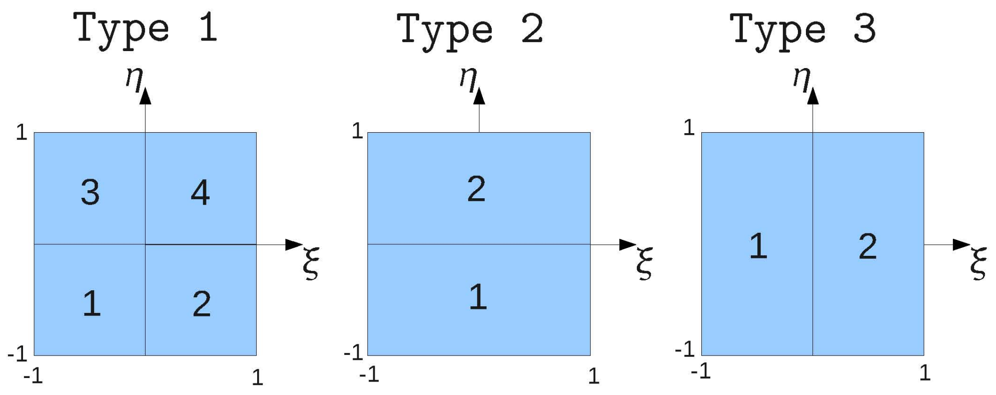


Fig. 1.16: Possible types of mortar interfaces, with ξ, η denoting the sides local parameter space.

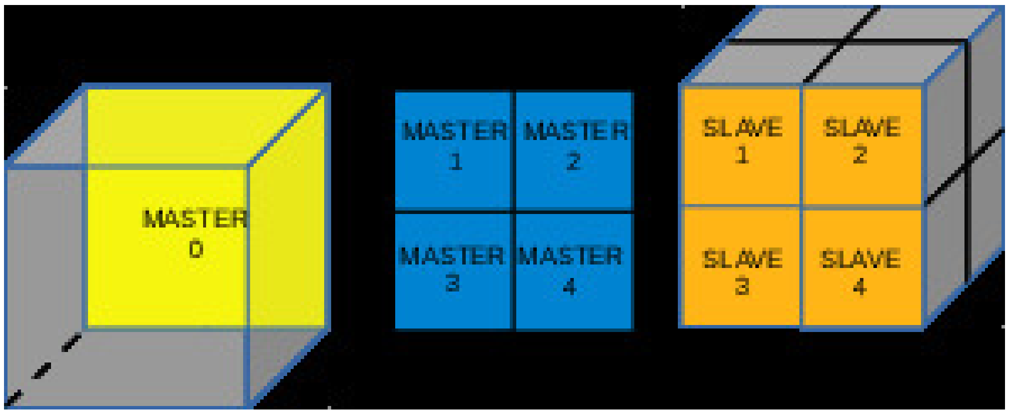


Fig. 1.17: Structure of the mortar interface, with the local mortar ID defined from $0 - 4$

Table 1.35: Sorting example for sides in SideInfo, for an element containing two mortar interfaces of type 1 and type 2/3. Note that local SideID and MortarID are not stored in SideInfo.

Global SideID	local SideID	local MortarID
42	1	0
43	1	1
44	1	2
45	1	3
46	1	4
47	2	0
48	3	0
49	3	1
50	3	2
51	4	0
52	5	0
53	6	0

1.2.6.2 Additional Information for Octrees

In addition to the existing format defined above, the mortar format contains non-necessary additional information concerning the octrees. It contains the octree node coordinates and a mapping of the element to the octrees. Note that the polynomial degree of the element mapping is defined as N_{geo}, while the octrees may have an independent polynomial degree $N_{g,tree}$. Elements and trees can be identical in case the element is on the lowest octree level and the polynomial degrees are identical.

Octree Global Attributes

In addition to the global attributes defined in Section *Global Attributes*, the non-conforming mesh format includes the following attributes.

Table 1.36: Additional mesh file attributes for octrees.

Attribute	Data type	Description
IsMortarMesh	INTEGER	Identify mesh as a mortar mesh, if present
NgeoTree $\geq$ 1	INTEGER	Polynomial degree $N_{g,tree}$ of tree mapping, used to determine the number of nodes per element
nTrees	INTEGER	Total number of octrees
nNodesTree	INTEGER	Total number of tree nodes: $(N_{g,tree}+1)^3 \cdot nTrees$

Mapping of the Global Element Index (ElemID) to the Octree Index (TreeID)

Name in file:	**ElemToTree**
Type:	INTEGER Size: Array(1:**nElems***)
Description:	The mapping from the global element index (ElemID) to its corresponding octree index (TreeID) if applicable.

Element Bounds in Tree Reference Space

Name in file:	**xiMinMax**
Type:	REAL Size: Array(1:3,1:2,1:**nElems***)
Description:	The array contains the element bounds in the tree reference space $[-1,1]^3$ given by the minimum (1:3,1,ElemID) and maximum (1:3,2,ElemID) corner nodes.

Node Coordinates of the Octrees

Name in file:	**TreeCoords**
Type:	REAL Size: Array(1:3,**nNodesTree***)
Description:	The coordinates of the nodes of the tree, as a set for each tree.

1.3 Appendix

1.3.1 Tested compiler combinations

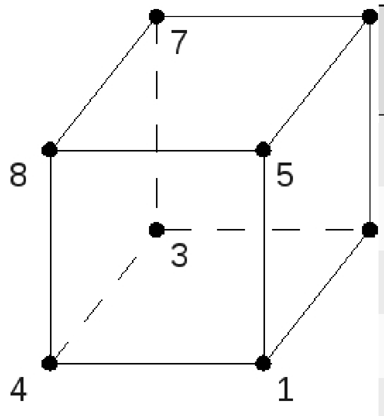
Dev	Version (Date)	System	Compiler	HDF5 (pre-compiled)	MPI	CMa	CGNS (pre-compiled)	Notes
XX	XX (Nov X)	Laptop Ubuntu 22.04	gnuX.)	X.X.X	openmpi-X.X.X	X.X.)	X.X.X	Does not work with more than 3 processors
SC	1.0.0 (Dec 14)	Laptop Ubuntu 22.04.1 LTS	gcc12.:	1.12.0 (no)	OFF	3.24.:	v3.4.1 (no)	
SC	1.0.0 (Dec 14)	Laptop Ubuntu 22.04.1 LTS	gcc12.:	12.2.0 (yes, configure)	OFF	3.24.:	v3.4.1 (no)	

1.4 List of Parameters

Parameters	Example
`BCIndex`	`BCIndex=(/1,2,3,4,5,6/)`
`BoundaryName`	`BoundaryName=BC_zminus`
`BoundaryOrder`	`BoundaryOrder=5`
`BoundaryType`	`BoundaryType=(/4,0,0,-1/)`
`checkElemJacobians`	`checkElemJacobians=T`
`conformConnect`	`conformConnect=T`
`Corner`	`Corner=(/0.,0.,0. ,,1.,0.,0. ,,1.,1.,0. ,,0.,1.,0. ,,0.,0.,1. ,,1.,0.,1. ,,1.,1`
`curvingMethod`	`curvingMethod=1`
`Debugvisu`	`Debugvisu=T`
`DebugvisuLevel`	`DebugvisuLevel=1`
`doExactSurfProjection`	`doExactSurfProjection=F`
`dozcorrection`	`dozcorrection=F`
`DXmaxToDXmin`	`DXmaxToDXmin=(/6.,100.,1./)`
`DZ`	`DZ=2`
`elemtype`	`elemtype=108`
`ExactNormals`	`ExactNormals=(/1,1/)`
`ExactSurfFunc`	`ExactSurfFunc=(/1,1/)`
`fac`	`fac=(/1.5,2.2,10/)`
`factor`	`factor=(/-1.75,1,-1.5/)`
`filename`	`filename=spheremesh.cgns`
`jacobianTolerance`	`jacobianTolerance=1.E-16`
`l0`	`l0=(/0,1,5,0/)`
`lowerZ_BC,upperZ_BC`	`lowerZ_BC=(/2,1,0,0/)`
`MeshIsAlreadyCurved`	`MeshIsAlreadyCurved=T`
`meshscale`	`meshscale=0.001`
`Meshtype`	`Meshtype=3`

Parameters	Example
Mode	Mode=1
nAnalyze	nAnalyze=5
nCurvedBoundaryLayers	nAnalyze=3
nElems	nElems=(/2,3,4/)
nElemsZ	nElemsZ=1
nExactNormals	nExactNormals=1
nFineHexa	nFineHexa=2
NormalsType	NormalsType=2
NormalVectFile	NormalVectFile=filename
nSkip	nSkip=2
nSkipZ	nSkipZ=2
NVisu	NVisu=5
nZones	nZones=1
outputFormat	outputFormat=1
ProjectName	ProjectName=cartbox
R_0	R_0=0.5
R_INF	R_INF=20
SpaceQuandt	SpaceQuandt=1.0
SplitElemFile	SplitElemFile=filename
SplitToHex	SplitToHex=T
stretchType	stretchType=(/3,1,0/)
useCurveds	useCurveds=T
vv	vv=(/0,0,1./)
WhichMapping	WhichMapping=4
zLength	zLength=1.0
zperiodic	zperiodic=T
zstart	zstart=0.

Table 1.38: Surface corner nodes mapping.



Surface	Corner Nodes	Position in the Cart. coordinate system
1	1-2-3-4	z < 0 plane (z-)
2	1-2-5-6	y < 0 plane (y-)
3	2-3-6-7	x > 0 plane (x+)
4	3-4-7-8	y > 0 plane (y+)
5	1-4-5-8	x < 0 plane (x-)
6	5-6-7-8	z > 0 (z+)

Fig. 1.18: The cartesian box.

This user guide is organized to both guide the first steps as well as provide a complete overview of the simulation code's features from a user and a developer point of view.

- Chapter *Installation* contains step by step instructions from obtaining the source code up to running hopr and visualizing the generated mesh.
- Chapter *HOPR HDF5 Curved Mesh Format* describes in detail how the hopr mesh format is constructed and how mesh information is processed.
- Chapter *Appendix* contains additional information, e.g., tested compiler and library versions.
- Chapter *List of Parameters* gives an overview of all read-in parameters that are available in hopr.

CHAPTER

TWO

DEVELOPER GUIDE

The developer guide is intended to be for people who come in more close contact with HOPR, i.e., code developers and performance analysts as well as people who are tasked with working or extending the documentation of HOPR.

2.1 Github Workflow

Code development is performed on the Github platform, with the protected `master` and `master.dev` branches. The actual development is performed on feature branches, which can be merged to `master.dev` following a merge request and the completion of a merge request checklist. After a successful pass of the nightly and weekly regression test, the `master.dev` can be merged into the `master`. A merge of the `master.dev` to the `master` should be associated with a release tag, where the changes to previous version are summarized.

