NaSt3DGP

A Parallel 3D Navier-Stokes Solver

User's Guide

Institute of Numerical Simulation
Division of Scientific Computing and Numerical Simulation
University of Bonn

Michael Griebel, Roberto Croce, Martin Engel

http://wissrech.iam.uni-bonn.de/research/projects/NaSt3DGP/index.htm
Contact: nast3dgp@ins.uni-bonn.de

Contents

Chapter 1

Introduction

NaSt3DGP is a `C++` implementation of a solver for the incompressible, time-dependent Navier-Stokes equations in three dimensions. It is based on a Chorin-type projection method and uses finite difference approximations for the spatial derivatives on non-uniform staggered meshes. Among the implemented features are Adams-Bashforth and Runge-Kutta methods for handling of the time derivatives and various upwind schemes like QUICK, SMART and VONOS for discretization of the convective terms. The calculation of temperature or concentration fields within the flow is also possible.
The code can be compiled to get a parallel version based on the message passing interface *MPI*, or to get a single processor version, which does not require any installation of *MPI*.
NaSt3DGP provides a user-friendly interface to describe the geometry and the various problem dependent parameters by a macro language. This interface also provides import and export capabilities for the numerical results. In addition, NaSt3DGP includes some utilities for the generation of smooth non-uniform grids and for the import and export of numerical results from/to various other software packages such as MatLab or Tecplot.

The remainder of this guide is as follows.

Chapter 2 explains the installation of NaSt3DGP and the basic usage is shown for the standard testcase of Driven-Cavity flow.

Chapter 3 describes the numerical scheme. This includes the description of the difference stencils used for the spatial discretization and the boundary conditions.

Chapter 4 introduces **navcalc** and **navcalcmpi** - the parts of NaSt3DGP which actually perform the numerical simulations. A basic outline of the implementation is given. This should help users to adapt the code for their own purposes if necessary.

Chapter 5 deals with **navsetup** - the interface tool. First the command line arguments are introduced. **navsetup** is able to read so-called scene description files which describe different parameters of the flow, the spatial discretization and a description of the geometry of the domain using a simple macro language. The key words of this language are given in subsection 5.2.

Chapter 6 explains various utilities that come with NaSt3DGP.

Chapter 7 shows some example simulations done with NaSt3DGP. The scenefiles for these simulations are included in the distribution.

Chapter 8 describes how to report bugs.

Chapter 9 contains the license under which NaSt3DGP is made available.

Chapter 2

Installation

NaSt3DGP uses the GNU autoconf and automake installation mechanism. Therefore, on most UNIX systems and on the Cygwin platform running under Windows, the usual process of

```
./configure
make
make install
```

should work. Detailed installation instructions and options to the configure script are given in the following sections.

2.1 The build and install process on UNIX platforms

NaSt3DGP is distributed as a standard UNIX tarball. First, you need to unpack the distribution. Depending on your version of tar, you can use one of the following commands to unpack the tarball to the current directory.

```
gzip -dc nast3dgp.tar.gz | tar xvf -
```

or

`tar xvzf nast3dgp.tar.gz` (if your version of tar support this).

Now change into the created subdirectory nast3dgp and issue the following command:

```
./configure [OPTIONS]
```

The command

```
./configure --help
```

will give you a complete list of available options with a short help message. The most important and NaSt3DGP-specific options are shown in table 2.1. Each option has a default value (also given in table 2.1) which will be used if you do not specify the corresponding option. Table 2.2 lists other available configure options.

Table 2.1: Some useful options to configure.

`--prefix=PREFIX`	with this option you can control where NaSt3DGP will be installed when the `make install` command is issued. PREFIX is a directory. You need write-access to PREFIX when installing the program. The default value for PREFIX is /usr/local/.
`--enable-mpi`	with this option you control whether the parallel version of NaSt3DGP based on MPI will be built. You need a proper installation of an MPI implementation for this to work. The default value is always yes, if configure can detect an installation of MPI(configure tries to do this when running). If the configure script is unable to find a MPI installation, parallel build will be disabled. You can explicitly disable building the parallel version of NaSt3DGP by specifying `--disable-mpi`.
`--enable-timer`	with this option you control the output of timing information. If enabled, additional profiling information will be generated. The default value is no. (Remark: this switch affects only the parallel version, it is ignored when building the serial version.
`--with-docdir=DIR`	with this option you can specify a separate directory for installation of the documentation. The default value for DIR is [PREFIX/share/doc/nast3dgp].

Further finetuning of the build process can be achieved by supplying additional environment variables on the configure command-line, i.e.

```
./configure [OPTIONS] VAR=VALUE
```

See table 2.3 for a list of used environment variables.
Perhaps the most important variable is `CXXFLAGS`. You can use it for instance to control the level of optimization(default optimization level depends on the compiler detected by configure, for GNU compilers it is usually `-O2`). For example, on a Linux system running

Table 2.2: Some more options to configure.

`-h, --help`	display this help and exit
`--help=short`	display options specific to this package
`--help=recursive`	display the short help of all the included packages
`-V, --version`	display version information and exit
`-q, --quiet, --silent`	do not print 'checking...' messages
`--cache-file=FILE`	cache test results in FILE [disabled]
`-C, --config-cache`	alias for '–cache-file=config.cache'
`-n, --no-create`	do not create output files
`--srcdir=DIR`	find the sources in DIR [configure dir or '..']
`--prefix=PREFIX`	install architecture-independent files in PREFIX [/usr/local]
`--exec-prefix=EPREFIX`	install architecture-dependent files in EPREFIX [PREFIX]
`--bindir=DIR`	user executables [EPREFIX/bin]
`--sbindir=DIR`	system admin executables [EPREFIX/sbin]
`--libexecdir=DIR`	program executables [EPREFIX/libexec]
`--datadir=DIR`	read-only architecture-independent data [PREFIX/share]
`--sysconfdir=DIR`	read-only single-machine data [PREFIX/etc]
`--sharedstatedir=DIR`	modifiable architecture-independent data [PREFIX/com]
`--localstatedir=DIR`	modifiable single-machine data [PREFIX/var]
`--libdir=DIR`	object code libraries [EPREFIX/lib]
`--includedir=DIR`	C header files [PREFIX/include]
`--oldincludedir=DIR`	C header files for non-gcc [/usr/include]
`--infodir=DIR`	info documentation [PREFIX/info]
`--mandir=DIR`	man documentation [PREFIX/man]
`--program-prefix=PREFIX`	prepend PREFIX to installed program names
`--program-suffix=SUFFIX`	append SUFFIX to installed program names
`--program-transform-name=PROGRAM`	run sed PROGRAM on installed program names

on a pentium class or higher processor, you could use the following command:

```
./configure [OPTIONS] CXXFLAGS='-Wall -ansi -O3 -mcpu=pentiumpro -ffast-math
-funroll-loops'
```

By default, configure picks the first compiler it detects on the hosts system. With the variable `CXX` you can explicitly specify the C++-compiler which should be used during the build process.
In order to build a parallel version of NaSt3DGP (see option `--enable-mpi` in table 2.1) you need a proper installation of an MPI implementation. You can download the free *MPI* implementation *MPICH* from the Computer Science Division of the Argonne National Laboratory **http://www-unix.mcs.anl.gov/mpi/mpich/**.
By specifying the variables `MPICXX` and `MPILIBS` you can build a parallel version of NaSt3DGP even if the configure script could not detect an installation of MPI in one of the standard locations. With some versions of MPI you can use the variable `MPICXX` to force the usage fo a specific C++-Compiler, for example

```
./configure [YOUROPTIONS] MPICXX='mpiCC -CC=/path/to/your/compiler/bin/g++'
```

For driver-scripts which do not support the `-CC` option (usually the driver-scripts of SCore don't) you might use the following workaround: just make sure that the compiler you want to use is the first to appear in your path, that is the one some driver-script use (e.g. SCore version 5 does).

After a successfull run of configure, just type

```
make
```

or

```
make -j 2
```

if building on a 2-processor machine. After the build process has successfully finished, type

```
make install
```

to install NaSt3DGP into PREFIX, where PREFIX is the directory you specified with the –prefix option when configuring. After installation you will find the following subdirectories and files located under PREFIX:

- in PREFIX/bin you will find the binaries **navsetup**, **vrml2nav**, **navcalc** and also **navcalcmpi** if the parallel version was built. **navsetup** is a multi-purpose interface tool for generating input files for **navcalc** and generating output for visualization.

Table 2.3: Additional environment variables for configure.

`CXX`	specify a C++ compiler other than the one detected by configure
`CXXFLAGS`	flags for the default compiler or the one specified with `CXX`
`LDFLAGS`	linker flags, e.g. -L ⟨lib dir⟩ if you have libraries in a nonstandard directory ⟨lib dir⟩
`CPPFLAGS`	C/C++ preprocessor flags, e.g. -I ⟨include dir⟩ if you have headers in a nonstandard directory ⟨include dir⟩
`CC`	C compiler command
`CFLAGS`	C compiler flags
`CXXCPP`	C++ preprocessor
`MPICXX`	MPI C++ compiler command
`MPILIBS`	necessary libraries to link MPI C code

A detailed description is given in chapter 5. The binaries **navcalc** and **navcalcmpi** are used to perform the actual numerical calculations. The program **vrml2nav** is used to convert geometry data in vrml 1.0 format into the nav-format readable by navsetup.

- in PREFIX/share/doc/nast3dgp you will find the documentation in Adobe PDF format.

- in PREFIX/share/nast3dgp/examples you will find several subdirectories containing example input files for **navsetup**.

- in PREFIX/share/nast3dgp/tools/GridGen you will find **GridGen**, a tool to generate smoothly distributed grid lines. More information on this tool is given in section 6.2. Please note that this tool is included only for convenience, it is not fully supported. **GridGen** is distributed in Fortran77 source code only and will not be generated automatically, you need an F77-compiler to build the binary.

- in PREFIX/share/nast3dgp/tools/matlab you will find Matlab-functions to read NaSt3DGP-generated data-files into Matlab and to write files in the same format

2.2 Build and install on Windows

Since NaSt3DGP is developed mainly on Unix-based systems the support for Windows is very rudimentary. We only provide a simple Makefile for the MS VC5 compiler which can be used with nmake. The file `Makefile.VC5` is included in the distribution in the directory NaSt3DGP/src.
Another way to use NaSt3DGP on Windows machines is to install a Cygwin-Environment. You can download Cygwin for free from **http://www.cygwin.com**. If you use it, simply follow the build process as described in the Unix-section.

2.3 Additional remarks and known problems

The current release of NaSt3DGP is known to succesfully install at least on the following systems and platforms:

- Intel 32-bit
  - Redhat Linux 7.2 with libc 2.2.4 and gcc versions 2.96, SCore 2.4.1
  - Redhat Linux 7.3 with libc 2.2.x and gcc 3.2, SCore 5.4
  - Redhat Linux 7.3 with libc 2.2.x and gcc 3.2, SCore 5.4
  - Redhat Linux 9.0 with libc 2.3.2 and gcc 3.0 or higher, MPICH 1.2.5
  - Debian 3.0 with libc 2.3.2 and gcc 3.2, MPICH 1.2.5
  - Redhat Linux 9.0 with libc 2.3.2 and icc 7.1 Build 20030822Z (Remarks: due to a bug in this and prior versions of icc, dependency tracking may not be fully functional)
- Intel 64-bit (Itanium)
  - Debian Linux with libc 2.2.5 and gcc 3.2, MPICH 1.2.5

2.3.1 Known issues

Some versions of MPICH and SCore provide buggy driver-scripts for the compiler. The scripts parse the command-line and alter it before handing it down to the actual compiler. This may lead to a false assumption about the style of dependency tracking. As a consequence, rebuilds of the source may fail with an error message similar to

```
cpp0: you must additionally specify either -M or -M
```

If this happens, there are several possible work-arounds and fixes:

- in the subdirectories mpi/ and src/ of the build-tree, delete the object files producing the error, that is, the ones corresponding to the source files that were changed, then rebuild with `make`
- supply the driver-script as the CXX compiler, i.e. use `./configure [YOUROPTIONS] CXX=mpiCC`
- disable dependency tracking by using the switch `--disable-dependency-tracking` when running configure.
- patch the driver scripts `mpic++` and `mpiCC` (when using SCore, they are located in a directory similar to `/opt/score/mpi/mpich-1.2.4/i386-redhat7-linux2_4_gnu/bin`

2.4 Running a first example

In this section the general usage of the software package NaSt3DGP is shown, involving all the steps from generating an input file until visualising the final output. For detailed information on the usage of `navsetup` and `navcalc/navcalcmpi` please refer to the corresponding chapters of this manual. Both programs show a basic help message when called with the option `-h`.

First, choose a directory where you want to store and and work with the generated data, e.g. `/home/my_home/Nast-Examples`. Change to this directory and copy the file `cavity.nav` from `PREFIX/share/nast3dgp/examples/Cavity2D` into this directory. The format and all parameters of this file are described in detail in section 5.2. Now execute `navsetup` in the following way:

```
navsetup -s cavity.nav -b cavity.bin
```

Remark: Depending on the installation PREFIX you choose, you may have to call `navsetup` with the complete path, i.e. `PREFIX/bin/navsetup` or else include the directory PREFIX/bin into your PATH environment variable. This holds for all subsequent calls to binaries of the NaSt3DGP package.

The setup program **navsetup** reads the configuration file `cavity.nav` which contains all necessary parameters for the description of the 'Driven Cavity'-testcase and generates a binary data file cavity.bin. This binary file contains all the data required for the subsequent computation with **navcalc**, i.e. the computational grid, initial values for $\mathbf{u}$ and p and other necessary parameters. The generated binary file can be used for both serial and parallel computations.

To start the actual simulation, use the command

Table 2.4: Sample output at startup of navcalc.

```
Reading file cavity.bin...
Time: 0, finish at 0.2
Gridpoints: 5120 (32x32x5), containing 0 obstacle cells
   maxgridsize=0.03125   mingridsize=0.03
   eps=1e-08   omg=1.7   alpha=0   alphatg=1
   tfdiff=0.5   tfconv=0.5
   reynolds=1000   froude=1
   gx=0   gy=0   gz=0
temperature is not calculated
chemicals are not calculated

periodic bound in z-direction.
            time handling:  Runge-Kutta 3rd
         iteration scheme:  BiCGStab
convective terms handling:  VONOS
```

```
navcalc -b cavity.bin -p3
```

for the serial version and the following one for running a parallel calculation on 2 processors:

```
mpirun -np 2 navcalcmpi -b cavity.bin -p3
```

Remark: the actual command for running a parallel calculation may differ from the one above, depending on your installation of MPI.
The first calculation runs only for ten timesteps in order to get results quickly.

At the beginning, navcalc prints some information on the calculation it is about to perform, you can use this output to check if the most important parameters in the scene description file are set correctly. When running in parallel mode, additional output about the distribution on different processors/machines is displayed. Sample output looks similar to the one shown in tables 2.4 and 2.5.
During the calculation `navcalc` prints some information on the screen(table 2.6), e.g. the actual time, the size of the timestep, the divergence of the velocity field and information about the residuals of the pressure poisson equation.

When the final time is reached, the data file `cavity.bin` is overwritten with the calculated data from the final time. The command

Table 2.5: Additional output of navcalcmpi.

```
2 (1/2/1) procs, I am 1 (0/1/0) (yourhost.yourdomain),
neighbours N:-1 S:-1 W:-1 E: 0 T:-1 B:-1.
Process 1 (1-32, 17-32, 1-5)  CompTemp:0  CompFrBd:0  CompChem:0  nChem:0
Number of started processes 2
2 (1/2/1) procs, I am 0 (0/0/0) (yourhost.yourdomain),
neighbours N:-1 S:-1 W: 1 E:-1 T:-1 B:-1.
Process 0 (1-32, 1-16, 1-5)  CompTemp:0  CompFrBd:0  CompChem:0  nChem:0
Init. Arrays
iobc1_vol: 0.000000e+00
volume of domain = 1.500000e-01
Build local Lists  0
inflow_surface = 0.075  outflow_surface = 0  flow_in = 0
Initialization done 0
```

Table 2.6: Runtime information of navcalc/navcalcmpi.

```
Step:     418 Time: 8.360000 TimeStep: 0.02000000000000  Time: 0h 5m 49.62s
  it:       1  res: 1.332181e-04    itmax: 100
  it:      51  res: 1.254539e-07    itmax: 100
Iter:      88  Res: 4.54237223e-09     all It:    38306
Mass Balance (rhs): 2.11625890e-01
Mass Balance (aps): 4.54237223e-09
flow_in=0  flow_out=0  mass_diff=0
Output...

Step:     419 Time: 8.380000 TimeStep: 0.02000000000000  Time: 0h 5m 50.27s
  it:       1  res: 1.299858e-04    itmax: 100
  it:      51  res: 1.233602e-07    itmax: 100
Iter:      85  Res: 9.69241550e-09     all It:    38391
Mass Balance (rhs): 2.11616888e-01
Mass Balance (aps): 9.69241550e-09
flow_in=0  flow_out=0  mass_diff=0
Output...
```

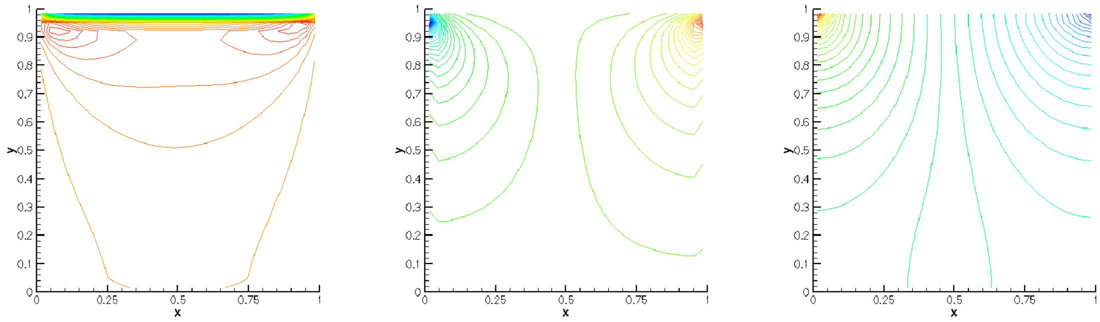

Figure 2.1: Contour lines of u,v and p (left to right) at time $t = 0.1$.

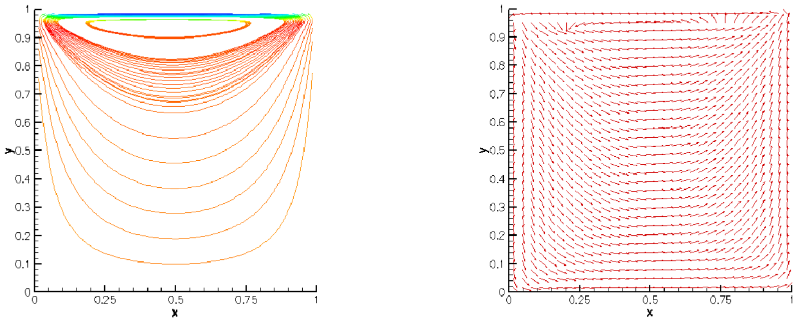

Figure 2.2: Streamlines and velocity vectors at time $t = 0.1$.

```
navsetup -TC cavity.bin -o results
```

generates a file named **results.dat** which is suitable for processing with Tecplot. The NaSt3DGP package supports several different output formats, e.g. for Matlab or VTK[1]. For details about different data conversion possibilities of **navsetup**, refer to chapter 5.

The results for the first, very short simulation are shown in figures 2.1 and 2.2.

Now edit the scene description file cavity.nav and increase the simulation time by setting a value of 250 for `Tfin`. Rerun this example by following the steps above for the new cavity.nav file[2]. Now the flof pattern does look like what you would expect for the

[1]The Visualization ToolKit, a C++ class library available at http://public.kitware.com/VTK/

[2]of course increasing the final time results in a longer computing time, on a 2-processor Xeon 933MHz machine the simulation takes about three quarters of an hour

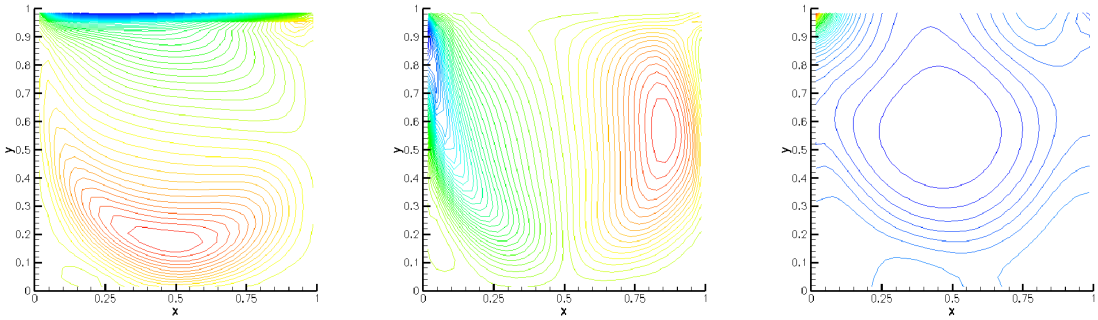

Figure 2.3: Contour lines of u,v and p (left to right) in steady state (time $t = 250$).

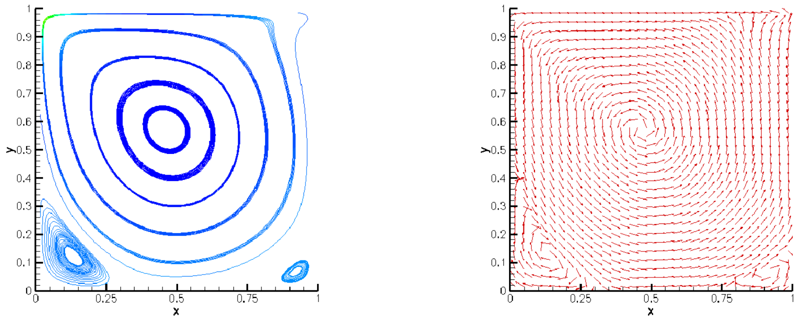

Figure 2.4: Streamlines and velocity vectors in steady state (time $t = 250$).

'Driven Cavity'-problem, since the solution has reached the steady state (see [1] or [4]). Visualizations of the result done with Tecplot are shown in figures 2.3 and 2.4.

More examples are presented in chapter 7. See section 5.2 for details on the format of the scene description file to design your own problem descrption files.

Chapter 3

Numerical Method

In this chapter we give a brief description of the numerical method underlying NaSt3DGP. For more information please refer to [1]. In several cases, different numerical methods, for example, different discretizations for time derivatives or convective terms, are possible. Details on how to control these schemes using the scene description file are given in section 5.2.

The basic mathematical model are the dimensionless time-dependent incompressible Navier-Stokes equations

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} &= \frac{\mathbf{g}}{Fr} - \nabla p + \frac{1}{Re}\Delta\mathbf{u}\ , \quad \mathbf{x}\in\Omega,\ 0\le t\le T_{fin} && (3.1)\\ \nabla\cdot\mathbf{u} &= 0\ , && (3.2) \end{aligned}$$

subject to appropriate initial and boundary conditions. Re is the dimensionless Reynolds-number which determines the ration between inertia and viscous forces in the flow. the Reynolds-number is given by

$$Re = \frac{\rho L u_\infty}{\mu}$$

where μ is the dynamic viscosity, ρ the (constant) density, L a characteristic length and u_∞ a characteristic velocity of the flow configuration. Fr is a modified Froude-number defined as

$$Fr = \frac{u_\infty^2}{L}$$

and specifies the ration of inertia to gravitational forces.