In the following the envisioned development process using issues and milestones, the release & deploy procedure as well as other developer relevant issues are discussed.

2.1.1 Issues & Milestones

Issues are created for bugs, improvements, features, regression testing and documentation. The issues should be named with a few keywords. Try to avoid describing the complete issue already in the title. The issue can be assigned to a certain milestone (if appropriate).

Milestones are created based on planned releases (e.g. Release 1.2.1) or as a grouping of multiple related issues (e.g. Documentation Version 1, Clean-up Emission Routines). A deadline can be given if applicable. The milestone should contain a short summary of the work performed (bullet-points) as its contents will be added to the description of the releases. Generally, merge requests should be associated with a milestone containing a release tag, while issues should be associated with the grouping milestones.

As soon as a developer wants to start working on an issue, she/he shall assign himself to the issue and a branch and merge request denoted as work in progress (`WIP: ...`) should be created to allow others to contribute and track the progress. For this purpose, it should be created directly from the web interface within the issue (`Create merge request`). This creates a branch, naming it automatically, leading with the issue number (e.g. `60-fix-boundary-condition`) and associates the branch and merge request to the issue (visible in the web interface below the description). To start working on the issue, the branch can be checked out as usually.

Ideally, issues should be created for every code development for documentation purposes. Branches without an issue should be avoided to reduce the number of orphaned/stale branches. However, if branches are created outside of the issue context, they should be named with a prefix indicating the purpose of the branch, according to the existing labels in Github. Examples are given below:

```
feature.chemistry.polyatomic
improvement.tracking.curved
bug.compiler.warnings
reggie.chemistry.reservoir
documentation.pic.maxwell
```

Progress tracking, documentation and collaboration on the online platform can be enabled through creating a merge request with the WIP prefix for this branch instead of an issue. An issues created afterwards cannot be associated with an already created branch, without renaming the branch to include the issue number at the beginning. However, this should be avoided.

2.1.2 Merge Request

Merge requests that are not WIP are discussed every Monday by the developer group to be considered for a merge. The following checklist has to be completed before a merge request should be approved. For bugs only the first points have to be considered, while for features and improvements the complete list has to be completed. The **Feature** merge request template considers the following bullet points

- [ ] Style Guide
- [ ] Maximum of 10 compile warnings via *./tools/test_max_warnings.sh*. How many warning were found?
- [ ] Descriptions for new/changed routines
  - [ ] Short header description (do not just spell out the name of the subroutine, units for important variables if applicable)
  - [ ] Workflow
    - [ ] Short summary in the header
    - [ ] Inside the routine at the appropriate positions
- [ ] Reggie
  - [ ] Add small test setup
  - [ ] Add entry in REGGIE.md table
  - [ ] Check automatic restart functionality of reggie example via Load Balance (checks correct allocation and deallocation for the test case)
- [ ] New feature description in appropriate documentation (user/developer guide)
- [ ] Check that no large files were added to the repository

For this purpose, the developer can select the respective template for his merge request (**Bug**: only first two to-do's or **Feature**: all to-do's, Improvements can utilize either depending on the nature of the improvement). The appropriate checklist will then be displayed as the merge request description. Merge requests generated automatically through the Issues interface have already `Closes #55` as a description. When editing the merge request, the description gets overwritten by the template. Thus, the issue number has to be added manually after the template is chosen. The templates for merge requests are stored in *./.Github/merge_request_templates/*.

2.1.3 Release and deploy

A new release version of HOPR is created from the **master** repository, which requires a merge of the current **master.dev** branch into the **master** branch. The corresponding merge request should be associated with a release milestone (e.g. *Release 1.X.X* within which the merge request is referenced, e.g., "See merge request !283"). Within this specific merge request, the template `Release` is chosen, which contains the to-do list as well as the template for the release notes as given below. After the successful completion of all to-do's and regression checks (check-in, nightly, weekly), the **master.dev** branch can be merged into the **master** branch.

2.1.3.1 Release Tag

A new release tag can be created through the web interface (Repository -> Tags -> New tag) and as the `Tag name`, the new version number is used, e.g.,

```
v1.X.X
```

The tag is then created from the **master** branch repository and the `Message` is left empty. The release notes, which were used within the corresponding milestone, shall be given in the following format

```
## Release 1.X.X

### Documentation

* Added section about particle emission

### Reggie

* Added a regression test of the chemistry routine

### Features

* Skipping field update for the HDG solver for user-defined number of iterations

### Improvements

* Speed-up by skipping/cycle over neutral particles in deposition

### Fixes

* Treatment of non-linear polyatomic molecules during analyze and wall interaction
```

Headlines without changes/additions within a release can be omitted.

2.1.3.2 Collaborative Numerics Group

The collaborative numerics group for HOPR is located at *https://Github.com/collaborative-numerics-group/HOPR*. There are two possible ways of sharing code with other people and are explained in the following.

Single master branch repository

Method 1 involves a single master branch that is updated from the internal HOPR repository, similar to the repository of HOPR at *github.com*.

The master branch of development group can be merged after the successful regression check with the master of the collaborative group. For this purpose, the collaborative repository can be added as a remote (this step has only to be performed once)

```
git remote add remote_name git@Github.com:collaborative-numerics-group/HOPR/HOPR.git
```

First, make sure to have the most recent version of the master branch (of the development repository)

```
git checkout master && git pull
```

Now you can checkout the most recent version of the master branch of the collaborative-numerics-group and create a local branch with that version (performing only a simple checkout will create a detached HEAD state)

```
git fetch
git checkout -b branch_name remote_name/master
```

The master branch of the development repository can now be merged into the newly created branch

```
git merge origin/master
```

Finally, the changes can be pushed from the local branch *branch_name* to the master of collaborative-numerics-group

```
git push remote_name master
```

If a tag has also been created, it should be pushed separately.

```
git push remote_name tag_name
```

Afterwards, the local branch *branch_name* can either be deleted or utilized for future merges

```
git branch -d branch_name
```

Pushing each branch to a separate repository

Method 2 involves pushing each development branch from the internal HOPR repository to a separate repository in the HOPR CRG separately.

Extract solely a single branch by cloning only *myfeaturebranch* via

```
git clone -b myfeaturebranch --single-branch git@Github.com:HOPR/HOPR.git HOPR ;
```

Navigate to the newly created directory

```
cd HOPRs_new_CRG
```

Push this repository that only contains one branch to the CRG via

```
git push --mirror git@Github.com:collaborative-numerics-group/HOPRs/HOPRs.
↪myfeaturebranch.git
```

which created a new repository that only contains the desired feature branch. When sharing this new repository with new people, simply add them as members of this new repository (not the complete CRG!).

2.1.3.3 GitHub

Finally, the release tag can be deployed to GitHub. This can be achieved by running the `Deploy` script in the CI/CD -> Schedules web interface. At the moment, the respective tag and the release have to be created manually on GitHub through the web interface with **HOPRs-framework** account. The releases are accessed through Releases and a new release (including the tag) can be created with `Draft a new release`. The tag version should be set as before (`v1.X.X`) and the release title accordingly (`Release 1.X.X`). The release notes can be copied from the Github release while omitting the `## Release 1.X.X` headline as it was given with the release title before.

2.2 Style Guide

- Why do we need a style guide?
  - It creates a unified appearance and coding structure
  - It makes the code more understandable and therefore important information is understood more easily
  - It forces the developers to think more actively about their work
- General rules
  - Coding language: English
  - A maximum of 132 characters are allowed per line (incl. Comments)
  - Indentation: 2 spaces (no tabs!)
  - Line breaks in comments -> the following line must be indented appropriately
  - Comments of modules and input-/output variables: Doxygen style
  - Comments of preprocessor directives in C-Style

2.2.1 Header of Functions and Subroutines

Function calls must always supply the variable name of optional arguments. Always use `USE` statements with `ONLY`

```
USE MODULE, ONLY: ...
```

this accounts for variables and function/subroutines. An exception are the initialization and finalization routines.

```
!==================================================================
!> \brief Fills the solution array U with a initial solution.
!>
!> Fills the solution array U with a initial solution provided by the ExactFunc␣
↪subroutine through interpolation. Function is
!> specified with the IniExactFunc paramter.
!==================================================================
SUBROUTINE FillIni(NLoc,xGP,U)
!------------------------------------------------------------------
! MODULES
USE MOD_PreProc
```

(continues on next page)

(continued from previous page)

```
USE MOD_Equation_Vars ,ONLY: IniExactFunc
USE MOD_Exactfunc     ,ONLY: ExactFunc
USE MOD_Mesh_Vars     ,ONLY: nElems
IMPLICIT NONE
!----------------------------------------------------------------
! INPUT/OUTPUT VARIABLES
INTEGER,INTENT(IN)             :: NLoc                              !<␣
↪Polynomial degree of solution
REAL,INTENT(IN)                :: xGP(3,    0:NLoc,0:NLoc,0:NLoc,nElems)    !<␣
↪Coordinates of Gauss-points
REAL,INTENT(OUT)               :: U(1:PP_nVar,0:NLoc,0:NLoc,0:NLoc,nElems)  !< Solution␣
↪array
!----------------------------------------------------------------
! LOCAL VARIABLES
INTEGER                        :: i,j,k,iElem
!================================================================

! Evaluate the initial solution at the nodes and fill the solution vector U.
DO iElem=1,nElems
  DO k=0,NLoc; DO j=0,NLoc; DO i=0,NLoc
    CALL ExactFunc(IniExactFunc,0.,xGP(1:3,i,j,k,iElem),U(:,i,j,k,iElem))
  END DO; END DO; END DO
END DO
END SUBROUTINE FillIni
```

The separators `!====` and `!----` are exactly 132 characters long (here they have been shortened for visualization purposes).

2.2.2 Variables

- Preprocessor variables: `PP_$var`

  ```
  PP_nVar
  ```

  Note that

  – `USE MOD_Preproc` cannot be used with `ONLY` because the pro-processor flags sometimes result in constants and not variables

  – `PP_N` and other pre-processor variables that may be constants cannot be assigned in `ASSOCIATE` constructs

- Counters: the counting variable (lower case) + description (the first character is capital case)

  ```
  DO iVar=1,PP_nVar
  ```

- Variables generally begin with a capital letter (composite words also)

  ```
  ActualElem
  ```

- Dimension allocations must be specified by giving both a lower and an upper boundary

  ```
  ALLOCATE(U(1:PP_nVar,0:NLoc,0:NLoc,0:NLoc,nElems))
  ```

- When using single characters: small at the beginning when using composite words otherwise in capital letters. Both is possible when purely single characters are used. Exceptions are allowed in special cases, but they are not recommended.

```
hTilde, TildeH, (Elem%U)
```

2.2.3 Functions and Control Structures

- User-defined functions and subroutines should carry meaning in their name. If their name is composed of multiple words, they are to be fused together without underscores (_) and the first letter of each words should be capital.

```
GetParticleWeight(), isChargedParticle()
```

  An exception to this rule is the usage of underscores (_) for shared memory arrays, where `_Shared` indicates that the property is available to all processors on the same shared-memory domain (generally a node). Furthermore, the words `Get`, `get`, `Is`, `is`, `Do`, `do` indicate the intention of the function/subroutine in supplying or acquiring specific properties or values. User-defined functions and subroutines should not, in general, be named in all-capital letters.