If thermal effects or the behaviour of a scalar, driven by the flow, are of interest, (3.1) and (3.2) are completed by equations for the energy (temperature) and the transport of a scalar

$$\begin{aligned} \frac{\partial T}{\partial t} + \mathbf{u}\cdot\nabla T &= \frac{1}{Re\,Pr}\Delta T && (3.3)\\ \frac{\partial C}{\partial t} + \mathbf{u}\cdot\nabla C &= \nu_C\Delta C \quad . && (3.4) \end{aligned}$$

T is the temperature and C the scalar. The dimensionless number Pr is the Prandtl number and is given by

$$Pr = \frac{\nu}{\alpha}$$

where $\nu = \mu/\rho$ is the kinematic viscosity and α is the heat diffusion coefficient. ν_C is the diffusion constant of C.

3.1 The Projection-Method

NaSt3DGP employs a Chorin-type projection method for the decoupling of momentum and continuity equations. The general procedure for projection methods is a predictor-corrector approach. In a first step a preliminary velocity field is computed utilising the momentum equations. This velocity does not satisfy the continuity equation. In a second step a poisson-type equation for the pressure is solved which is derived using the continuity equation. In the last step the preliminary velocity field is projected onto a divergence-free velocity field using the computed pressure.
Neglecting the spatial discretization (for the time being) this procedure reads

1. Given $\mathbf{u}^n$, solve

$$\frac{\tilde{\mathbf{u}} - \mathbf{u}^n}{\Delta t} = \mathbf{g} - \mathbf{u}^n \cdot \nabla \mathbf{u}^n + \nu \Delta \mathbf{u} \tag{3.5}$$

2. Solve

$$\frac{\mathbf{u}^{n+1} - \tilde{\mathbf{u}}}{\Delta t} = -\nabla p^{n+1} \tag{3.6}$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0 \; . \tag{3.7}$$

Application of the divergence operator to (3.6) leads to the Poisson equation for the pressure

$$\Delta p^{n+1} = \frac{1}{\Delta t} \nabla \cdot \tilde{\mathbf{u}} \; . \tag{3.8}$$

3.2 The Boussinesq-Approximation

For the discretization of the energy equation, a first order Boussinesq approximation is used, which takes into account buoyancy effects induced by temperature differences. Here one assumes that the density, viscosity and the Prandtl number do not depend on the temperature. Furthermore, dissipation terms like $\Delta\mathbf{u} \cdot \mathbf{u}$ are not considered for the energy equation. The temperature only affects the forcing term $\mathbf{g}$, i.e. if there is a volume force $\mathbf{g}_0$, then the resulting forcing term in (3.1) reads

$$\mathbf{g} = (1 - \beta(T - T_{ref}))\mathbf{g}_0 \; . \tag{3.9}$$

β is the volume expansion coefficient and T_{ref} is a certain reference temperature, like the mean temperature in Ω or so.

3.3 Time discretization

For the discretization of the time derivatives different schemes can be used. An explicit first order Euler scheme, a second order explicit Adams-Bashforth and a second order explicit Runge-Kutta scheme can be employed for all time derivatives, for the time discretization in equations (3.3) and (3.4) in addition a Runge-Kutta-Method of third order is available. By default the first order Euler-scheme is used, for information about how to apply another method, please refer to section 5.2.

3.4 Boundary conditions

The computational domain Ω is embedded in a 3D cuboid domain Ω_R. In the current version the following boundary conditions are implemented:

- **periodic boundary conditions**
  These conditions hold for $\mathbf{u}$, $\tilde{\mathbf{u}}$, p^{n+1}, T and C on opposite faces of Ω_R.
- **Dirichlet boundary conditions**
  On subsets $\Gamma \subset \partial\Omega$ Dirichlet conditions $\mathbf{u} = \mathbf{u}_0$ may be defined. The same b.c. hold for $\tilde{\mathbf{u}}$, i.e. $\tilde{\mathbf{u}} = \mathbf{u}_0$. These b.c. lead to homogeneous Neumann conditions for the pressure p^{n+1} on Γ. If T or C is calculated, Dirichlet b.c. for these quantities have to be specified on Γ.
- **Slip conditions**
  Here, homogeneous Dirichlet conditions hold for the normal velocity component and homogeneous Neumann conditions for the tangential velocity components. The same b.c. hold for $\tilde{\mathbf{u}}$. These b.c. lead to homogeneous Neumann conditions for the pressure p^{n+1}.
- **Outflow boundary condition I**
  On faces Γ of Ω_R homogeneous Neumann b.c. for the velocity may be defined as
  $$\frac{\partial \mathbf{u}}{\partial n} = 0 \; .$$
- **Outflow boundary condition II**
  Alternatively a boundary condition of convective type can be applied at outflow boundaries, i.e. it consists of a discretization of
  $$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = 0$$

- If no Dirichlet or periodic boundary conditions are specified, then homogeneous Neumann conditions are assumed for T and C.

3.5 Computational grid and spatial discretization

In [1] only uniform grids were used. Therefore we present the difference stencils used for the staggered grid. A 2D example is shown in figure 3.1. For convenience we denote

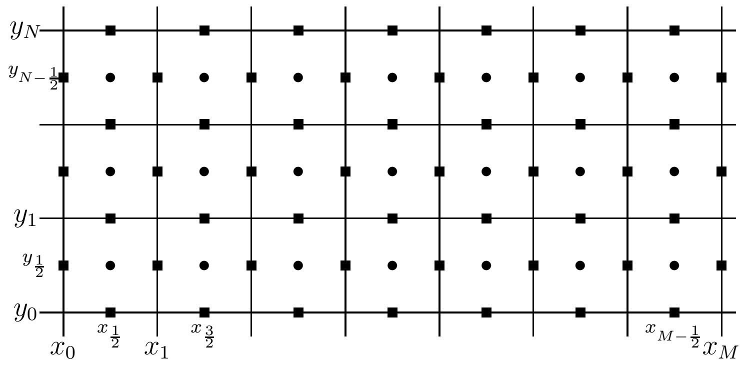

Figure 3.1: 2D staggered mesh

the coordinates of the mesh lines by $x_0, ..., x_M$, $y_0, ..., y_N$ and $z_0, ..., z_K$. These values completely define the computational grid. How to tell **navcalc** these values is explained in chapter 5. Here and in the following we denote by *cells* the rectangular subdomains $[x_i, x_{i+1}] \times [y_j, y_{j+1}] \times [z_k, z_{k+1}]$. The computational domain Ω is a union of cells.
Velocity components and pressure values are defined on the nodes:

$u_{i,j,k}$	defined on	$(x_i, y_{j+1/2}, z_{k+1/2})$
$v_{i,j,k}$	"	$(x_{i+1/2}, y_j, z_{k+1/2})$
$w_{i,j,k}$	"	$(x_{i+1/2}, y_{j+1/2}, z_k)$
$p_{i,j,k}$	"	$(x_{i+1/2}, y_{j+1/2}, z_{k+1/2})$
$T_{i,j,k}, C_{i,j,k}$	"	$(x_{i+1/2}, y_{j+1/2}, z_{k+1/2})$

, where $i, j, k \in \mathbb{Z}$.

The following stencils are used for the spatial discretization. We use the notation

$$\Delta x_i = x_i - x_{i-1} \quad \text{and} \quad \Delta x_{i+1/2} = (\Delta x_i + \Delta x_{i+1})/2 \quad .$$

The values Δy_j, $\Delta y_{j+1/2}$, Δz_k, $\Delta z_{k+1/2}$ are defined analogously. To preserve the second order accuracy of the stencils a smooth distribution of the grid spaces Δx_i,.. is required. To obtain such smooth grids use the `GridGen` utility.

Diffusive terms:

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_{i,j,k} = \frac{1}{\Delta x_{i+1/2}}\left(\frac{u_{i+1,j,k}-u_{i,j,k}}{\Delta x_i} - \frac{u_{i,j,k}-u_{i-1,j,k}}{\Delta x_{i-1}}\right)$$
$$\left[\frac{\partial^2 u}{\partial y^2}\right]_{i,j,k} = \frac{1}{\Delta y_j}\left(\frac{u_{i,j+1,k}-u_{i,j,k}}{\Delta y_{j+1/2}} - \frac{u_{i,j,k}-u_{i,j-1,k}}{\Delta y_{j-1/2}}\right)$$

The other diffusive terms are discretized in a similar fashion. Stencils similar to the one for $\frac{\partial^2 u}{\partial y^2}$ are used for $\frac{\partial^2 \Theta}{\partial x^2}$, $\frac{\partial^2 \Theta}{\partial y^2}$ and $\frac{\partial^2 \Theta}{\partial z^2}$, where Θ is either T or C.

Convective terms:
Five different discretizations of the convective terms are possible:

1. Donor-Cell (hybrid-scheme) (1st/2nd order)
2. Quadratic upwind interpolation for convective kinematics (QUICK) (2nd-Order)
3. Hybrid-Linear Parabolic Arppoximation (HLPA) (2nd-Order)
4. Sharp and Monotonic Algorithm for Realistic Transport (SMART) (2nd-Order)
5. Variable-Order Non-Oscillatory Scheme (VONOS) (2nd/3rd-Order) (default)

To select one of these schemes, you have to set the appropriate variable in the scene description file. How to do this is explained in section 5.2. By default the VONOS-scheme is used. In the following the Donor-Cell scheme is briefly described. More details about the other schemes can be found in [3].

Second order convective terms:

$$\left[\frac{\partial u^2}{\partial x}\right]_{i,j,k} = \frac{(u_{i+1,j,k}+u_{i,j,k})^2 - (u_{i,j,k}+u_{i-1,j,k})^2}{4\Delta x_{i+1/2}}$$
$$\left[\frac{\partial vu}{\partial y}\right]_{i,j,k} = \frac{v_{i+1/2,j,k}u_{i,j+1/2,k} - v_{i+1/2,j-1,k}u_{i,j-1/2,k}}{\Delta y_i}$$

Stencils similar to the one for $\frac{\partial vu}{\partial y}$ are used for the discretization of the convective terms in (3.3) and (3.4), e.g. we have

$$\left[\frac{\partial vT}{\partial y}\right]_{i,j,k} = \frac{v_{i+1/2,j,k}T_{i,j+1/2,k} - v_{i+1/2,j-1,k}T_{i,j-1/2,k}}{\Delta y_i} \quad .$$

The unknown values, e.g. $v_{i+1/2,j,k}$ are computed by linear interpolation.

$$
\begin{aligned}
u_{i,j+1/2,k} &= \frac{\Delta y_j u_{i,j+1,k} + \Delta y_{j+1} u_{i,j,k}}{\Delta y_j + \Delta y_{j+1}} \\
v_{i+1/2,j,k} &= \frac{\Delta x_i v_{i+1,j,k} + \Delta x_{i+1} v_{i,j,k}}{\Delta x_i + \Delta x_{i+1}}
\end{aligned}
$$

First order upwind:

$$
\begin{aligned}
\left[\frac{\partial u^2}{\partial x}\right]_{i,j,k} &= \frac{k_R u_R - k_L u_L}{\Delta x_{i+1/2}} \quad \text{, where } k_R = (u_{i+1,j,k} + u_{i,j,k})/2 \quad , \quad k_L = (u_{i+1,j,k} + u_{i,j,k})/2 \\
& \text{and } u_R = \begin{cases} u_{i,j,k} & : \; k_R > 0 \\ u_{i+1,j,k} & : \; k_R \leq 0 \end{cases} \quad ; \quad u_L = \begin{cases} u_{i-1,j,k} & : \; k_L > 0 \\ u_{i,j,k} & : \; k_L \leq 0 \end{cases} \\
\left[\frac{\partial vu}{\partial y}\right]_{i,j,k} &= \frac{k_R u_R - k_L u_L}{\Delta y_j} \quad \text{, where } k_R = v_{i+1/2,j,k} \quad , \quad k_L = v_{i-1/2,j,k} \\
& \text{and } u_R = \begin{cases} u_{i,j,k} & : \; k_R > 0 \\ u_{i,j+1,k} & : \; k_R \leq 0 \end{cases} \quad ; \quad u_L = \begin{cases} u_{i,j-1,k} & : \; k_L > 0 \\ u_{i,j,k} & : \; k_L \leq 0 \end{cases}
\end{aligned}
$$

Stencils similar to the one for $\frac{\partial vu}{\partial y}$ are used for the upwind discretization of the convective terms in (3.3) and (3.4). First and second order terms can be blended using a parameter α by, e.g.

$$
\frac{\partial u^2}{\partial x} = \alpha(\text{first order}) + (1-\alpha)(\text{second order}) \quad .
$$

The blending parameter α is user definable (see chapter 5) and may be chosen different for the equations of momentum and energy or transport of a scalar.

Laplacian for pressure:
We employ a conservative discretization which is simply the nested application of the centered difference for the pressure gradient and the centered difference for the natural discretization of the divergence, e.g.

$$
\left[\frac{\partial^2 p}{\partial x^2}\right]_{i,j,k} = \frac{1}{\Delta x_i}\left(\frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x_{i+1/2}} - \frac{p_{i,j,k} - p_{i-1,j,k}}{\Delta x_{i-1/2}}\right) \quad .
$$

Poisson solvers:

For the solution of the linear equation arising from discretization of the pressure poisson equation, the following numerical methods are implemented:

1. Successive Overrelaxation (SOR)
2. Symmetric SOR (forward/backward)
3. Red-Black scheme
4. 8-Color SOR
5. 8-Color Symmetric SOR (fw/bw)
6. BiCGStab

To select a method, select the corresponding option in the scene description file(see section 5.2 on how to do this). By default, the Poisson-equation is solved using the BiCGStab-method.

3.6 Discretization of boundary conditions

As described in [1], Dirichlet conditions for the velocity are implemented by setting the nodal value of the normal velocity component or by linear interpolation for the tangential velocity components. The homogeneous boundary Neumann conditions for the pressure are discretized by, e.g.

$$p_{0,j,k} = -p_{1,j,k} \ .$$

In case of convex edges some special treatment is required. Consider the situation shown in Figure 3.2. If Γ_1 denotes the face between the cells $_{i,j}$ and $_{i,j+1}$ and Γ_2 the face between

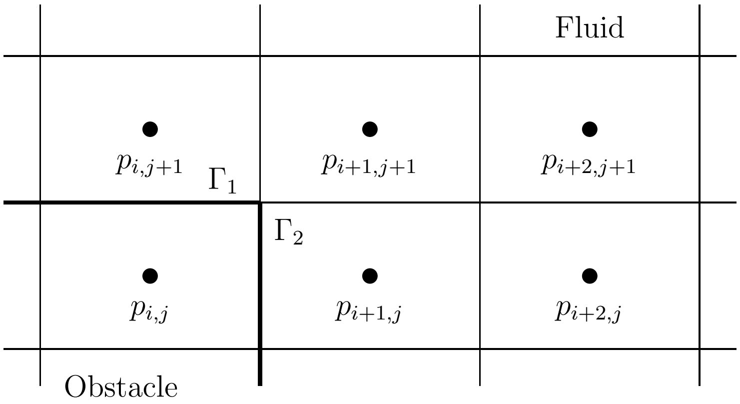

Figure 3.2: convex corner

the cells ${}_{i,j}$ and ${}_{i+1,j}$, then the homogeneous Neumann conditions are discretized by

$$p_{i,j} = -\frac{|\Gamma_1|p_{i,j+1} + |\Gamma_2|p_{i+1,j}}{|\Gamma_1| + |\Gamma_2|} \quad .$$

Chapter 4

Navcalc and Navcalcmpi

The binary **navcalc** (or **navcalcmpi** for parallel computation) is used to run the simulation. At the end of a calculation the computed values of $\mathbf{u}, p, \{T, C\}$ are written in the data file. The original binary data file is overwritten and can now be used for a restart of a longer calculation or for analysing and postprocessing the computed data. By setting a corresponding option in the scene description file you also have the possibility to save computed data in regular intervals. Details on how to specify how often and where the data is to be stored can be found in section 5.2.

4.1 Command line arguments

The following parameters can be used on the command-line of **navcalc**:

`-h`	Prints the list of parameters and a short description
`-b` ⟨*DataFile*⟩	binary data file. If this option is not given, **navcalc** or **navcalcmpi** try to read the data from a file named `default.bin` in the current directory.
`-t` ⟨n⟩	Sets maximum number of seconds to run. With this option the maximum computing time for a specific job can be limited. Default is no restriction.
`-p`⟨n⟩	Set info level to $n = 0, 1, 2, 3$. Depending on the `-p` argument the amount of additional information that is printed to `stdout` can be adjusted.

4.2 Parallelization issues

For the parallelization the domain Ω_R is divided into several rectangular subdomains $\Omega_1, \ldots, \Omega_\mu$. A simple 2D example is shown in figure 4.1. Each process holds some ghost

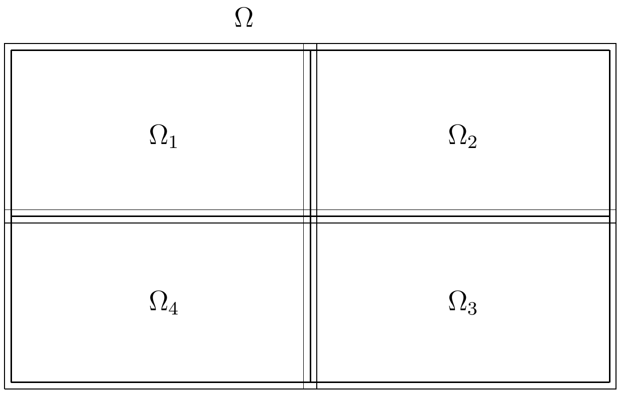

Figure 4.1: 2D parallelization example.

cells which overlap inner cells of the adjacent process. Values are copied from these to the ghost cells when necessary. To minimize communication, the program divides Ω_R in a way that minimizes the area of the touching faces and equilibrates the number of cells in the different subdomains.
The data required for parallelization is kept in the class `ParParams` which can be found in `parallel.hpp`. The most important variables are described in the following table:

variable	description
`biug`, `biog`, `bjug`, `bjog`, `bkug`, `bkog`	This defines the computational domain for this process (*excluding* the necessary ghost cells) as $\{\texttt{biug},\dots,\texttt{biog}\}\times\{\texttt{bjug},\dots,\texttt{bjog}\}\times\{\texttt{bkug},\dots,\texttt{bkog}\}$. For your convenience, read access is possible using the macros `iug`, `iog`, etc. For example, $\{1,16\}\times\{1,10\}\times\{1,10\}$ will be divided into $\{1,8\}\times\{1,10\}\times\{1,10\}$ and $\{9,16\}\times\{1,10\}\times\{1,10\}$.
`me`	Process ID of this task.
`tn`, `ts`, `tw`, `te`, `tb`, `tt`	IDs of the neighbour processes in each direction (see figure 5.2). The value -1 means that there is no neighbour process.

4.3 Implementational details

In this section a quick overview of the general structure and the most important aspects of the implementation are given. For a detailed documentation of the code please refer to the reference manual.

The main loop of the program reads:

```
1    SetObstacleCond(0,0);
2    Par->CommMatrix(U, 1, 0, 0, UCOMM);
3    Par->CommMatrix(V, 0, 1, 0, VCOMM);
4    Par->CommMatrix(W, 0, 0, 1, WCOMM);
5    S.delt=TimeStep();
6    do {
7        CompFGH();
8        CompRHS();
9        res=Poisson();
10       AdapUVW();
11       SetObstacleCond(0,0);
12       Par->CommMatrix(U, 1, 0, 0, UCOMM);
13       Par->CommMatrix(V, 0, 1, 0, VCOMM);
14       Par->CommMatrix(W, 0, 0, 1, WCOMM);
15       S.delt=TimeStep();
16       if (S.CompTemp==TRUE) {
17           CompTG(T,S.alphatg,S.nu/S.prandtl);
18           Par->CommMatrix(T, 0, 0, 0, TCOMM);
19       }
20       if (S.CompChem==TRUE) {
21           int n;
22           for (n=0; n<S.nchem; n++) {
23               CompTG(CH[n],S.alphatg,S.chemc[n]);
24               Par->CommMatrix(CH[n], 0, 0, 0, CHCOMM+n);
25           }
26       }
27       S.t+=S.delt;
28       n++;
29   } while (S.Tfin < 0);
```

In lines 1 to 5, a first initialization of boundary values is set and the initial size Δt of the time steps is computed. In line 7, $\tilde{\mathbf{u}}$ is computed, in the next line $\frac{1}{\Delta t}\nabla \cdot \tilde{\mathbf{u}}$, see (3). In line 9, equation (3.8) is solved and the new velocity values are computed in line 10 according to (3.6). In lines 11 to 14, boundary conditions are set and velocities are exchanged. From the new velocities, the size of the next time step is calculated in line 15 and in line 16 to 26 optional transport equations are solved. The following list shows where the most important classes can be found.

class	file
`Scene`	`typen.hpp`
`ParParams`	`parallel.hpp`
`Navier`	`navier.hpp`
`NavierCalc`	`navier.hpp`
`Matrix<T>`	`matrix.hpp`
`List<T>`	`list.hpp`
list objects	`typen.hpp`

Data structures

The physical parameters, information about the grid, precalculated discretizations and some other stuff is kept in the class `Scene`. The most important variables are listened below.

variable	description
`gridp[3]`	This defines the number of cells in every direction for the whole domain Ω_R, the inner cells are numbered from 1 to `gridp[`i`]`, $1 \leq i \leq 3$. The cells which are computed by the actual process are defined in the class `ParParams`, which is described in (4.2).
`d[3][]`	This variable contains the spaces between the grid lines in every direction (see figure 3.1). For convenience: access is possible using the macros `DX[`i`]`, `DY[`j`]` and `DZ[`k`]`.
`dm[3][]`	This variable contains the spaces between the middles of the cells in every direction (marked by the circles in figure 3.1). For convenience: access is possible using the macros `DXM[`i`]`, `DYM[`j`]` and `DZM[`k`]`.
`kabs[3][]`	This variable contains the absolute positions of the grid lines.
`ddstar[3][3][]`, `ddPstar[3][3][]`, `ddSstar[3][3][]`	These variables contain different discretizations for the second derivative, where the first index selects the direction and the second index selects the appropriate weight (left - 0, middle - 1, right - 2). The discretizations are described in (3.5).
`periodbound`	This is a bit field indicating if periodic boundary conditions are set in the appropriate direction (bit 0 - x-direction, bit 1 - y-direction, bit 2 - z-direction).

The 3D matrices, for example velocity and pressure data, are kept in the class `Navier`. There are two derived classes, `NavierSetup` and `NavierCalc`. The dimensions of these matrices are defined in the following table.

matrix	dimension
flag, P, F, G, H, RHS, T, CH[i]	$\{\texttt{biug}-1,\texttt{biog}+1\}\times\{\texttt{bjug}-1,\texttt{bjog}+1\}\times\{\texttt{bkug}-1,\texttt{bkog}+1\}$
U	$\{\texttt{biug}-1-\mu_x,\texttt{biog}+1\}\times\{\texttt{bjug}-1,\texttt{bjog}+1\}\times\{\texttt{bkug}-1,\texttt{bkog}+1\}$
V	$\{\texttt{biug}-1,\texttt{biog}+1\}\times\{\texttt{bjug}-1-\mu_y,\texttt{bjog}+1\}\times\{\texttt{bkug}-1,\texttt{bkog}+1\}$
W	$\{\texttt{biug}-1,\texttt{biog}+1\}\times\{\texttt{bjug}-1,\texttt{bjog}+1\}\times\{\texttt{bkug}-1-\mu_z,\texttt{bkog}+1\}$

The variable μ_x is set to 1 if periodic boundary conditions are set in x-direction or if the current process has a neighbour in `south`-direction and 0 otherwise. The variables μ_y and μ_z are set in a similar way.
The cells where boundary values need to be set (for one specific parallel process) are stored in lists which are defined in `NavierCalc`.

list	contains
`InflowList`	cells where Inflow b.c. are set (parameters are stored in list)
`SlipList`	cells where Slip b.c. (Neumann) are set
`NoslipList`	cells where Noslip b.c. (Dirichlet) are set
`PdpdnList[`*color*`][`*face*`]`	cells where prescribed boundary conditions (b.c.) lead to homogeneous Neumann b.c. for the pressure and which have the color *color*. These cells have exactly one fluid cell neighbouring at face *face*.
`PSpecdpdnList[`*color*`]`	cells where prescribed boundary conditions (b.c.) lead to homogeneous Neumann b.c. for the pressure and which have the color *color*. These cells have more than one neighbouring fluid cell. This requires some more involved boundary treatment.
`PInOut2List[`*color*`][`*face*`]`	cells with the prescribed outflow condition 2, it is assumed that no cells with more than one neighbouring fluid cell exist. Some special boundary treatment is required.
`PInOut1List[`*color*`][`*face*`]`	same as `PInOut2List` for outflow condition 1.