- FORTRAN intrinsics generally in capital letters

```
ALLOCATE(), DO, MAX(), SQRT(), INT(), etc.
```

- END-X is to be separated by a space

```
END DO, END IF, END SUBROUTINE
```

- For loops and `IF` statements etc. comments are to be inserted at the end (and in-between, e.g. when `ELSE IF` is used)

```
DO iVar=1,PP_nVar
  IF (a.EQ.b) THEN
...
  ELSE ! a.NE.b
...
  END IF ! a.EQ.b
...
END DO ! PP_nVar
```

2.2.4 Workflow Description

Additionally to the header description, a short workflow table of contents at the beginning of the subroutine or function is required for longer subroutines in which multiple tasks are completed. Example:

```
! -----------------------------------------------------------------------------
! MAIN STEPS        []=FV only
! -----------------------------------------------------------------------------
! 1.  Filter solution vector
! 2.  Convert volume solution to primitive
! 3.  Prolong to face (fill U_master/slave)
! 4.  ConsToPrim of face data (U_master/slave)
```

(continues on next page)

(continued from previous page)

```
![5.] Second order reconstruction for FV
! 6.  Lifting
! 7.  Volume integral (DG only)
![8.] FV volume integral
! 9.  IF EDDYVISCOSITY: Prolong muSGS to face and send from slave to master
! 10. Fill flux (Riemann solver) + surface integral
! 11. Ut = -Ut
! 12. Sponge and source terms
! 13. Perform overintegration and apply Jacobian
! -----------------------------------------------------------------------------
```

Furthermore, the steps are required to be found at the appropriate position within the code. It is not allowed to just incorporate the corresponding number of the step within the code.

```
! (0. Nullify arrays)
! NOTE: UT and U are nullified in DGInit, and Ut is set directly

! 1. Filter the solution vector if applicable, filter_pointer points to cut-off
IF(FilterType.GT.0) CALL Filter_Pointer(U,FilterMat)

! 2. Convert Volume solution to primitive
CALL ConsToPrim(PP_N,UPrim,U)

! X. Update mortar operators and neighbour connectivity for the sliding mesh
CALL PrepareSM()

! 3. Prolong the solution to the face integration points for flux computation
! --------------------------------------------------------------
! General idea: The slave sends its surface data to the master
! where the flux is computed and sent back to the slaves.
! Steps:
! (these steps are done for all slave MPI sides and then for all remaining sides):
! 3.1)  Prolong solution to faces and store in U_master/slave.
!       Use them to build mortar data (split into 2/4 smaller sides).
![3.2)] The information which element is a DG or FV subcells element is stored
!       in FV_Elems per element.
![3.3)] The reconstruction of slopes over element interfaces requires,
!       besides U_slave and FV_Elems_slave, some more information that
!       has to be transmitted from the slave to the master MPI side.
! 3.4)  Finish all started MPI communications (after step 2. due to latency hiding)

#if USE_MPI
! Step 3 for all slave MPI sides
! 3.1) Prolong solution to faces and store in U_master/slave.
!      Use them to build mortar data (split into 2/4 smaller sides).
CALL StartReceiveMPIData(U_slave,DataSizeSide,1,nSides,MPIRequest_U(:,SEND),SendID=2) !␣
↪Receive MINE / U_slave: slave -> master
CALL StartReceiveSM_MPIData(PP_nVar,U_MorRot,MPIRequestSM_U,SendID=2) ! Receive MINE / U_
↪slave: slave -> master
CALL ProlongToFaceCons(PP_N,U,U_master,U_slave,L_Minus,L_Plus,doMPISides=.TRUE.)
CALL U_MortarCons(U_master,U_slave,doMPISides=.TRUE.)
CALL U_MortarConsSM(U_master,U_slave,U_MorStat,U_MorRot,doMPISides=.TRUE.)
```

(continues on next page)

(continued from previous page)

```
CALL StartSendMPIData(   U_slave,DataSizeSide,1,nSides,MPIRequest_U(:,RECV),SendID=2) !␣
↪SEND YOUR / U_slave: slave -> master
CALL StartSendSM_MPIData(   PP_nVar,U_MorRot,MPIRequestSM_U,SendID=2) ! SEND YOUR / U_
↪slave: slave -> master
#if FV_ENABLED
! 3.2) The information which element is a DG or FV subcells element is stored
!      in FV_Elems per element.
CALL FV_Elems_Mortar(FV_Elems_master,FV_Elems_slave,doMPISides=.TRUE.)
CALL StartExchange_FV_Elems(FV_Elems_slave,1,nSides,MPIRequest_FV_Elems(:,SEND),
↪MPIRequest_FV_Elems(:,RECV),SendID=2)
#endif /* FV_ENABLED */
```

2.3 Building the Documentation

The user and developer guides are built automatically via Read The Docs. When changing the guides, build the html and pdf files locally before committing changes to the repository.

Prerequisites

The following shows how to create the html and pdf files for the user/developer guide. First, install the required prerequisites. Install *python3* and make sure that pip (third-party Python packages) is installed. If you have an old version of, e.g., Ubuntu visit this website.

```
sudo apt install python3-pip
```

Navigate to the documentation folder from the HOPR top level directory

```
cd docs/documentation
```

Run pip to install the required extensions and packages for compiling the user guide (only once)

```
python3 -m pip install --exists-action=w --no-cache-dir -r requirements.txt
```

Make sure that *latexmk* is installed on the system for compiling the PDF version of the user guide. For Ubuntu, follow this link for installation.

```
sudo apt-get install latexmk
```

HTML Version

Compile the html version of the user guide via

```
python3 -m sphinx -T -E -b html -d _build/doctrees -D language=en . _build/html
```

Check that no errors occur during compilation and then navigate to the created html files

```
cd _build/html
```

Open index.html to see if everything has worked out correctly (e.g. with your favourite browser). Note that you can simply run the script *buildHTML.sh* in the *documentation* directory for this task.

PDF Version

Next, create the pdf output.

```
python3 -m sphinx -b latex -D language=en -d _build/doctrees . _build/latex
```

and switch into the output directory

```
cd _build/latex
```

Finally, compile the pdf file

```
latexmk -r latexmkrc -pdf -f -dvi- -ps- -jobname=HOPR -interaction=nonstopmode
```

and check if the pdf exists

```
ls _build/latex/HOPR.pdf
```

Note that you can simply run the script *buildPDF.sh* in the *documentation* directory for this task.

2.4 Compiler Options

Todo: scribed current compiler options and what their purpose is (what was encountered in the past in order for the options to be as they are now?)

- Release: optimized with -O3 for execution runs
- Debug: debugger options
- Sanitize: GNU sanitizer for further debugging

Compi Flag	Op- tions	What does it do?
–ffpe- trap=lis	*in- valid*	invalid floating point operation, such as SQRT(-1.0)
	zero	division by zero
	over- flow	overflow in a floating point operation
	un- der- flow	underflow in a floating point. **DO NOT USE**. Because a small value can occure, such as exp(-766.2). operation
	pre- ci- sion	loss of precision during operation
		Some of the routines in the Fortran runtime library, like **CPU_TIME**, are likely to trigger floating point exceptions when ffpe-trap=precision is used. For this reason, the use of ffpe-trap=precision is not recommended.
	de- nor- mal	operation produced a denormal value
- fbacktra		runtime error should lead to a backtrace of the error
- fcheck=	*all*	enable all run-time check
	array temps	Warns at run time when for passing an actual argument a temporary array had to be generated. The information generated by this warning is sometimes useful in optimization, in order to avoid such temporaries.
	boun	Enable generation of run-time checks for array subscripts and against the declared minimum and maximum values. It also checks array indices for assumed and deferred shape arrays against the actual allocated bounds and ensures that all string lengths are equal for character array constructors without an explicit typespec.
	do	Enable generation of run-time checks for invalid modification of loop iteration variables
	mem	Enable generation of run-time checks for memory allocation. Note: This option does not affect explicit allocations using theALLOCATE statement, which will be always checked.
	point	Enable generation of run-time checks for pointers and allocatables.
	re- cur- sion	Enable generation of run-time checks for recursively called subroutines and functions which are not marked as recursive. See also -frecursive. Note: This check does not work for OpenMP programs and is disabled if used together with -frecursiveand -fopenmp.
- fdump- core		Request that a core-dump file is written to disk when a runtime error is encountered on systems that support core dumps. This option is only effective for the compilation of the Fortran main program
- fstack- arrays		Adding this option will make the Fortran compiler put all local arrays, even those of unknown size onto stack memory. If your program uses very large local arrays it is possible that you will have to extend your runtime limits for stack memory on some operating systems. This flag is enabled by default at optimization level -Ofast.
- frepack arrays		In some circumstances GNU Fortran may pass assumed shape array sections via a descriptor describing a noncontiguous area of memory. This option adds code to the function prologue to repack the data into a contiguous block at runtime.This should result in faster accesses to the array. However it can introduce significant overhead to the function call, especially when the passed data is noncontiguous.
- finline- matmul limit=n		
-finit- local- zero		The -finit-local-zero option instructs the compiler to initialize local INTEGER, REAL, and COMPLEX variables to zero, LOGICALvariables to false, and CHARACTER variables to a string of null bytes

2.5 Building the AppImage Executable

Navigate to the HOPR repository and create a build directory

```
mkdir build && cd build
```

and compile HOPR using the following cmake flags

```
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
```

and then

```
make install DESTDIR=AppDir
```

Then create an AppImage (and subsequent paths) directory in the build folder

```
mkdir -p AppDir/usr/share/icons/
```

and copy the HOPR logo into the icons directory

```
cp ../docs/Meshformat/pics/HOPR_logo.png AppDir/usr/share/icons/hopr.png
```

A desktop file should already exist in the top-level directory containing

```
[Desktop Entry]
Type=Application
Name=hopr
Exec=hopr
Comment=Tool create .h5 hopr meshes to be read by piclas, flexi, fluxo, etc.
Icon=hopr
Categories=Development;
Terminal=true
```

Next, download the AppImage executable and run

```
./linuxdeploy-x86_64.AppImage --appdir AppDir --output appimage --desktop-file=../hopr.
↪desktop
```

If an error is encountered, see the Section *Troubleshooting* section for a possible solution. The executable should be created in the top-level directory, e.g.,

```
hopr-2de94ad-x86_64.AppImage
```

2.6 Troubleshooting

This section collects typical errors that are encountered when trying to build the AppImage.