Chapter 5

Navsetup

The main components of the NaSt3DGP package are the two parts **navsetup** and **navcalc**. As mentioned before, **navsetup** produces from a so-called scene description file (`*.nav`) the so-called data-file (`*.bin`). The scene description file must contain all information to describe the flow configuration. This includes the numbers of grid points, the mesh, the geometry of obstacles, place and type of boundary conditions, etc. From this data a compact, internal representation of the flow configuration is computed and written in the data-file. This data-file now contains initial values of the pressure and velocity (p, $\mathbf{u}$), a flag field which describes the geometry and some general information such as boundary conditions or physical parameters of the fluid.

The data-file is read by **navcalc** to run the simulation. After a prescribed number of time steps and before finishing, **navcalc** writes the actual pressure, velocity, etc. values in this file and thus overwrites the initial data. This new data-file can be used for a restart or for the analysis of the simulated flow. Since the format of the data-file is rather complicated **navsetup** provides the possibility to read a data-file and to write the pressure, velocity components, etc. in separate files which are easier to handle. A simple *MatLab V* tool which reads such files and stores the entries in an 3D array is `tools/matlab/ReadNast.m`

5.1 Command line arguments

The following is a list of command-line arguments for **navsetup**:

`-h`	Prints the list of parameters and a short description and prints the version number of NaSt3DGP.
`-s` ⟨*SceneFile*⟩	Reads the scene description file ⟨*SceneFile*⟩ and generates a data-file
`-b` ⟨*DataFile*⟩	The name of the data-file, the default name is `default.bin`
`-l` ⟨*DataFile*⟩	load field variables(velocity, pressure etc.) from an existing datafile. This is useful if you want to run a calculation with previsously computed values $p,\ \mathbf{u}\ \{,\ T,\ c,\ ..\}$ but want to change some simulation parameters. You can create a binary file for the new calculation with the following command: `navsetup -s` ⟨*SceneFileNew*⟩ `-b` ⟨*DataFileNew*⟩ `-l` ⟨*DataFile*⟩ The data will be interpolated if the number of grid points changes.
`-L` ⟨*DataFiles*⟩	Same as `-l`, but reads several data files ⟨*DataFiles*⟩`.*`.
`-A` ⟨*TextFiles*⟩	Same as `-L`, but reads several text files (written by −`a`) ⟨*TextFiles*⟩`.txt.*`.
`-o` ⟨*OutFile*⟩	This option sets the base of the filename to ⟨*OutFile*⟩ for the options which write files. If this option is not used, ⟨*OutFile*⟩ is set to 'default'.
`-TC` ⟨*DataFile*⟩	Reads the data $p,\ \mathbf{u}\ \{,\ T,\ c,\ ..\}$ from the data-file and writes to ⟨*Outfile*⟩`.dat`. The file format is ASCII Tecplot. The data written is located at the cell centers.
`-T` ⟨*DataFile*⟩	same as $-TC$ but writes each field to a separate file named ⟨*Outfile*⟩`.`⟨*field*⟩`.dat`, e.g. ⟨*Outfile*⟩`.u.dat` .
`-g` ⟨*DataFile*⟩	Reads the data $p,\ \mathbf{u}\ \{,\ T,\ c,\ ..\}$ from the data-file and writes each of it in ⟨*Outfile*⟩`.u`, ⟨*Outfile*⟩`.v` etc. These files can be read in Matlab with the script `tools/matlab/ReadNast.m`.
`-a` ⟨*DataFile*⟩	Same as $-g$, but use ASCII-format(text files) which are named ⟨*OutFile*⟩`.txt.*`.
`-G` ⟨*DataFile*⟩	Reads the data $p,\ \mathbf{u}\ \{,\ T,\ c,\ ..\}$ from the data-file and writes each of it in ⟨*Outfile*⟩`.vtk.u`, ⟨*Outfile*⟩`.vtk.v` etc. The files are written in the VTK[1] rectilinear grid format.
`-GS`⟨*DataFile*⟩	same as $-G$ but uses the VTK Structured Points format and the files are named ⟨*Outfile*⟩`.vtksp.*` .
`-f` *uvwptcV*	create a VTK file including the velocity-field V or multiple scalars $u, v, w, p, t, c0 \dots cN$. This option has to be used together with $-G$, e.g. use the command `navsetup -f` Vp `-G` ⟨*DataFile*⟩ `-o` ⟨*Outfile*⟩ to write the the velocity as a vectorfield and the pressure p to a file named ⟨*Outfile*⟩`.vtk.mix` or use `navsetup -f` puv `-G` ⟨*DataFile*⟩ `-o` ⟨*Outfile*⟩ to write the pressure p and the first two velocity components u and v.

5.2 Scene description file

The scene description file (`*.nav`-file) is a text-file which is used to specify all necessary parameters for the calculation. It contains information about the geometry, the mesh, the fluids and is also used to specify initial and boundary conditions. In the following sections the syntax of the scene description file is explained.

5.2.1 Blocks

The scene description file consists of blocks of the form

$$\begin{array}{l}\langle \textit{blocktype} \rangle \; \{ \\ \quad \quad \ldots \\ \} \end{array}$$

where ⟨*blocktype*⟩ is one of the keywords

`dimension, parameter, box, sphere, cylinder, halfspace, union, intersection, poly`

which are explained below. The blocks are processed in the order as they appear in the file.

5.2.2 Comments

Comments begin with a double slash followed by a space and end with an end-line character, for example

```
// this is a first comment
dimension { // this is a comment
    ...
}
```

5.2.3 `dimension`

The `dimension` block contains information about the grid and the size of the domain Ω. The following keywords are allowed:

`length` <x,y,z> defines the domain Ω as

$$\Omega = [0, x] \times [0, y] \times [0, z] \subset \mathbb{R}^3.$$

If not specified, the default domain is $\Omega = [0, 1]^3$.

`resolution` <M,N,K> defines the number of cells in each direction, numbered from 1 to M, 1 to N and from 1 to K. The program will add "ghost cells" in each direction, so the exact number of cells is $N + 2$, respectively numbered from 0 to $N + 1$. If not specified, the default number of cells will be $M = N = K = 15$. This parameter has to be specified *before* one of the following parameters `x`, `y` or `z`.

`x` k_0 $x_0 \ldots k_M$ x_M defines a non-equidistant grid, where the x_l values define the positions of the grid lines and the k_l values are optional numbers of the grid lines. These values are ignored by the parser and only serve for compatibility with `gnuplot`'s data format. The program will compute the width of every cell from x_l. Thus the first grid line does not need to be positioned at 0. Furthermore, the width of the "ghost cells" will be the same as the width of the adjacent inner cell. If not specified, the grid will be equidistant and the widths are calculated using resolution and length.

`y` k_0 $y_0 \ldots k_N$ y_N See `x` k_0 $x_0 \ldots k_M$ x_M.

`z` k_0 $z_0 \ldots k_K$ z_K See `x` k_0 $x_0 \ldots k_M$ x_M.

Example:

```
dimension {
   resolution <5, 4, 1>
   length <3.0, 2.0, 0.5>
 x 0  0.0
   1  0.5
   2  1.1
   3  1.9
   4  2.5
   5  3.0
}
```

generates the grid shown in figure 5.1.

5.2.4 `parameter`

The `parameter`-block contains several parameters which control the computation and several constants describing the flow and physical quantities. If a parameter is not given, a

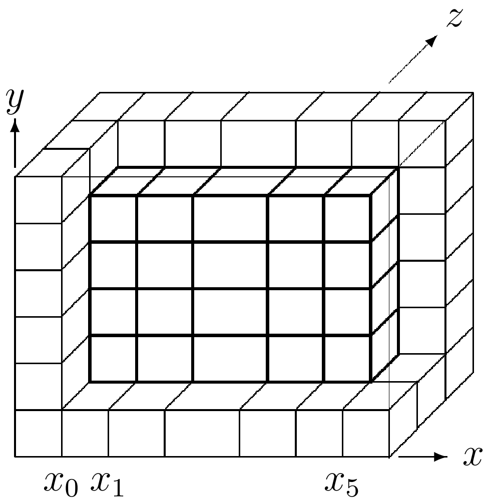

Figure 5.1: Ghost cells are bounded by thin lines and inner cells are bounded by thick lines.

default value will be used. A complete list of all possible parameters together with an explanation and the default value is given in table 5.1.
All physical quantities, like Dirichlet values for the velocity, temperature or the viscosity are assumed to be given in standard ISO units (m, s, kg, K and derived units), unless the `dimensionless`-flag is set(see below).

Name	Type	Default	Description
Flow parameters			
`Tfin`	double [s]	1.0	defines the (physical) timespan of the simulation
`reynolds`	double	10.0	is the dimensionless Reynolds-number which describes the ratio between inertia and viscous forces in the flow

Name	Type	Default	Description
`nuC`	int	0	the paramter `nuC` sets the number of species to be transported with the flow. For each species, a diffusion constant has to be specified in a comma-separated list after nuC, i.e. the line has to look like `nuC` $n_C, \nu_1, \ldots, \nu_{n_C}$ where n_C is an integer specifying the number of species and ν_i is a double value which stands for the diffusion parameter of the i-th species. You have to make sure that you also specify initial conditions using the keyword `cheminit`. Details on how to specify initial and boundary conditions are given in section 5.2.5
`gx`	double [m/s^2]	0.0	x-component of external volume force $\mathbf{g}$ in (3.1) (or $\mathbf{g}_0$ in (3.9) if the temperature is calculated). In the latter case $\mathbf{g}$ is computed from $\mathbf{g}_0$ by means of (3.9)
`gy`	double [m/s^2]	0.0	Same as `gx` for the y-component
`gz`	double [m/s^2]	0.0	Same as `gx` for the z-component
`froude`	double	1.0	the Froude-number is a dimensionless number describing the ratio between inertial and gravitational forces
`beta`	double [1/K]	1e-4	volume expansion coefficient in equation (3.9)
`TempRef`	double [K]	273.0	reference temperature in (3.9)
`prandtl`	double	1.0	the Prandtl-number is a dimensionless number describing the ratio between momentum and heat transfer in the fluid
Timestep control			
`deltmax`	double [s]	1.0	upper bound for Δt

Name	Type	Default	Description
`tfconv`	double	0.1	security factor for the timestep restriction arising from the convective terms. In every step, Δt will be set less or equal than $$\Delta t \leq \texttt{tfconv} \cdot \min_{c \in \mathcal{F}} \left\{ \frac{\Delta x_c}{u_c}, \frac{\Delta y_c}{v_c}, \frac{\Delta z_c}{w_c} \right\}$$ where $\mathcal{F}$ denotes the set of cells $$c = [x, x+\Delta x_c] \times [y, y+\Delta y_c] \times [z, z+\Delta z_c]$$ in Ω
`tfdiff`	double	0.2	security factor for the timestep-restriction arising from the diffusive terms. In every step, Δt will be set less or equal than $$\Delta t \leq \texttt{tfdiff} \min_{\nu \in \mathcal{N}, c \in \mathcal{F}} \left[\nu \left(\frac{1}{{\Delta x_c}^2} + \frac{1}{{\Delta y_c}^2} + \frac{1}{{\Delta z_c}^2} \right) \right]^{-1}$$ and $\mathcal{N} = \{\nu\} \cup \underbrace{\{1/(\nu \cdot \texttt{prandtl})\}}_{\text{only if temperature is computed}} \cup \underbrace{\{\nu_1, \ldots, \nu_{nC}\}}_{\text{only if scalars are computed}}$.
Data output			
`prstep`	int	20	defines when to write the computed values to the binary file. Each `prstep`-th timestep the current solution is written to the binary file(the file is overwritten)
`TimePrintStep`	string	-	defines an interval (in physical time) after which the solution should be written to files in a directory specified by `TargetDirectory`. If `TimePrintStep` is not specified, no output will be generated.
`TargetDirectory`	string	-	specifies the directory(absolut pathname) where the files generated by `TimePrintStep` should be stored

Name	Type	Default	Description
Parameters controlling numerical methods			
`TimeDis`	string	EU1	defines the time discretization to be used. Possible values are EU1 for first order explicit Euler-Method, AB2 for second order explicit Adams-Bashforth-Method, RK2 for second order Runge-Kutta-method and RK3 for third order Runge-Kutta-method. Remark: The Runge-Kutta-method of third order is only available for the time derivatives in equations (3.3) and (3.4), setting `TimeDis` to RK3 results in application of the Runge-Kutta-method of second order to the other time derivatives
`ConvectiveTerms`	string	VONOS	defines the discretization scheme to be used for the convective terms. Possible values are DC (Donor-Cell, 1st/2nd order), HLPA (Hybrid Linear-Parabolic Approximation, 1st/2nd order), QUICK (Quadratic Upwind Interpolation for Convective Kinematics, 2nd order), SMART (Sharp And Monotonic Algorithm for Realistic Transport, 2nd order) and VONOS (Variable-Order Non-Oscillatory Scheme, 2nd order)
`PoissonSolver`	string	BiCGStab	set the method for solution of the linear system arising from discretization of the pressure poisson equation. Possible values are SOR, SSOR, RedBlack, 8ColorSOR, 8ColorSSOR and BiCGStab (preconditioned with Jacobi-Method)
`alpha`	double	1.0	defines the blending parameter α in the convex combination of the central difference/upwind discretization of the convective terms of F, G, H. `alpha` $= 1$ means pure upwind and `alpha` $= 0$ results in pure central difference discretization

Name	Type	Default	Description
`alphaTC`	double	1.0	same as `alpha`, but for convective terms in the transport equation used for computation of temperature and scalars
Parameters for the linear solver			
`itermax`	int	100	defines the maximal number of iterations in the linear solver (BiCGStab,SOR, SSOR etc.)
`eps`	double	0.001	defines the stopping criterion for the iterations in the linear solver. The parameter `eps` is the upper bound for the residual of the poisson equation, i.e. the iterations are stopped if $$\sqrt{\frac{1}{\sharp\mathcal{F}}\sum_{c\in\mathcal{F}}\left(\Delta_h p_c - \frac{1}{\Delta t}(\nabla_h\cdot\tilde{\mathbf{u}})_c\right)^2} \leq \texttt{eps}\,.$$ Here, $\mathcal{F}$ is the set of all fluid cells in Ω and $\sharp\mathcal{F}$ is the cardinality of $\mathcal{F}$. Δ_h and ∇_h denote the discrete Laplacian and gradient operator.
`omega`	double	1.7	sets the relaxation parameter for the SOR-type solvers
Boundary conditions			
`periodboundx`			sets periodic boundary conditions in direction of x-coordinate
`periodboundy`			same as `periodboundx` only for the y-coordinate-direction
`periodboundz`			same as `periodboundx` only for the z-coordinate-direction

Table 5.1: List of entries in the `parameter`-block

5.2.5 Objects

Objects are used to describe the geometry of the simulation domain and to specify initial and boundary conditions. There are four basic object types, `box`, `sphere`, `cylinder` and `halfspace`. Additional geometries and obstacles can be created with the CSG operations `union`, `difference` and `intersection`. There exists one restriction for obstacle cells, namely two opposite faces are not allowed to touch fluid cells.
As already mentioned in section 5.2.1, objects are just another type of blocks. The commands given in table 5.2 can be used inside any `object`-block.

Name	Description
`coords` <X_1,Y_1,Z_1>,<X_2,Y_2,Z_2>	sets the size of `box`, `sphere` or `cylinder` objects to $[X_1, X_2] \times [Y_1, Y_2] \times [Z_1, Z_2]$. The default unit used is one cell. This can be changed by the command `absolute`
`absolute`	allows use of absolute coordinates given by `length`
`fluid`	marks the concerning cells as fluid cells. This command makes it possible for example to cut quite easily holes into other objects. `fluid` can not be defined on ghost cells (e.g. $i \in \{0, M+1\}$) with non-periodic boundary conditions in the specific direction
`init` <u_0,v_0,w_0>,p_0	initializes the concerning cells with velocity values (u_0, v_0, w_0) and pressure p_0. The default initialization is $u_0 = v_0 = w_0 = p_0 = 0$. These values are not concerned as Dirichlet boundary condition
`noslip`	defines the Dirichlet boundary condition $\mathbf{u} = 0$ (see 3.4) for the concerning cells. This is the default value. Homogeneous Neumann boundary conditions are set for temperature and scalars. This is the difference to `inflow <0,0,0>` (see below) where also $\mathbf{u} = 0$ is set
`slip`	defines Slip boundary condition (see 3.4) for the concerning cells
`inout` n,v_Γ	defines Outflow boundary condition of type n, $1 \leq n \leq 2$, where the additional parameter v_Γ is ignored if $n = 1$. See section 3.4 for an explanation of the different types. If $n = 2$ and $v_\Gamma = 1$ the velocity field at the outflow boundaries is corrected to enforce the compatibility condition (set v_Γ to 0 if you do not want this).
`inflow` <u_0,v_0,w_0>	defines Dirichlet boundary condition $\mathbf{u} = (u_0, v_0, w_0)$ (see 3.4) for the concerning cells. When temperature or scalars are computed, Dirichlet boundary conditions for these quantities are set. You must specify the particular values by `temperature` and/or `cheminit`

Name	Description
`temperature` $T_{\texttt{init}}$	defines Dirichlet boundary conditions on boundary cells for the temperature T_{init} or initializes fluid cells with temperature T_{init}
`cheminit` $c_1, \ldots, c_{n_C}$	defines Dirichlet boundary conditions for scalars $c_1, \ldots, c_{n_C}$ on boundary cells or initializes fluid cells with these values

Table 5.2: valid commands inside an `object`-block

Now we describe the differences between the object-types.

box: The command `box` defines the box $[X_1, X_2] \times [Y_1, Y_2] \times [Z_1, Z_2]$ except one of the keywords `north`, `south`, `west`, `east`, `top` or `bottom` is given. In this case, the box is the appropriate "wall" (see figure 5.2):

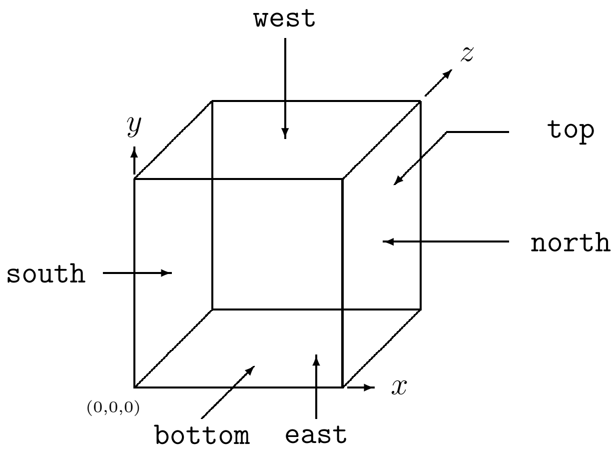

Figure 5.2: wall naming scheme

`north`	$\{M+1\} \times \{0,..,N+1\} \times \{0,..,K+1\}$
`south`	$\{0\} \times \{0,..,N+1\} \times \{0,..,K+1\}$
`west`	$\{0,..,M+1\} \times \{N+1\} \times \{0,..,K+1\}$
`east`	$\{0,..,M+1\} \times \{0\} \times \{0,..,K+1\}$
`top`	$\{0,..,M+1\} \times \{0,..,N+1\} \times \{K+1\}$
`bottom`	$\{0,..,M+1\} \times \{0,..,N+1\} \times \{0\}$

sphere: The command `sphere` defines the ellipsoid which touches all faces of the box $[X_1, X_2] \times [Y_1, Y_2] \times [Z_1, Z_2]$ defined by `coords`. The main axes are parallel to the coordinate axes.

cylinder: In the same way, `cylinder` defines a cylinder whose rotation axis is defined by one of the commands `x`, `y` or `z`.

halfspace: The command `halfspace` defines the set of cells

$$\{p \in \mathcal{C} \mid ax_p + by_p + cz_p + d \geq 0\},$$

where a, b, c, d are given by the command

`hesse <`a`,`b`,`c`>,`d

and (x_p, y_p, z_p) is the middle of the corresponding cell p.

poly: The command `poly` defines a polytope bounded by polygons. The points are specified using the command

`points` n`,<`a_0`,`b_0`,`c_0`>,...,<`a_{n-1}`,`b_{n-1}`,`c_{n-1}`>` .

The first point indexed with 0. After defining points, the polygons have to be be specified using the command

`vertices` n`,`p_1`,`p_2`,...,`p_n

where n is the total number of indices (including control indices -1). One polygon is defined by enumerating the points in clockwise or counterclockwise order, where the first point has to be enumerated once again as last point. Then this series is finished with index -1. The following example defines a cube:

```
poly {
  points 8,<0.0,0.0,0.0>,
           <1.0,0.0,0.0>,
           <1.0,1.0,0.0>,
           <0.0,1.0,0.0>,
           <0.0,0.0,1.0>,
           <1.0,0.0,1.0>,
           <1.0,1.0,1.0>,
           <0.0,1.0,1.0>
  vertices 36,0,1,2,3,0,-1,
           1,5,6,2,1,-1,
           4,5,6,7,4,-1,
           0,4,7,3,0,-1,
           3,2,6,7,3,-1,
           0,1,5,4,0,-1
}
```

CSG-operations The blocks `union`, `difference` and `intersection` define a CSG-operation on two of the objects described above. Such an operation is defined as a block in which two `object`-blocks are defined, for example

```
union {
  box {
    ...
  }
  box {
    ...
  }
}
```

The cells of the first given object are called $\mathcal{O}_1$ and the cells of the second are called $\mathcal{O}_2$. Then, the CSG-commands define the following cells:

`union`	$\mathcal{O}_1 \cup \mathcal{O}_2$
`difference`	$\mathcal{O}_1 \setminus \mathcal{O}_2$
`intersection`	$\mathcal{O}_1 \cap \mathcal{O}_2$

Chapter 6

Utilities

In this chapter we give an overview over various useful tools contained in the distribution of NaSt3DGP.

6.1 vrml2nav

The `vrml2nav` program is a utility to generate mesh and geometry information in the scene description file format from a scene given in the VRML 1.0 format[1]. After installation of the NaSt3DGP-package it can be found in the `bin` directory of the installation path(where also `navsetup` and `navcalc` reside).
The usage is very simple, just type

```
vrml2nav ⟨vrmlfile.wrl⟩
```

where ⟨`vrmlfile.wrl`⟩ is the file containing the description of the scene in VRML 1.0 format. In addition, `vrml2nav` reads the textfile `vrml2nav.cfg`. In this file you just need one line to specify where the output (the scene description in nav-format) should be written. The line begins with the keyword `outputnav` followed by a filename, i.e.

```
outputnav ⟨outfile.nav⟩
```

This procedure generates a scene description file which can be processed with `navsetup` as described in chapter 5 to generate an input file for the calculation. The `vrml2nav` utility writes some default values for the different parameters like Reynolds-number etc. in the nav-file. These parameters as well as the grid resolution should be adjusted according to the calculation to be done. The nav-file generated by `vrml2nav` can be edited with any

[1] Virtual Reality Modelling Language, see http://www.web3d.org/technicalinfo/specifications/VRML1.0/

text-editor and all parameter settings explained in section 5.2 can be used. The `poly`-blocks in the scene description represent the actual geometry given in the VRML 1.0 file.