2.6.1 dlopen(): error loading libfuse.so.2

If the error

```
dlopen(): error loading libfuse.so.2

AppImages require FUSE to run.
You might still be able to extract the contents of this AppImage
if you run it with the --appimage-extract option.
See https://github.com/AppImage/AppImageKit/wiki/FUSE
for more information
```

is encountered, this might be due to a missing fuse installation. On Debian/Ubuntu systems, simply run

```
sudo apt install libfuse2
```

2.7 Markdown Examples

2.7.1 hyperlinks

one with a title. Unclear how to enforce new window.

2.7.2 Code environment

Either use fenced style (tildes)

```
if (a > 3) {
  moveShip(5 * gravity, DOWN);
}
```

or indented style (4 whitespaces)

```
if (a > 3) {
  moveShip(5 * gravity, DOWN);
}
```

Both works with pandoc and wordpress. Also see pandoc verbatim code.

2.7.3 Equations

(@gleichung1) $$a = b * c$$ As (@gleichung1) shows, blabla.

2.7.4 Bibtex, cite

Hindenlang [@Hindenlang2015]. Only works with pandoc!

[bibshow file=references.bib]

Hindenlang [bibcite key=Hindenlang2015], Gassner [bibcite key=gassner2011disp]

2.7.5 section references

2.7.6 Figures, caption

Fig. 2.1: This is an example caption.

See Fig. 2.1 for an image from the web embedded in this documentation.

See Fig. 2.2 for embedding a local file.

Fig. 2.2: This is an example caption.

2.7.7 tables

2.7.8 unnumbered section headings

just add

```
{-}
```

after the heading

2.7.9 Code blocks for various languages

```
int a = 32;
int a = 32;
```

This guide is organized to guide the first implementation steps as well as provide a complete overview of the simulation code's features from a developer's point of view.

- The first Chapter *Github Workflow* shall give an overview over the development workflow within the GitHub environment, and the necessary steps to create a release and deploy updates to GitHub.
- The second Chapter *Style Guide* describes the rules and guidelines regarding code development such as how the header of functions and subroutines look like.
- Chapter *Building the Documentation* describes how to build the html and pdf files locally before committing changes to the repository.
- Chapter *Compiler Options* gives an overview of compiler options that are used in HOPR and their purpose.
- Chapter *Building the AppImage Executable* described how an AppImage executable of HOPR is created.
- Chapter *Markdown Examples* gives a short overview of how to include code, equations, figures, tables etc. in the user and developer guides in Markdown.

CHAPTER

THREE

BUILT-IN MESH GENERATORS

3.1 Straight-Edged Boxes

HOPR has several simple built-in mesh generators.

3.1.1 Cartesian Box

This tutorial shows how to generate a simple mesh of a cubical box and the definition of boundary conditions. The parameter file can be found in

```
tutorials/1-01-cartbox/parameter.ini
```

See *Cartesian Box: Exemplary Variations of Boundary Conditions* for cases with different boundary conditions.

3.1.1.1 Cartesian Box: Description of Parameters

The following table describes all parameters present in the parameter file. Therefore, it is limited to the parameters needed to generate a Cartesian box mesh. A description of all parameters can be found in *List of Parameters*.

Table 3.1: Parameters Cartesian Box.

Parameters	Setting	Description
`Proj`	`cartbox`	Defines the name embedded in the generated mesh file and is used as prefix for the output files (`cartbox_mesh.h5`, `cartbox_Debugmesh.dat`, `cartbox_Debugmesh_BC.dat`, . . .).
`Debu`	`T`	Generation of visualization files for the volume and boundary mesh (`*_Debugmesh.dat`, `*_Debugmesh_BC.dat`) to aid debugging.
`Mode`	`1`	Mode of mesh generation; 1: Cartesian mesh (build-in), 3: CFD General Notation System (CGNS, extern)
`nZon`	`1`	Number of Cartesian boxes
`Corn`	`(/0.,0.,0. ,,1.,0.,0. ,,1.,1.,0. ,,0.,1.,0. ,,0.,0.,1. ,,1.,0.,1. ,,1.,1.,1. ,,0.,1.,1. /)`	Coordinates of the box's corner nodes in the three-dimensional Cartesian coordinate system. Refer to Fig. 1.18 for the node ordering.Furthermore, the corner nodes define the six surfaces of the cartesian box, see Table 1.38.
`nEle`	`(/2,3,4/)`	Number of elements per box in the direction of the Cartesian coordinate axes; `(/nElemX,nElemY,nElemZ/)`
`BCIn`	`(/1,2,3,4,5,6/)`	The `BCIndex` parameter assigns a bondary condition to each surface of the cartesian box in order of the surfaces. The number of a vector's component represents the nth boundary condition in order of its position in the file. Hence, each position refers to the six box sides `(/z-,y-,x+,y+,x-,z+/)`. Indices can be applied repeatedly, see *Cartesian Box: Exemplary Variations of Boundary Conditions* for details.
`Elem`	`108`	Type of cells/elements used for discretization; `104`: Tetrahedron, `105`: Pyramid, `106`: Prism with triangular base, `108`: Hexahedron
`Boun`	`BC_zminus`	Name of each boundary condition
`Boun`	`(/4,0,0,0/)`	Type of each boundary condition, provided with the components `(/ Type, curveIndex, State, alpha /)`.For a single Cartesian box, only the component `Type` has to be set.The other components are set to the default value `0`. Further description of the components can be found in the next tutorials or in the *List of Parameters*.

3.1.1.2 Cartesian Box: Boundary Conditions and Sketch

Fig. 1.2 shows the sketch of the current flow setup. Fig. 1.3 gives the corresponding excerpt of the parameter file containing the boundary conditions. Entries are colored to highlight the connection between the `BCIndex` and the boundary conditions. The same colors are used for the visualization below.

The index of the first component of the `BCIndex` vector `1` assigns the boundary condition on position one, `BC_zminus`, to the first surface. Consequently, the index of the second component of the `BCIndex` vector `2` assigns the second boundary condition `BC_yminus` to the second surface of the Cartesian box and so on. More examples are given in *Cartesian Box: Exemplary Variations of Boundary Conditions*.

3.1.1.3 Cartesian Box: Output Visualization

The following section highlights the visualization of the above setup. For more details on the HOPR visualization output, see *Visualization*.

This is a visualization of the cartbox_Debugmesh.dat file.

This is a visualization of the `cartbox_Debugmesh_BC.dat` file. The colors of the surfaces represent the boundary conditions and are identical to the ones in the excerpt of the parameter file.

3.1.1.4 Cartesian Box: Exemplary Variations of Boundary Conditions

For a better understanding of the interaction between the parameter `BCIndex` and the declaration of the boundary conditions, two different examples are presented below. Both examples are slight variations of the tutorial Cartesian Box in that way, that there are only three different boundary conditions: `WALL`, `INFLOW` and `OUTFLOW`. This means the components belonging to the surfaces will be equal in the `BCIndex` vector (see *List of Parameters*).

Example 1

The parameter file of this example can be found in

```
/tutorials/1-01-cartbox/parameter_ex1.ini
```

Fig. 1.11 shows a sketch of the current problem. Figure 1.12 shows a snippet of the parameter file which deals with the boundary conditions. In this code snippet, some text elements are colored to show the connection between the surfaces and their associated boundary conditions. The same colors are used for the visualization below.

The first four components of the `BCIndex` vector are equal. The index of these components `1` means that the first boundary condition, `BC_wall`, is assigned to the surfaces one to four. Furthermore, the fifth component of the `BCIndex` vector with the index `2` means that the second boundary condition `BC_inflow` is assigned to the fifth surface of the Cartesian box. The third boundary condition `BC_outflow` is assigned to the the sixth surface. Therefore, the last or the sixth component of the `BCIndex` vector is set to `3`.

If you need help visualizing the HOPR output, visit the *Visualization* page.

A visualization of the mesh of the cartbox_ex1_Debugmesh.dat file is given below.

This is a visualization of the cartbox_ex1_Debugmesh_BC.dat file. The colors of the surfaces represent the boundary conditions and are the same as in the snippet of the parameter file.

Example 2

The parameter file of this example can be found in

```
/tutorials/1-01-cartbox/parameter_ex2.ini
```

Figure 1.17 shows the sketch of the current problem. Figure 1.18 shows a snippet of the parameter file which deals with the boundary conditions. In this code snippet, some text elements are colored to show the connection between the surfaces and their asociated boundary conditions. The same colors are used for the visualization below.

In this example the first, third, and sixth component of the `BCIndex` vector are equal. The index of these components `1` means that the boundary condition at position one, `BC_wall`, is assigned to the surfaces one, three and six. Furthermore, the fifth component of the `BCIndex` vector with the index `2` means that the second boundary condition `BC_inflow` is assigned to the fifth surface of the cartesian box. The third boundary condition `BC_outflow` is assigned to the second and the forth surface. Therefore, the second and fourth component of the `BCIndex` vector are set to `3`.

If you need help visualizing the HOPR output, visit the *Visualization* page.

A visualization of the mesh of the cartbox_ex2_Debugmesh.dat file is given below.

This is a visualization of the cartbox_ex2_Debugmesh_BC.dat file. The colors of the surfaces represent the boundary conditions and are the same as in the snippet of the parameter file.

3.1.2 Periodic Boundary Conditions

This tutorial shows how to define periodic boundary conditions, using only slightly modified definitions from the Cartesian Box tutorial. The parameter file can be found in

```
tutorials/1-02-cartbox_periodic/parameter.ini
```

3.1.2.1 Periodic Boundary Conditions: Description of Parameters

In the following parameters of the parameter file are explained. This description consists only of parameters which are necessary to generate a periodic boundary condition. A description of all parameters of the parameterfile can be found in the previous tutorial Cartesian Box and *List of Parameters*.

Table 3.2: Parameters Periodic Boundary Conditions.

Parameters	Setting	Description
`Boun`	`BC_z`	Name of the boundary condition
`Boun`	`(/ 1, 0, 0, -1/ )`	For each periodic boundary condition three parameters are mandatory, the `BoundaryName`, the `BoundaryType` and the displacement vector `vv`. The Type parameter consists of fourcomponents to set: `(/ Type, curveIndex, State, alpha /)`. For a periodic boundary condition the component Type has always to set to "1". The fourth component `alpha` assigns adisplacement vector `vv` and its direction (-/+) to the periodic boundary. An `alpha` of "-1" means that the first ("1") defined displacement vector is assigned to this surface in the oppositedirection ("-1") as he was defined. For a simple cartesian box the other components `curveIndex` and `State` have to be set 0.The further description of the components can be found in the next tutorials or in the *List of Parameters*.
`vv`	`(/ 0. , 0. , 1. /)`	The displacement vector must be specified in the three-dimensional Cartesian coordinate system and must be normal to a surface to which the vector was assigned. In addition, the displacement vector mustpoint to the inside of the Cartesian box. In case of two parallel surface planes, both with periodic boundary conditions, only one displacement vector needs to be defined. Therefore, thedifferent directions of the vectors can be compensated by changing the sign of `alpha`, the fourth component of the `BoundaryType` vector.It has to be taken into account that the displacement vector has to be as long as the distance between the surfaces to which it is assigned. The index of a displacement vector is alsodefined by the position of its definition, as the parameter `BCIndex`. Multiple boundary condition definitions between two definitions of displacement vectors do not affect the index ofthe displacement vectors.

3.1.2.2 Periodic Boundary Conditions: Boundary Conditions and Sketch

Figure 1.23 shows the sketch of the current problem. It is similar to the problem in the Cartesian Box tutorial, but instead of Dirichlet, periodic boundary conditions are assigned to the surfaces one, two, four, and six. Below is a code snippet of the parameter file that deals with the periodic boundary conditions. In this code snippet, some text elements are colored to show the connection between boundary conditions and their related displacement vectors. The same colors are used for the visualization in Fig. 1.17.

As one can see, the first four boundary conditions are periodic because the last `alpha` components of the `BoundaryType` parameters are not equal to zero. In the definitions of the first two boundary conditions which are assigned to the surfaces one and six (see `BCindex`) the `alpha` component is set to 1. This means that the assiciated displacement vector is the first defined displacement vector in the parameter file. The sign of `alpha` can be explained by the position of the associated surfaces. On the one hand, the vector must point to the inside of the Cartesian box. On the other hand, the vector must also point to the other surfaces that have been assigned with the periodic boundary condition. The vector itself must point in the direction of the z-axis because it must be normal to the surface. In addition, the side length of the Cartesian box is one, so all defined displacement vectors have a length of one. For the other two periodic boundary conditions of surfaces two and four, the second defined displacement vector is consulted (see `alpha` value of the `BoundaryType` parameters). The components of the displacement vector `(/0.,1.,0./)` result from the required perpendicularity to the surfaces and the side length of the Cartesian box.

3.1.3 Multiple Cartesian Boxes

This tutorial shows how to create a mesh consisting of multiple Cartesian boxes and how to link them via boundary conditions. The parameter file can be found in

```
tutorials/1-03-cartbox_multiple/parameter.ini
```

3.1.3.1 Multiple Cartesian Boxes: Definition of Multiple Cartesian Boxes

The following general snippet of the parameter file shows how to define multiple Cartesian boxes in the parameter file.

```
!=================================================================== !
! MESH
!=================================================================== !
  Mode         =1                       ! Mode for Cartesian boxes
  nZones       =n                       ! number of boxes

! ===  zone 1 ===
  ...

! ===  zone 2  ===
  ...

  .
  .
  .

! ===  zone n  ===
 ...
```

First, the parameter `nZones` must be adapted to the number of Cartesian boxes to be defined. The Cartesian boxes can be defined simply by writing them and their specifications `(Corner, nElems, BCIndex, elemtype, ...)` among themselves. Furthermore, it is important that the boxes are correctly defined to each other. If there is to be a contact between two boxes, it is mandatory that the corresponding surfaces will coincide. This means that the corner nodes of the surfaces must also coincide.

However, a correct definition of the corner nodes is not sufficient for the functionality. Therefore, the parameter `BCIndex`, which assigns boundary conditions to the surfaces of the boxes must be adapted.

Table 3.3: Multiple Cartesian Boxes.

Parameters	Setting	Description
`BCIn`	`(/ 0, 0, 0, 0, 0, 0/ )`	The `BCIndex` parameter assigns a boundary condition to each surface of the Cartesian box in the order of the surfaces. The number of the component of a vector represents the nth boundarycondition in order of its position in the file. Thus, each position refers to the six sides of the box `(/z-,y-,x+,y+, x-,z+/)`.In case of multiple Cartesian boxes, there are surfaces that coincide with other surfaces. No boundary condition can be assigned to such surfaces. Therefore, the number of the correspondingvector's component is set to 0. Here, all components of the parameter `BCIndex` are set to 0 (`(/0,0,0,0, 0,0/)`).This means that this box is surrounded by six other boxes, so that no boundary condition can be assigned to a single surface.

A description of all parameters of the parameter file can be found in *List of Parameters*.

3.1.3.2 Multiple Cartesian Boxes: Sketch

Figure 1.25 shows the sketch of the current problem. As one can see, the generated mesh shall consist of three Cartesian boxes. These zones are defined in the parameter file in the following order:

```
1st zone: lower left zone

            BCIndex setting: (/-,-,-,-,-,0/)

2nd zone: upper left zone

            BCIndex setting: (/0,-,-,-,-,-/)

3rd zone: upper right zone

            BCIndex setting: (/-,-,-,-,0,-/)
```

For a better understanding, the different settings of the parameter `BCIndex` are also given. The given settings only consider the components that are set to 0 due to coinciding surfaces.

3.1.3.3 Multiple Cartesian Boxes: Output Visualization

If you need help visualizing the HOPR output, visit the *Visualization* page.

A visualization of the mesh of the `cartbox_multiple_Debugmesh.dat` file is given below.

A visualization of the mesh of the `cartbox_multiple_Debugmesh_BC.dat` file is given below. The colors of the surfaces represent the boundary conditions and are the same as in the snippet of the parameter file.

3.1.4 Stretching Functions

This tutorial shows how to create a mesh consisting of a boxes with a stretched element arrangement. The parameter file can be found in

```
tutorials/1-04-cartbox_multiple_stretch/parameter.ini
```

3.1.4.1 Stretching Functions: Definition of Stretching Functions

Stretching functions can be used to generate a mesh consisting of boxes with a stretched element arrangement. Therefore, two new parameters can be defined in the parameter file: `factor` and `l0`. Each of them can be used to stretch the elements of a box. Their meaning and connection is shown as one-dimensional case in Fig. 1.28.

In case of a stretched element arrangement, the next element of a box is always stretched by a factor \(f\) in the direction of the coordinate axis. The value of factor \(f\) can have a positive or a negative sign. Here \(f\) has either a negative and an absolute value >1 or a positive sign and an absolute value <1. The length of the first element is called \(l_{0}\). All streched elements \(N\) together have the length \(l_{ges}\).

3.1.4.2 Stretching Functions: Building a Cartesian Box with Stretched Elements

To obtain a single Cartesian box with a stretched element arrangement, it is important to know that the parameters \(l_{0}\) and \(N\) are defined even before the stretching parameters `factor` and `l0` are defined. The length \(l_{ges}\) is defined by the boundaries of the Cartesian box and the number of elements per box in the direction of the Cartesian coordinate axes, hereafter called \(N\), is defined by the parameter `nElems`.

For a stretched element arrangement either the parameter `factor` or the parameter `l0` must be defined. The other missing parameter is calculated internally with the following equation:

$$\frac{l_{ges}}{l_0} = \sum_{i=1}^{N} f^{i-1} = \frac{1 - f^N}{1 - f}$$

The structure of the two parameters is explained below. A description of all parameters in the parameter file can be found in *List of Parameters*.

Table 3.4: Stretching Functions.

Parameters	Setting	Description
`fact`	`(/-1.75,1,-1.5/)`	Stretching factor of the elements in the direction of the Cartesian coordinate axes. A value >1 means an increase of the element size in the direction of the coordinate axis, while a value in theintervall (0,1) means a decrease. A value of 1 has no effect on the element sizes, and a value of 0 disables the stretching function for this axis. Furthermore, the stretchingbehavior can be mirrored by adding a negative sign to the values. A combination with the parameter `l0` ignores the element number of the defined box.In case of `(/-1.75,1,-1.5/)` each following element is compressed by a factor of 1.75 in the direction of the x-axis and 1.5 in the direction of the z-axis. The element arrangementalong the y-axis is not changed.
`l0`	`(/0,1,5,0/)`	The length of the first element of a stretched element arrangement of a Cartesian box. Each component of the vector represents an axis of the Cartesian coordinate system. A value of 0 disablesthe stretching function for that axis. A negative sign defines the length of the first element of the other side of the box. A combination with the parameter `factor` ignores theelement number of the defined box. Here, the stretching function is disabled for the x- and z-axis, while the first element in the direction of the y-axis has a size of 1.5.

There are several ways to get a stretched element arrangement due to the two different stretching parameters `factor` and `l0`. These cases are:

- Definition of `factor`: In this case, the next element of a box is stretched by a factor \(f\) in the direction of the coordinate axis. The parameter `l0` is calculated internally by the equation provided above. The number of elements \(N\), defined by the parameter `nElems`, remains unchanged.
- Definition of `l0`: In this case, the length of the first element (or of the last, if the sign is negative) in the direction of the coordinate axis is defined. The parameter `factor` is calculated internally by the equation provided above. The number of elements \(N\), defined by the parameter `nElems` remains unchanged.
- Definition of `factor` and `l0`: In this case, the parameters must be defined manually by the equation provided above. Otherwise, the stretched element arrangement will most likely not achieved the desired shape. In case of an inaccurate definition, the parameter `factor` will ne adjusted to the parameter `l0` which means that `factor` will changed internally. Furthermore, the number of elements \(N\), defined by the parameter `nElems` will probably not be kept. Instead, \(N\) will be rounded to nearest natural number.

These three different cases are illustrated below using a small cube with an edge length of one and with four elements per axis. For a better understanding, only the x- and y- values have been changed and visualized.

3.1.4.3 Stretching Functions: Building Multiple Cartesian Boxes with Stretched Elements

To create a mesh consisting of multiple Cartesian boxes with a stretched element arrangement at least one of the parameters `factor` and `l0` must be defined for each Cartesian box. The reason for this is that if there is to be a contact between two boxes, the corner nodes of the surfaces have to coincide. This means that defining a stretch function for one Cartesian box leads to the need of a stretch function for the neighboring Cartesian box. A visualization of this issue can be seen in the sketch of the tutorial's problem.

3.1.4.4 Stretching Functions: Sketch

The following is an example of a mesh of multiple Cartesian boxes with a stretched element arrangement. The corresponding parameter file can be found in

```
tutorials/1-04-cartbox_multiple_stretch/parameter.ini
```

The arrangement of the Cartesian boxes is the same as in the Multiple Cartesian Boxes tutorial but instead of equidistant elements, a stretched element arrangement is created by inserting the parameters `factor` and `l0`. Furthermore, the number of elements of each box has been increased by changing the parameter `nElems` to better visualize the stretched element arrangement. The figure below shows how the elements are stretched.

Note that the figure above only shows in which directions the elements are stretched or compressed. Neither the number of elements per box nor the Cartesian box sizes shown correspond to the parameters of the parameter file.