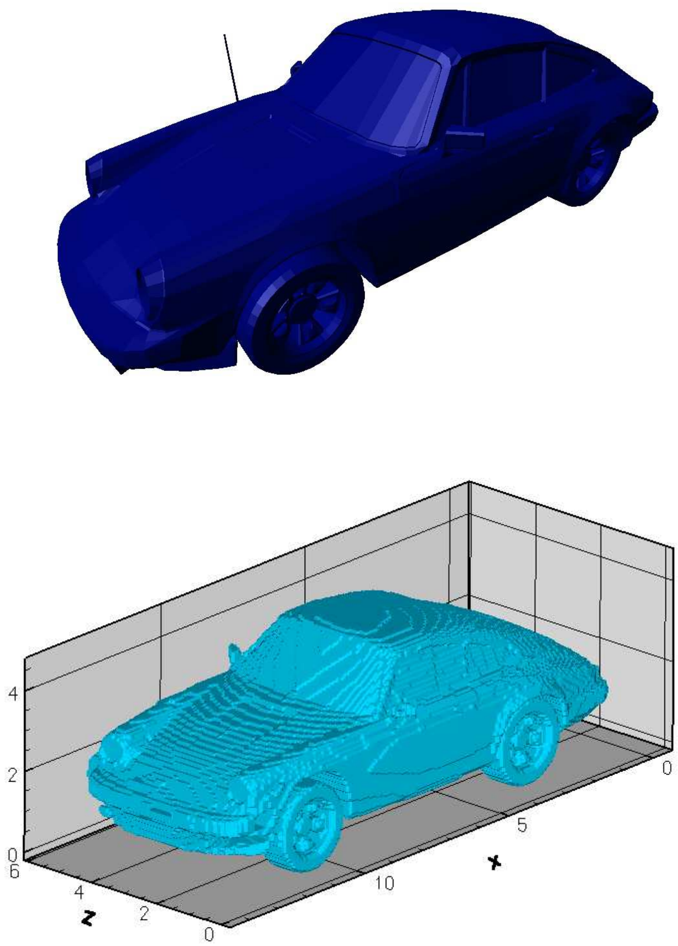

Figure 6.1: Snapshot of a Porsche model, VRML 1.0 scene (top), Tecplot visualization of flag field(bottom)

In figures 6.1 and 6.2 two examples are given. On the left a screenshot of the original VRML 1.0 description is shown, on the right side a visualization of the corresponding translation into the nav-format is shown. The visualization was done with Tecplot in the following way: first the binary data file was generated with `navsetup`, then, also with `navsetup`, output for Tecplot was generated (see chapter 5) and within Tecplot the isovalue 1.0 of the variable `flg` (which stands for flag field) is visualized. For the Porsche, a mesh resolution of 180 cells in x-direction, 90 cells in y-direction and 120 cells in z-direction was used, the

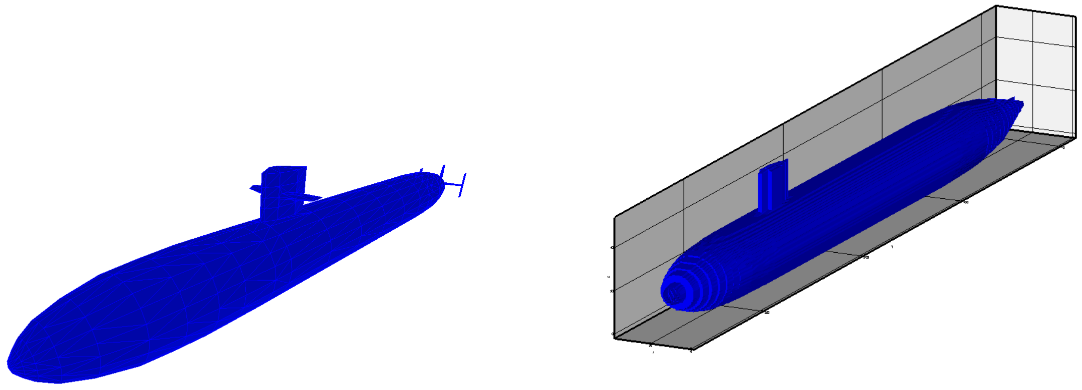

Figure 6.2: Snapshot of a submarine model, VRML 1.0 scene (left), Tecplot visualization of flag field(right)

mesh for the submarine has 128 cells in x-direction and 32 cells in y- and z-direction. Note that very thin features (for example, like the antenna of the Porsche car are translated by the `vrml2nav` converter, but can of course only be represented in the binary data file if the mesh resolution is fine enough.

6.2 GridGen

Another useful tool included in the distribution is located in `/tools/GridGen`. This utility can be used for the generation of smoothly distributed grid lines for one coordinate direction. Such smooth grid spaces are required to preserve the second order accuracy of the difference stencils. `GridGen` uses routines from [6].
`GridGen` is included in the distribution only for convenience and is not officialy supported. It is distributed as source only, you need a Fortran-77 compiler to build a binary. Since the tool consists of only one source file, this is a straightfoward compile process, e.g. with the Gnu Fortran compiler you would just use `g77 GridGen.f -o gridgen` which would produce the binary file `gridgen`.
The necessary parameters for the use of `GridGen` are explained in table 6.1.
`GridGen` reads these parameters from standard input. If you run `GridGen` without arguments, it will prompt you for each needed parameter, but it may be be more convenient to put the parameters in a simple text file (just one parameter a line) and run `GridGen` with the command `GridGen < input.dat`.
An example file named `gg-input.txt` is located in `tools/GridGen`. Running `GridGen` with this file as input produces a point distribution with a grid spacing of approximately

Table 6.1: Necessary parameters for `GridGen`

value	type	description
L	int	number of locations where grid spaces should take prescribed values
x0 : xL	float	locations in increasing order
dx0 : dxL	float	prescribed grid spaces at locations x0...xL
K	int	number of grid lines, which should have exactly some prescribed coordinates
xe0 : xeK	float	locations in increasing order Note, xe0==x0 and xeK==xL are required This feature allows to generate meshes, which respect some geometrical constraints
fak	float	the grid spaces dx0,..dxL can be multiplied with fak. This allows a simple coarsening / refinement of the computational grid
"workdir"	string	a directory in which the output files should be written

.01295 at the boundary points 0 and 1 and a spacing of approximately 0.0518 around the midpoint.
The results of `GridGen` are written to the following files:

`gridpoints.gnu`	contains a numbered list of the gridpoints which can then be used in a scene description file (see section 5.2)
`gridspaces.gnu`	this file contains the distribution of grid spaces

Both files can be plotted by `gnuplot` to control the results.

6.3 Interface to MatLab

MatLab is an extremely powerful software package that is very useful for post- and preprocessing as well as analysing the computed data in detail. Therefore we include two MatLab-scripts in the NaSt3DGP-package. These scripts, `ReadNaSt3D.m` and `WriteNaSt3D.m`, can be used within Matlab to read or write data in the format that **navsetup** produces when using the `-g` switch. The format of these files is shown in table 6.2.
After installation of the NaSt3DGP-package, the MatLab-scripts can be found in `tools/matlab`.

type	count	description
int	3	Number of cells (M, N, K) in each direction (including ghost cells)
float	M	Coordinates of grid lines in x-direction
float	N	Coordinates of grid lines in y-direction
float	K	Coordinates of grid lines in z-direction
float	$M \cdot N \cdot K$	Data of the corresponding field

Table 6.2: file format generated by `navsetup -g`

6.4 Interface to VTK

The Visualization Toolkit (VTK) is a freely available[2] software package for visualization, 3D computer graphics and image processing. It consists of a C++ class library and interfaces to several scripting languages such as Python or Tcl/Tk.
For visualization of NaSt3DGP-generated data with VTK we have included the script `viewNaSt3DGP.tcl` in the package. It uses the Tcl/Tk bindings of VTK to provide an easy-to-use GUI to control the displayed data. The script requires a proper installation of VTK (release 4.0 or newer), compiled with support for the Tcl/Tk bindings.
After installation of NaSt3DGP the script can be found in `tools/visualization/vtk/`.
To visualize data using `viewNaSt3DGP.tcl` first create files in VTK-readable format using `navsetup` with the `-G` and `-o ⟨datafile⟩` switch (this will create the files ⟨datafile⟩.vtk.p, ⟨datafile⟩.vtk.u, ⟨datafile⟩.vtk.v and so on, see section 5.1 for details). While converting the data `navsetup` will produce output which contains a passage similar to that in table 6.3.

Table 6.3: Sample output of navsetup while converting data.

```
Writing data.vtk.fl...
Writing data.vtk.u...
u: min  -1.70416e-07   max  1.70436e-07
Writing data.vtk.v...
v: min  -2.58760e-07   max  2.16669e-07
Writing data.vtk.w...
w: min  -3.55411e-09   max  3.55600e-09
Writing data.vtk.p...
p: min  -3.02805e-01   max  3.02544e-01
Writing data.vtk.t...
t: min  2.92460e+02   max  2.93542e+02
```

[2] see http://public.kitware.com/VTK/ for further information and download

Now you may use the command `vtk viewNaSt3DGP.tcl` ⟨`datafile`⟩ to start the Tk-based GUI (depending on the locations of your datafiles and `viewNaSt3DGP.tcl` you may have to supply appropriate paths to the above command). This will produce two windows on your screen (see figure 6.4), the Tk-based GUI and the window into which VTK renders its output. Initially the render window will only show the bounding box of the data set (you can turn the bounding box on and off by clicking the button labeled “Frame”).

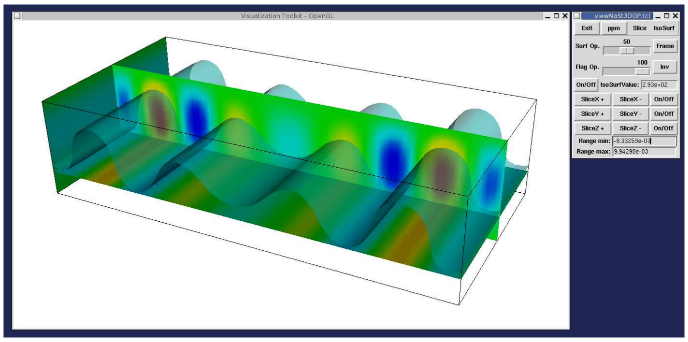

Figure 6.3: The VTK render window and the viewNaSt3DGP GUI

Inside of the render window you can use the buttons of your mouse to rotate (left button), move (middle button/both buttons) and zoom (right button) the displayed object. The button labeled “Exit” of course ends the application. Clicking the button “ppm” will export the current content of the render window to a ppm file, the files will be named snapshot_1.ppm, snapshot_2.ppm and so on (Note, however, that the window content is not rendered off-screen, so e.g. overlapping windows or the mouse-pointer would also be exported to the ppm-file).
You can now add selected data using the two pulldown-menus on the top right of the GUI window. The menus “Slice” and “IsoSurf” are used to set the field variable which is to be displayed either on cut-planes (slices) or on an isosurface, respectively. In the entry field “IsoSurfValue” you have to enter the value for which the isosurface should be computed. With the button labeled “On/Off” you can toggle the isosurface. The two sliders are used to control the opacity of the isosurface and the displayed flag field (if there are obstacles in your data file). With the remaining buttons you can toggle the visibility of the cut-planes or move them through the domain. To adjust the color palette to the range of the displayed variable simply enter the appropriate maximum and minimum values into the correspondingly labeled fields.

Chapter 7

Examples

In this chapter we present some examples of calculations done with the NaSt3DGP-package. The scene description files for these and some other testcases are included in the distribution and can be found under the `examples`-directory.