3.1.4.5 Stretching Functions: Output Visualization

If you need help visualizing the HOPR output, visit the *Visualization* page. A visualization of the mesh of the `cartbox_multiple_stretch_mesh.h5` file is given below.

3.2 Curved Meshes

3.2.1 Curved Structured Mesh

This tutorial shows how to generate a curved structured mesh with an equidistant or with a stretched element arrangement alternatively. The parameter file can be found in

```
tutorials/1-05-curved_structured/parameter.ini
```

To generate a curved structured mesh the following parameter settings are mandatory:

- `Mode=11` (curved structured block with hexahedral elements): This mode activates a transformation of the cartesian coordinate system to a rotated cylindrical coordinate system as shown in Fig. 2.1. The element distribution which the user can determine by the parameter `nElems` refers subsequently to the new coordinate system.
- `nZones=1`
- `MeshType=3` (for curved mesh)

The HOPR user has to choose whether he wants to generate a half or a full cylindrical mesh. Therefore the new parameter `WhichMapping` is provided. For specifying the general shape of the (half) cylinder three parameters are provided: `R_0`, `R_INF` and `DZ`. Their meaning is visualized in Fig. 2.2. It must be taken into account that the value for the inner radius (`R_0`) must not be zero and the value for `DZ` corresponds to the half thickness of the (half) cylinder.

The assignment of the boundary conditions to the surfaces refers to the new coordinate system by the parameter `BCIndex (z-,y-,x+,y+,x-,z+)`. For the case that the parameter `WhichMapping` is set to 4 (full cylindrical mesh), the third and fifth surface (x+, x-) coincide and the corresponding components of the `BCIndex` vector have to be set to zero. An overview of the mentioned parameters is given below. A description of all parameters can be found in *List of Parameters*.

Table 3.5: Curved Structured Mesh: Overview of parameters.

Parameters	Setting	Description
`Meshtype`	`3`	1: Cube (origin + dimensions)2: Bilinear (8 points CGNS notation)3: Curved (add WhichMapping)
`WhichMappin`	`4`	Type of mapping using 6 boundary faces to build the curved structured mesh:3: Half cylinder4: Full cylinder
`R_0`	`0.5`	Inner radius of curved structured mesh. The value 0 is not allowed.
`R_INF`	`20`	Outer radius of curved structured mesh.
`DZ`	`2`	Dimension in z-direction: `[-DZ,DZ]`

3.2.1.1 Stretching Functions

Similar to straight-edged boxes one can generate curved structured meshes with a stretched element arrangement. For this purpose, three additional parameters have to be defined in the parameter file: `stretchType`, `fac` and `DXmaxToDXmin`. Each parameter is defined for each axis in the rotated coordinate system as a vector (x,y,z). The parameters are explained below.

Table 3.6: Curved Structured Mesh: Stretching functions.

Parameters	Setting	Description
`stretchType`	`(/3,1,0/)`	(De)activation of the stretching functions for cylindrical coordinate axis (0: Stretching is deactivated)
		1: Stretching with a factor
		2: Stretching with a length ratio
		3: Stretching with a bell function
`fac`	`(/1.5,2.2,10/)`	Stretching factor of the elements in the direction of the cylindrical coordinate axis
`DXmaxToDXmi`	`(/6.,100.,1./)`	Frame ratio of the maximum to the minimum element size

A stretching factor `fac` of greater 1 means an increase of the element size in the direction of the coordinate axis, a value of the intervall (0,1) means a decrease. The value 1 does not affect the element sizes and means an deactivation of the stretching function for this axis. The value 0 is only allowed if the stretching function for this axis is deactivated (`stretchType` vector component for this axis is 0). Furthermore the stretching behaviour can be mirrored by adding a negative sign to the values. If the `stretchType` vector component for an axis is 3, the factor will be multiplied by -1 if the half distance is reached. In addition, `fac` has not the significant influence on the element arrangement anymore but the parameter `DXmaxToDXmin`. In case of `(/1.5,2.2,10/)` each following element in x-direction is stretched by the factor 1.5, in y-direction by the factor 2.2 and in the direction of the z-axis by the factor 10 (dependent on `stretchType`)

If the `stretchType` vector component for an axis is 3, the element arrangement is affected significantly by `DXmaxToDXmin` instead of the parameter `fac`. In case of `(/6,100,1/)` the maximum element size in x-direction can be 6 times larger than the minimum element size. In y-direction the maximum element size can be 100 times larger than the minimum element size. The value 1, here set for ratio of the z-direction, is used typically for a deactivated stretching

Calculation Formulas

For a better understanding how the element sizes are calculated, the formulas for different `stretchType` settings are shown below.

- Calculation of the element size for `stretchType = 1`:

$$\Delta x_{i+1} = f \cdot \Delta x_i f = \left(\frac{\Delta x_{max}}{\Delta x_{min}}\right)^{1/(nElems-1)}$$

- Calculation of the element size for `stretchType = 3`:

$$\Delta x(\xi) \sim 1 + \left(\frac{\Delta x_{max}}{\Delta x_{min}} - 1\right) \cdot \left(\frac{\exp[-(\xi \cdot f)^2] - \exp[-f^2]}{\exp[0] - \exp[-f^2]}\right)$$

Exemplary Stretching Cases

Furthermore, three different stretching cases are presented below with a full circle (`WhichMapping=4`) and an element distribution `nElems=(/8,6,4/)`, where only x- and y-values were visualized.

3.2.1.2 Examples

In the following two exemplary curved structured meshes are presented. The first mesh shall consist of twelve equidistant elements in x-direction, eight elements in y-direction and four elements in z-direction. The sketch of this problem is shown in Fig. 2.8. The second mesh consists of the same number of elements in each direction but with stretched elements. This sketch is presented in Fig. 2.9.

Curved Structured Mesh without Stretched Elements

Curved Structured Mesh with Stretched Elements

3.2.2 Mesh Curving by Post-Deformation

This tutorial shows how to generate a curved multi-block mesh, composed of several structured boxes, which are first assembled and then globally mapped to a curved domain. The parameter file can be found here:

```
tutorials/1-06-curved-postdeform/parameter.ini
```

Here, user-defined variables are used to parametrize the parameter file. They are searched and replaced in **all other lines(!)** of the parameter file (all strings between the = and ! sign are searched). They are either an Integer or Real value and defined in the parameter file as

```
DEFVAR=(INT):    i0 = 002    ! no. elems in inner square  i0xi0
DEFVAR=(REAL):   ri = 0.5    ! inner square dim
```

Note that each variable is searched and replaced one sequentially, so that names should be absolutely unique. In the example, a variable called `ri0` would not be allowed.

3.2.2.1 Post-Deformation from a box to a cylinder

The idea is to build first a simple box using the internal mesh procedures explained in Multiple Cartesian Boxes and then use a deformation function to obtain a cylinder. The post-deformation parameter is

```
MeshPostDeform=1
```

The undeformed and deformed mesh is shown in Fig. 2.14 and Fig. 2.15.

The order of the curved element mapping can be chosen arbitrarily

```
useCurveds   =T
BoundaryOrder=5
```

The mapping function maps the xy [-1;1]^2 coordinates to a circular domain of radius 1, but smoothed towards the center to avoid a singular mesh. The radius can be scaled with the parameter

```
MeshPostDeform_R0=1.0
```

We choose a periodic boundary condition in z direction.

3.2.2.2 Parameter Variations

In a variant of the parameter file, parameter2.ini, the extent of the domain in xy is [-2;2]^2 and is mapped to a circular domain with a radius of 2. The part of the domain inside [-1;1]^2 is mapped like in the example above, but ouside of [-1;1]^2, the mapping is perfectly circular. A final radius of 1 is then achieved by setting the scaling factor to, see Fig. 2.16:

```
MeshPostDeform_R0 = 0.5
```

In another variant of the parameter file, parameter3.ini, a mesh with 9 zones in built, and refined at a specific radius, using the stretching functions explained in Stretching Functions, see Fig. 2.17.

3.2.3 Curved Torus

The same post-deformation is applied to generate a torus. The parameter file can be found in

```
tutorials/1-07-curved-torus/parameter.ini
```

3.2.3.1 Post-Deformation from a box to a torus

Analogously to the previous tutorial Mesh Curving by Post-Deformation, we deform a box to a torus with a circular cross section. We only add the main radius of the torus as a parameter

```
MeshPostDeform    = 1                            ! deforms [-1,1]^2 to a cylinder with␣
↪radius Postdeform_R0
PostDeform_R0     = s0                           ! here domain is [-2,2]^2 mapped to a␣
↪cylinder with radius 0.5*2 = 1
PostDeform_Rtorus = rz                           ! z must be inside [0,1] and periodic
```

Since the connectivity of the mesh is created before the deformation, the boundary condition in z direction must be periodic on the undeformed mesh. The torus then has the correct connectivity.

3.2.4 Curved Sphere

The same post-deformation is applied to generate a mesh of a sphere and a spherical shell. The parameter file can be found in

```
tutorials/1-08-curved-sphere/parameter.ini
```

3.2.4.1 Post-Deformation from a box to a sphere

Analogously to the tutorial Mesh Curving by Post-Deformation, we deform a box to a sphere.

```
MeshPostDeform=2
PostDeform_R0=0.5
```

The initial box consists of 1 central zone and 6 neighbor zones, and forms a cube of [-2;2]^3 , being mapped to a sphere of radius 2. Again, `PostDeform_R0` can be used to scale the radius. The mapping of the domain inside [-1;1]^3 is again smoothed to avoid singular elements, and outside [-1,1]^3 is perfectly spherical, see Fig. 2.20 and Fig. 2.21 .

3.2.4.2 Spherical shell

In a variant of the parameter file, parameter_shell.ini, only 6 domains without the central domain are used and a spherical shell is generated. The boundary conditions have to be changed, and the central hole has a size of [-1,1]^3, see Fig. 2.22 and Fig. 2.23, where also the inner boundary face is shown.

CHAPTER

FOUR

EXTERNAL MESHES

In HOPR, it is possible to read unstructured meshes with straight edged elements, and, if needed, use several curving methods to account for curved domain boundaries.

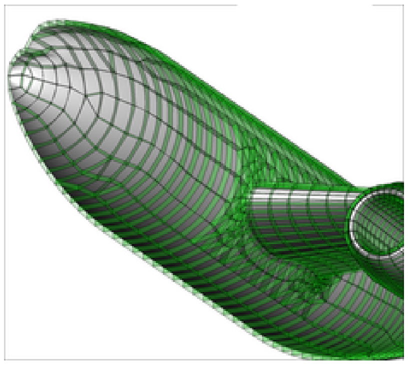


Fig. 4.1: HOPR output: Curved surface and first layer of curved elements

To help the HOPR user to get familiar with these different features for external meshes three hands-on tutorials are provided.

4.1 External Meshes without Curved Boundaries

This tutorial shows how to read in externally generated unstructured and structured meshes with straight-edged elements.

The parameter file can be found in

```
tutorials/2-01-external_mesh_CGNS_sphere/parameter.ini
```

4.1.1 External Mesh

The external mesh file that is to be used has to be available in the directory of the executed parameter file as a CGNS format file. This file is read-in by introducing the parameter `filename`. As one can see from the parameter.ini's excerpt and Fig. 1.1, the parameters of the parameter file have to be adapted to the definitions in the CGNS mesh file. This means that the parameters `Mode`, `nZones`, `BoundaryName` and `BoundaryType` cannot be set freely anymore because the structure of the external mesh must be retained. In this case, the external mesh spheremesh02 is available as CGNS file and consists of three zones. Therefore, the settings of the parameters are `Mode=3`, `nZones=3`, `filename=spheremesh02.cgns`.

Another important fact is that for external meshes no `BCIndex` parameter is needed which normally assigns the boundary conditions to the surfaces of the mesh. The reason for this is that the boundary conditions are assigned to their belonging surfaces by their names. The boundary condition, for example, of `Zone_1` of the CGNS file (`BC_sphere`) has to be defined as `BoundaryName=sphere` in the parameter file.

```
!================================================================= !
! MESH
!================================================================= !
Mode    =3                  ! 1 Cartesian 3 CGNS 4 STAR-CD V3
nZones  =3                  ! number of zones
filename=spheremesh02.cgns  ! name of mesh file
...
...

!================================================================= !
! CURVED
!================================================================= !
useCurveds=F                ! T to generate curved boundaries


!================================================================= !
! BOUNDARY CONDITIONS
!================================================================= !
BoundaryName=sphere         ! BC_Name must be defined in mesh file
BoundaryType=(/4,1,0,0/)
BoundaryName=inflow
BoundaryType=(/2,0,0,0/)
BoundaryName=outflow
BoundaryType=(/2,0,0,0/)
BoundaryName=mantel
BoundaryType=(/2,0,0,0/)
```

Furthermore, the `BoundaryType` parameter has to be adapted to the definitions in the CGNS mesh file. If a boundary of the external mesh is curved, the `curveIndex` component (2nd component) of the `BoundaryType` parameter has to be a value unequal to zero. Whether curved boundaries are generated or not is controlled by the parameter `useCurveds`. In this tutorial, `useCurveds=F`. The case `useCurveds=T` is the topic of the next tutorial which explains how to use mesh curving techniques to generate curved boundaries.

All new parameters of the parameter file of this tutorial are explained below.

Table 4.1: External Mesh Parameters.

Parameters	Setting	Description
`filename`	`spheremesh.cgns`	The name of the external mesh file. The necessary files have to be available in the directory of the executed parameter file as CGNS files.
`meshscal`	`0.001`	Scales all input mesh coordinates by a fixed factor
`SpaceQua`	`1000`	Characteristic length of the mesh
`useCurve`	`T`	T (True): If curved boundaries are definedF (False): If no curved boundaries are defined .

A description of all parameters of the parameter file can be found in *List of Parameters*.

4.1.2 Output Visualization

If there is a need for assistance of visualizing the HOPR output visit *Visualization*.

The figures below show the visualizations of the SPHERE_Debugmesh.vtu file. In addition, a visualization of the surfaces the first boundary condition sphere was assigned to (the `curveIndex` of the `BoundaryType` parameter is set to 1) of the SPHERE_Debugmesh_BC.vtu file is shown for each external mesh (see Fig. 1.4, Fig. 1.7, Fig. 1.10).

4.2 External Meshes with Curved Boundaries

This tutorial shows how several methods can be applied to curve the boundary faces of the elements.

The parameter file can be found in

```
tutorials/2-02-external_mesh_CGNS_sphere_curved/parameter.ini
```

4.2.1 Mesh Curving Techniques

In the development of next generation numerical methods for CFD, high-order methods are promising a substantial increase in efficiency and accuracy. While the particular high-order methods can be very distinct, they have in common that they must rely on a high-order approximation of curved geometries to maintain their high-order of accuracy. The generation of curved meshes is thus a topic the importance of which cannot be overstated, if one truly wants to apply high order methods to problems with industrial relevance. Especially aerospace applications heavily rely on complex geometries and pose high requirements to the quality of geometry representation.

HOPR provides several strategies to produce high-order meshes, relying on linear meshes, which are the standard of today's state-of-the-art meshing software and can be generated by commercial mesh generation tools. The two main strategies can be selected with the parameter `curvingMethod`. Therefore, the parameter `useCurveds` has to set to `T`.

Table 4.2: Mesh Curving Techniques Parameter.

Parameters	Setting	Description
`curvingMeth`	1	0: No curving method activated.1: Curving with normal vectors at surface points.3: Curving with subdivided surface mesh.

4.2.2 Curving Using Normal Vectors

Using normal vectors at the surface points, one can reconstruct the curved boundary face. Normal vectors can either be reconstructed from the existing coarse mesh, given in a point-normal file or prescribed analytically. The way of providing the normal vectors can be selected with the parameter `NormalsType`. For each setting, a different number of new but self-explanatory parameters are mandatory. These parameters are listed below.

It has to be taken into account that for generating the curved boundary splines, the parameter `boundaryOrder` has to be always set to 4 for a normal vector approach. Otherwise HOPR will maybe not work correctly.

Below, all parameters which are mandatory for this curving technique are explained. A description of all parameters can be found in *List of Parameters*.

Table 4.3: Curving Using Normal Vectors: Description of Parameters.

Parameters	Setting	Description
`Normals`	1	Source of the normal:1: Reconstructed from coarse surface mesh (no additional parameters, `CurveIndex` of BC must be >0).2: `NormalVectFile` (point normal vector file) needed3: Analytical normals from surface point positions
`NormalV`	`file`	Number of association between BC `CurveIndex` and analytical normal. Mandatory if `NormalsType=3`
`nExactN`	1	Number of association between BC `CurveIndex` and analytical normal. Mandatory if `NormalsType=3`
`ExactNo`	(/ 1, 1/)	`(/ BC curveIndex, number of analytical formula/)`. Build in formulas (see src/mesh/curved.f90):1: Sphere with origin (0,0,0)2: Cylinder around z-axis Mandatory if `NormalsType=3`
`boundar`	4	Not used here, fixed to cubic polynomial

The figures below show the visualization of the boundary sphere (`BoundaryType=(/4,1,0,0/)`) of the meshes spheremesh01.cgns (Fig. 2.4 - Fig. 2.6), spheremesh02.cgns (Fig. 2.8 - Fig. 2.10) and spheremesh04.cgns (Fig. 2.12 - Fig. 2.14) for the three different normal vector approaches. Therefore, the SPHERE_CURVED_SplineSurf.vtu file was used. Furthermore, for the second normal vector approach (`NormalsType=2`) the point normal vector file normals_out_000001.dat was used (called filename in the captions). For the purpose of comparison the surfaces of the original meshes the boundary sphere were assigned to are also shown (Fig. 2.3, Fig. 2.7, Fig. 2.11) and were visualized with the SPHERE_CURVED_Debugmesh_BC.vtu file.

If there is a need for assistance of visualizing the HOPR output visit *Visualization*.

4.2.3 Curving Using Subdivided Surface Mesh

Here, we need an additional surface mesh file, which contains a surface mesh (in CGNS format) generated by a mesh generator from the initial coarse surface mesh by subdivision. The additional surface points are lying on the CAD geometry and are then used as interpolation points for the polynomial description of the element face.

Below, all parameters which are mandatory for this curving technique are explained. A description of all parameters can be found in *List of Parameters*.

Table 4.4: Curving Using Subdivided Surface Mesh: Description of Parameters.

Parameters	Setting	Description
`SplitEl`	`filen`	Name of suvdivided surface mesh. Mandatory if `curvingMethod=3`
`boundar`	5	Order of spline-reconstruction for curved surfaces, corresponding to the number of subdivisions1 x subdivided: `boundaryOrder=3`2 x subdivided: `boundaryOrder=5`3 x subdivided: `boundaryOrder=9`…

The figures below show the visualization of the SPHERE_Debugmesh_BC.vtu file by using the mesh file spheremesh01.cgns (Fig. 2.3 - Fig. 2.6), spheremesh02.cgns (Fig. 2.7 - Fig. 2.10) and spheremesh01.cgns (Fig. 2.11 - Fig. 2.14). In addition, for each file three examples with a different subdivision of the surface mesh are given.

If there is a need for assistance of visualizing the HOPR output visit *Visualization*.

4.2.4 Use of pre-curved meshes

In some situations, the input mesh is already curved, e.g. when HOPRs own meshes are used for input or if the mesh generator already provides curved meshes (e.g. Gmsh). HOPR will read these meshes and directly use their high-order information. Existing high-order meshes from other sources can thus be directly translated into the HOPR format. Furthermore, HOPR's various mesh post-processing capabilities can be applied to these meshes and the mesh curving can optionally also be redone by using the normal vector and the subdivision approach described above.

Below, the mandatory parameters for reading a Gmsh mesh file are described. Note that in case of Gmsh meshes HOPR currently only supports linear pyramids and prisms, while tetrahedra and hexahedra are supported up to an order of 4. A description of all parameters can be found in *List of Parameters*.

Table 4.5: Use of pre-curved meshes: Description of Parameters.

Parameters	Setting	Description
`Filename`	`cylinder msh`	Name of mesh file.
`boundary`	`4`	Order of the geometry approximation in the mesh, i.e. the number of interpolation points per direction. Note: Gmsh denotes the polynomial degree by the term "order".

The figure below shows the visualization of the CYLINDER_SplineVol.dat file by using the mesh file cylinder.msh.

CHAPTER

FIVE

AGGLOMERATION OF BLOCK-STRUCTURED MESHES

A simple strategy to create a fully curved mesh is to use block-structured meshes (must be given in structured CGNS format) and agglomerate linear elements to high order elements.