7.1 Lid-driven cavity

A basic scene description file for the lid-driven cavity problem can be found in the examples directory in the subdirectory Cavity2D (this file was also used in the introductory chapter). For this calculation some parameters have to be adjusted. These benchmark results refer to the cavity flow at Re = 1000. For the convective terms, the 2nd/3rd order VONOS scheme was used. Calculations are done on a 48x48 grid and a 96x96 grid. The Poisson solver is stopped when Res$< 1e-8$.

All test computations were finished after 300 time units. The level-values for the stream-function/vorticity are taken from [4].

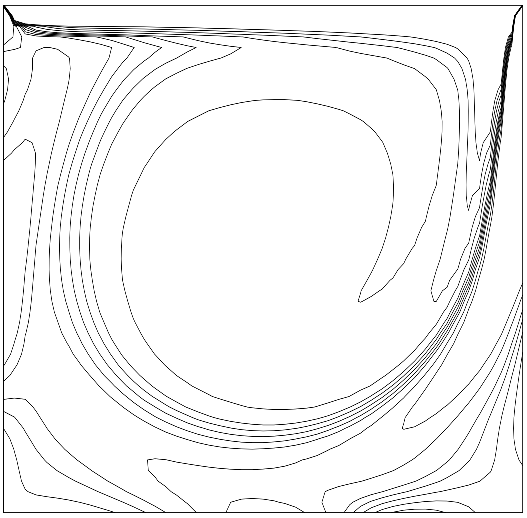

Fig.6: Vorticity, 48x48 grid

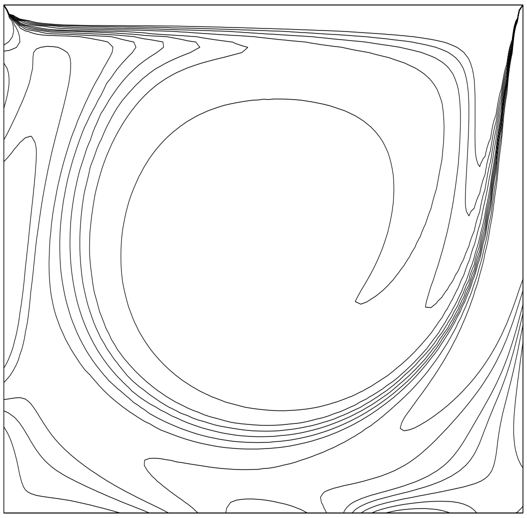

Fig.7: Vorticity, 96x96 grid

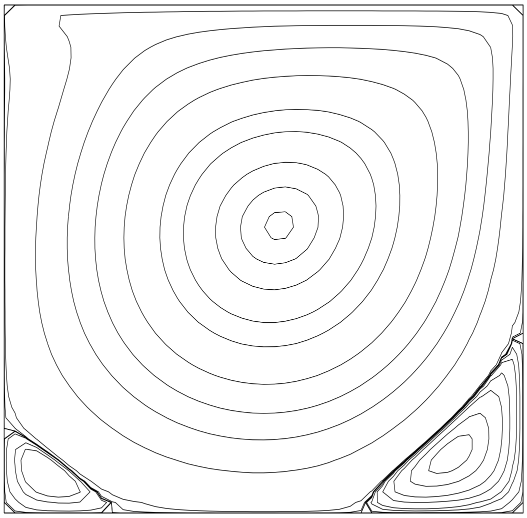

Fig.8: Streamfunction, 48x48 grid

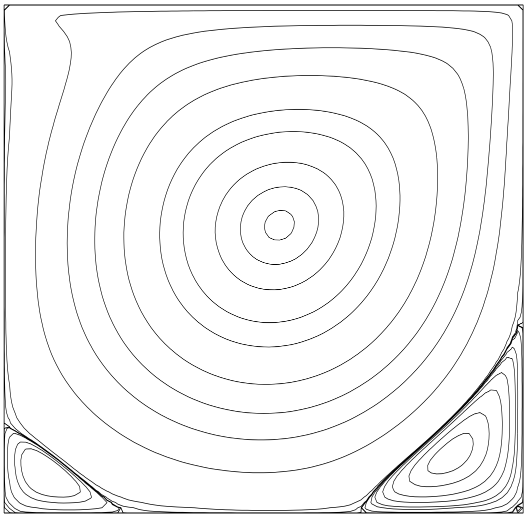

Fig.9: Streamfunction, 96x96 grid

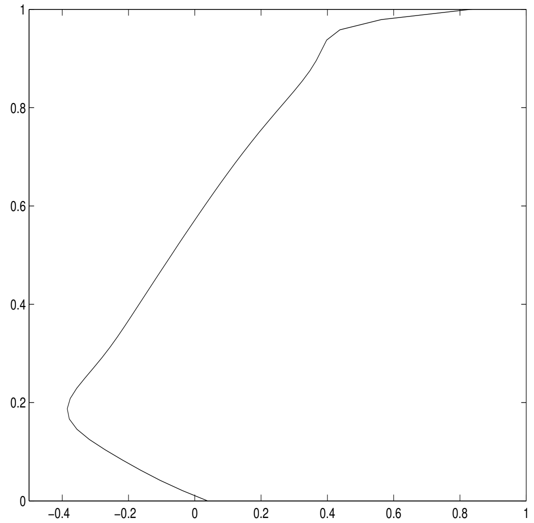

Fig.10: U-values through the vertically geometric center, 48x48 grid

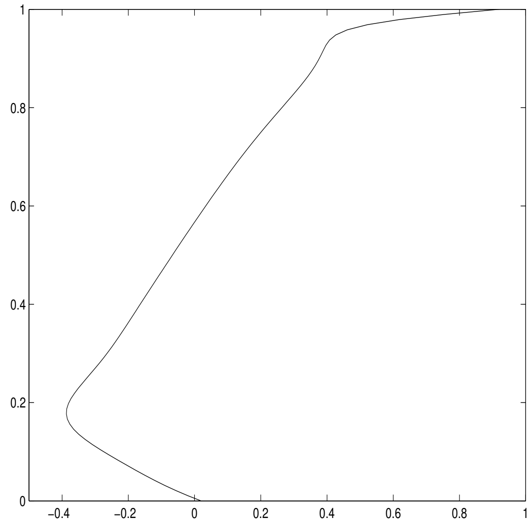

Fig.11: U-values through the vertically geometric center, 96x96 grid

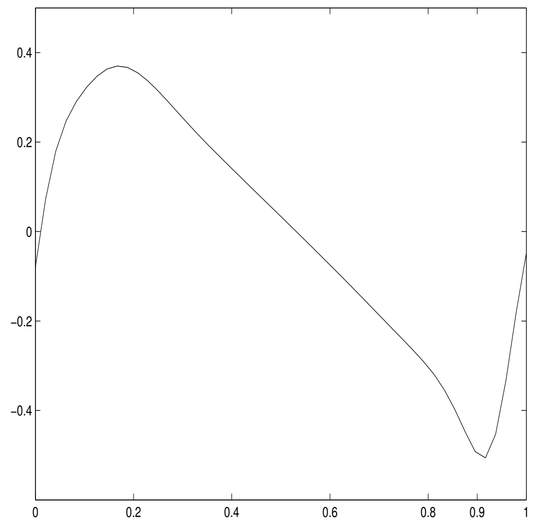

Fig.12: V-values through the horizontal geometric center, 48x48 grid

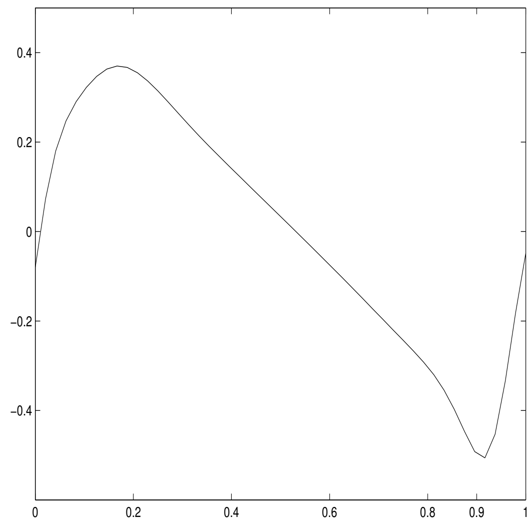

Fig.13: V-values through the horizontal geometric center, 96x96 grid

Chapter 8

In case of Trouble

8.1 Submitting bug reports

If you obtain any compile or runtime errors please report the bug via email to `nast3dgp@ins.uni-bonn.de`.
To simplify our handling of error reports please include the following information

- The version of NaSt3DGP.
- In case of build problems: the output of the make/compile process
- In case of runtime errors: the output of navsetup, navcalc or navcalcmpi.
- A detailed description of machine and operating system.
- The scene description file.

Please send reports about more than one problem in separate messages.

Chapter 9

License

Following is the license under which the software package NaSt3DGP is distributed. This license is also contained in the file `COPYING` included in the distribution.

License Agreement for the Software Package NaSt3DGP

between the Institut für Angewandte Mathematik der Rheinischen Friedrich-Wilhelms Universität Bonn - hereinafter referred as licenser - and You - hereinafter referred as licensee.

§1 Object of Agreement

The licenser lets to the licensee the right of the temporarily unrestricted use of the software package NaSt3DGP.

§2 Property and Copyright

The licenser is the owner of all rights of NaSt3DGP including all contributions made by others. We do this to make the code freely available on one hand, and on the other hand, to synchronize and steer the further development. Of course we appreciate and acknowledge the contributions of others by maintaing the "Thanks" section. You can also add 'Programmed by ...' comments in the code that you contribute. If this is unacceptable for you, then don't contribute code.
The licensee agrees to publish scientific results obtained with NaSt3DGP with an appropriate citation of the licenser. Details for appropriate citation are stated in the next section.

§3 Appropriate Citation

The licensee agrees that any reports or published results obtained with the Software NaSt3DGP will acknowledge its use by the appropriate citation of the licenser as follows:

"NaSt3DGP was developed by the research group in the Division of Scientific Computing and Numerical Simulation at the University of Bonn."
Any published work which utilizes NaSt3DGP shall include the following references:
1. M. Griebel, T. Dornseifer and T. Neunhoeffer, Numerical Simulation in Fluid Dynamics, a Practical Introduction, SIAM, Philadelphia,(1998)

§4 Obligations of the Licensee

It is in the licensees responsibility that all employees and students with access to NaSt3DGP follow the restrictions of this agreement.
It is not permitted to sell NaSt3DGP or parts of it or results obtained with it. Furthermore, it is not allowed to use NaSt3DGP for any commercial purposes.
Any modifications or extensions made by the licensee fall under these stipulations.

§5 Liability

NaSt3DGP is an experimental code. The licenser does not guarantee that NaSt3DGP is error free and that it complies to the special requirements of the licensee. The licenser does not assume any liability.

§6 Supplementary Provisions

The agreement shall be governed and construed in accordance with German law. The place of jurisdiction is Bonn/Germany.

Bibliography

[1] M. Griebel, T. Dornseifer and T. Neunhoeffer *Numerical Simulation in Fluid Dynamics, a Practical Introduction*, SIAM, Philadelphia,(1998)

[2] J.G. Heywood, R. Rannacher and S. Turek *Artificial Boundaries and Flux and Pressure Conditions for the Incompressible Navier Stokes Equations*

[3] V.G. Ferreira, M.F. Tome, N. Mangiavacchi, A.O. Fortuna, A.F. Castello, J.A. Cuminato, S. McKee *High Order Upwinding and the Hydraulic Jump*

[4] O.Botella and R. Peyret *Benchmark Spectral Results on the Lid-Driven Cavity Flow,* Computers & Fluids Vol. No. 4 pp. 421-433, 1998

[5] Croce *Ein paralleler, dreidimensionaler Navier-Stokes Löser für inkompressible Zweiphasenströmungen mit Oberflächenspannung, Hindernissen und dynamischen Kontaktflächen,* Masters Thesis, University of Bonn, 2002

[6] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery *Numerical Recipes in Fortran: the Art of Scientific Computing,* Cambridge University Press, 1986