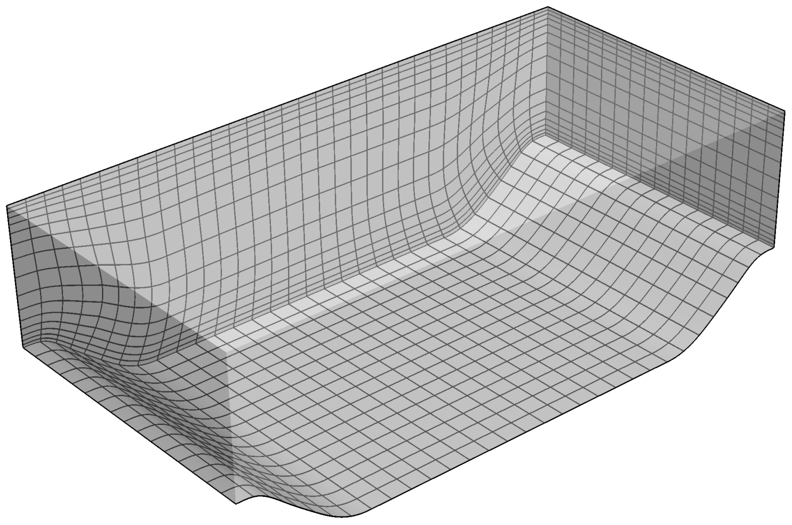


Fig. 5.1: Example of an agglomerated curved mesh of the periodic hill testcase

5.1 Block-Structured Meshes

This tutorial shows how to agglomerate block-structured grids with linear elements to get a high-order mesh consisting of fully curved hexahedral elements. The parameter file can be found in

```
tutorials/3-01-agglomeration_NACA/parameter.ini
```

5.1.1 Restrictions on the block-structured meshes

HOPR poses the following restrictions on the provided block-structured meshes:

- **One block face** must be exclusively associated to **one boundary condition**. Block faces with multiple BCs or a combination of boundary and internal element faces on a single block face are likely to produce wrong mesh topology! The user is responsible for **splitting blocks** in such cases. Otherwise, HOPR will be unable to perform the mesh connect step!
- When providing a block-structured mesh for agglomeration, only one **unique factor** is permitted in the `I,J,K` dimension of **all blocks**!

Each boundary conditions present in the provided mesh must be matched in the parameter file by its name and an associated boundary type:

```
        BoundaryName = wall_1
        BoundaryType = (/4,0,0,0/)
        BoundaryName = wall_2
        BoundaryType = (/4,0,0,0/)
```

Boundary names can match multiple BCs by specifying only the common part of the boundary name string. In the example above, `BoundaryName=wall` would match all boundary conditions.

5.1.2 Initial Meshes

Agglomeration of block-structured meshes provides a simple and robust curving technique for the generation of three-dimensional high-order meshes. In HOPR, mesh agglomeration is controlled by the `MeshIsAlreadyCurved` parameter. The application of mesh agglomeration is demonstrated on two meshes of a NACA-profile provided as CGNS files. Here, mesh 1 is suitable for inviscid simulations without boundary layer refinement and mesh 2 features boundary layer refinement.

All of the following figures were created with mesh 1.

```
MeshIsAlreadyCurved = T
useCurved           = T
BoundaryOrder       = 5
```

which lead to a coarsening in all three dimensions of the structured mesh, using the internal points as interpolation points for the curved mapping. The number of elements in each direction of the structured block must be a multiple number of `BoundaryOrder-1`! This situation is illustrated on an exemplary mesh in Fig. 1.5. For `BoundaryOrder=2`, the initial linear mesh retained without agglomeration.

In addtion, the parameter `nSkip` can be set to coarsen the initial mesh.

5.1.3 Description of Parameters

The following table describes all parameters associated with agglomeration. A description of all parameters is given in *List of Parameters*.

Table 5.1: Block-Structured Meshes: Description of Parameters.

Parameters	Setting	Description
`MeshIsAlreadyCurve`	`T`	Enables the agglomeration
`nSkip`	`2`	Coarsing of block-structured meshes1: no skip2: use every second point…
`nSkipZ`	`2`	Only if the mesh is extruded in z-direction, a different nSkip can be given in z-direction.

Mesh coarsening is controlled by two parameters: `nSkip` applies to all structured directions equally, and `nSkipZ` can be used for z-extruded meshes.

The figure on the left side shows the initial mesh. The parameter `nSkip=1` uses every point of the initial mesh. The mesh in the middle-left shows the mesh if `nSkip` is set to 2. That means that one node in each direction of the respective coordinate system is skipped and that the size of the new element reaches to the next node. The skipped nodes will no longer be used for mesh generation, including curving. The parameter `nSkipZ` has the same function as `nSkip` but only applies towards the z-direction. By providing the `nSkipZ` parameter, the corresponding entry in `nSkip` is ignored.

The following figures illustrate possible combinations of the `nSkip` and `nSkipZ` parameters, each applied to the identical initial mesh. Keep in mind that reading in a block-structured mesh only works if the value for the parameter(s) `nSkip` (and `nSkipZ`) is a common divisor of the number of all mesh elements for each axis.

Extruded meshes often suffer from issues with limited coordinate precision, preventing HOPR from finding corresponding surface elements on the upper/lower boundary. HOPR included a built-in correction routine for such cases. This function is not restricted to block-structured meshes, but can be applied to all meshes which are extruded along the z-direction. Enabling this setting requires five more parameters to be set.

Table 5.2: Correction for z-extruded meshes: Description of parameters.

Parameters	Setting	Description
`doZcorrection`	`F`	All elements are aligned exactly along z-direction to suppress grid generator errors
`nElemsZ`	`1`	The number of elements in z-direction (after agglomeration!)
`zStart`	`0.`	Set minimum z-coordinate
`zLength`	`1.0`	Set length of domain in z-direction
`zPeriodic`	`T`	Boundary conditions (`z_plus` and `z_minus`) are set to periodic.

CHAPTER

SIX

POST-PROCESSING MESHES

Once a mesh is built there are several post-processing options available.

6.1 Mesh uncurving

As high order numerical methods require a high order representation of the boundary, a number of methods have been developed to provide this level of accuracy. In some curving techniques, such as agglomerated meshes, it is not just the boundary that is curved, but the entire volume of the mesh is made up of curved cells. However, if the cell is not adjacent to curved boundaries, it is numerically preferable to use linear cells. Thus, HOPR provides a function for uncurving curved meshes, optionally at a given distance to the boundary. The parameter `nCurvedBoundaryLayers` specifies the number of cells from a curved boundary (i.e. boundaries with `curveIndex>0`) that stay curved. For the remaining mesh, a (tri-)linear mapping using only the corner nodes is applied.

- For `nCurvedBoundaryLayers=-1` the whole mesh stays curved (default).
- For `nCurvedBoundaryLayers=0` in the first cell only the boundary itself remains curved, all other faces and cells in the mesh will be linear.
- For `nCurvedBoundaryLayers=1-n` the first n cells away from the boundary remain curved.

These choices are depicted in the figures bellow, and the table lists the scaled Jacobian ranges for the elements. The mesh becomes significantly less distorted if only the first cell is curved. Note that it is often required to curve more than the first cell, especially for fine curved boundary layer meshes.

Table 6.1: Distribution of the smallest scaled Jacobians per element.

Number of elements with scaled Jacobians ranging between:	<0	<0.1	<0.2	<0.3	<0.4	<0.5	<0.6	<0.7	<0.8	<0.9	<1.0
Only boundary is curved:	0	0	0	0	8	4	8	208	100	276	1124
First element is curved:	0	0	0	0	8	4	16	208	100	288	1104
First 3 elements are curved:	0	0	0	0	16	12	32	224	92	284	1068
All elements are curved:	0	0	0	0	136	596	748	200	12	36	0

To test this feature, add `nCurvedBoundaryLayers=n` to the parameter file, which is also found in

```
tutorials/3-01-agglomeration_NACA/parameter.ini
```

6.2 Mesh Refinement

It is often desirable to refine existing meshes, which is done by subdividing the elements into smaller elements. Up to now, the refinement only works for hexahedra with linear edges. For meshes containing other element types than hexahedra, this option is not applicable. The feature is controlled by the flag `nFineHexa=x`, where x specifies the number of subdivisions of each element in each spatial direction.

To test this feature, add `nFineHexa=2` to the parameter file, which is also found in

```
tutorials/2-01-external_meshes_sphere/parameter.ini
```

6.3 Generation of Hexahedral Meshes

Since many solvers require purely hexahedral meshes, HOPR implements a subdivision strategy to split meshes consisting of tetrahedra, prisms and hexahedra into purely hexahedral meshes. This feature is activated using the parameter `splitToHex=T`. Note, that pyramids cannot be decomposed to hexahedra in a straightforward way, thus this feature cannot be applied to meshes containing pyramids. Also note that this option is currently limited to linear meshes.

To test this feature, set `elemtype` to either `104` (Tetrahedron) or `106` (Prism with triangular base) and add `splitToHex=T` to the parameter file, which is found in

```
tutorials/1-01-cartbox/parameter.ini
```

CHAPTER

SEVEN

VISUALIZATION

7.1 HOPR Output Parameter

Table 7.1: HOPR Output Parameter.

Parameters	Example	Data Type	Array Dim.	Default Value	Description
`Debug`	`Debugv`	Logical	1	F	T (True): Files will be generated, which enable you to visualize the mesh and the boundary mesh for debugging. These files can be found in the directory of the executed `parameter.ini` file.F (False): Files for visualization will not generated during executing of the `parameter.ini` file.
`Debug`	`Debugv`	Int	1	0	0: Visualization of linear mesh and BC (default).1: Visualization of linear mesh and BC and an additional curved surface visualization (`_SplineSurf.*`) if `useCurveds=T`.2: Visualization of linear mesh and BC and an additional curved volume visualization (`_SplineVol.*`) if `useCurveds=T`.
`NVisu`	`NVisu=`	Int	1	0	Number of visualization points per element edge if `useCurveds=T`.
`outpu`	`output`	Int	1	0	0: Paraview vtk (ASCII)1: Tecplot (ASCII)2: CGNS (binary)
`Proje`	`Projec`	Str	1	OBLIGATORY	Part of the output files' name which will be generated during the execution. These Files can be found in the directory of the executed `parameter.ini` file.

A description of all parameters of the parameter file can be found in *List of Parameters*

7.2 Visualization with Paraview

Open source multiple-platform application for interactive, scientific visualization. For more Information visit https://www.paraview.org/

7.2.1 Parameter Settings

```
Debugvisu=T
outputFormat=0
```

7.2.2 Recommended Settings

ParaView is susceptible for defective output visualizations in cases of using high order meshes or increasing the polynomial degree of supersampling. For this reason, it is recommended to edit several settings to get a correct high resolution visualization. At first the way of projection should be changed. Therefore, go to Edit > View Settings … (Fig. 1.2) and activate use parallel projection (Fig. 1.3).

If the polynomial degree of supersampling is higher than 1 the mesh elements will be subdivided depending of the polynomial degree. As a result, it is not apparent anymore where the elements begin and where they end. Because of that the two filters “Extract Surfaces” and “Extract Edges” have to be applied which make the actual elements visible again in high resolution. They can be found under Filters > Alphabetical.

CHAPTER

EIGHT

REFERENCES

BIBLIOGRAPHY

[1] F Hindenlang, T Bolemann, and C-D. Munz. Mesh Curving Techniques for High Order Discontinuous Galerkin Simulations. In *IDIHOM: Industrialization of High-Order Methods-A Top-Down Approach*, pages 133–152. Springer, 2015.