TYrolian
Computational
Hydr**O**dynamics
version 1.3.

Wolfgang Kapferer

October 10, 2013

Abstract

TYCHO is a multidimensional compressible hydrodynamics code written in C and parallelized with Open-MP. A Lagrangian remap version of the Piecewise Parabolic Method developed by Paul Woodward and Phil Colella (1984) is applied. The code is based on the freely available VH-1 Package. The simulation package is focused on gas-obstacle interactions and has special routines for obstacles in wind streams and advection of marker fields for investigations of obstacle-gas interactions. Version 1.3 inlcudes sound-emitter routines to calculate dB-maps. TYCHO is freely available to everyone. You are welcome to download it and do whatever you want with it. I would appreciate, however, if you would acknowledge the package in you publications/work and if you send me information for what purpose you use the code. Keep in mind that this code does not come with any guarantee.

Contents

Chapter 1

Setup the code

1.1 Compilation

As the code is written completely in C you need a C Compiler (e.g. the GNU C Compiler). If you wish to deploy a parallel execution of your simulation you should compile the Code with Open-MP functionality (e.g. GOMP. The package comes with the common

- ./configure
- make
- make install

procedure. The **make install** is typically not necessary. Just copy the TYCHO executable, which you find after a successful compilation in the sources directory, in a working directory of your choice. To give flags to the compiler you can set them by **configure**-arguments (e.g. ./configure CFLAGS="-O3 -fopenmp" LIBS="-lm -lgomp").

1.2 The Parameter File

The code needs a parameter file at start up given as an argument (e.g. ./ tycho parameterfile.txt). The file includes all important parameters such as

- input_/_output directories
- dimension(1D/2D/3D) and resolution
- wind on/_off

- windspeed
- ....

Note that the file needs to have all listet parameters, otherwise TYCHO would mix up things. The parameterfile for TYCHO version 1.2 an lower had 47 paramters. The difference is a viscosity switch, the heat-capacities for gas and the obstacles for the thermal-exchange between gas and obstacles and the new sound module - use tychoGUI for generating parameterfiles. One parameter disappeard in TYCHO version 1.0 compared to TYCHO version 0.93 and below: the thermal diffusivity of air.
A detailed description of all parameters follows:

1.2.1 The Header

A simple Header of the parameter file.

```
#-------WELCOME TO TYCHOS PARAMETER FILE--------------
#COMMENTS HAVE TO START WITH #
#THE ORDER OF PARAMETERS HERE IS MANDATORY
#IF YOU WISH TO INCLUDE MORE PARAMETERS HERE
#MAKE SURE TO EDIT START_FILE_READER.C
#HAVE FUN
#-------------------------------------------------
```

1.2.2 Filetypes

You can choose between TYCHO native Filetype [0], VTK filetype [1], AMIRA MESH filetype [2] and the IFRIT filetype [3] At the moment you can only start from a TYCHO native file. More information about this filetype can be found in the next chapter.

```
#TYCHO Files[0], VTK RECTILINEAR_GRID BINARY Files [1], AMIRA MESH [2]
#or IFRIT uniform scalar data files [3]
0
```

1.2.3 Initial conditions

If one wants to start the simulation with self made initial conditions one has to set this parameter to 1. Note that the files have to be in TYCHO's native fileformat. If the code should set up initial conditions as specified in the function make_ic (make_ic.c) the parameter has to be set to 0. In the case

only the obstacle distribution should be read in you have set the parameter to 2.
If you want to test the code with a Sod shock-tube problem you have to set the parameter to 3. The problem can be generated in 1D/2D and 3D.
A Kelvin Helmholtz instability test as described in the section examples in 2D and 3D can be set up with 4. The file type used in TYCHO is specified in the section about TYCHOs file type.

```
#Make ICs by editing the source file make_ic.c yes [0]
#Read in initial conditions [1]
#Only the obstacle distribution is read in [2]
#A Sod Shock Tube is generated with [3]
#Kelvin Helmholtz instabilities in 2D [4]
#Your initial conditions have to be in TYCHO file format
0
```

1.2.4 Paths of input files

The initial conditions in TYCHO file format. If you do not need an initial file, e.g. no initial velocity distribution, then write just dummy at the position of the initial velocity file. See the example-files in the parameterfile subdirectory of the code.

```
#Initial Conditions
#Initial density
#absolute path of the density file
#e.g. /home/user/tmp/rho_ic.tyc
#if not needed just write
dummy
#Initial temperature
#absolute path of the temperature file
#e.g. /home/user/tmp/temp_ic.tyc
#if not needed just write
dummy
#Initial velocities
#absolute path of the velocity file
#e.g. /home/user/tmp/vel_ic.tyc
#if not needed just write
dummy
#Initial wind
#absolute path of the wind emitter file
```

```
#e.g. /home/user/tmp/wind_ic.tyc
#if not needed just write
dummy
#Initial obstacles
#absolute path of the obstacle file
#e.g. /home/user/tmp/obstacle_ic.tyc
#if not needed just write
dummy
#Initial sound-emitter
#absolute path of the sound-emitter file
#e.g. /home/user/tmp/soundemitter.tyc
#if not needed just write
dummy
#Initial marker
#absolute path of the marker file
#e.g. /home/user/tmp/marker_ic.tyc
#if not needed just write
dummy
```

Sometimes one wants just very small initial velocities. By stating no file present [0] very small (i.e. $1E-50$[m/s]) velocities are given to the gas.

```
#Initial velocity field file present [1] or not [0]
0
```

1.2.5 Output directory

Here the path to the output files is specified. It is important to note here, that the / at the end is mandatory.

```
#Output Directory
/tmp/
```

1.2.6 Stratified atmosphere

The initial condition generation in the function make_ic is able to set up a stratified atmosphere. The stratification is defined as follows:
temperature gradient $T(z) = T_0 + A(z - z_0)$ with $T_0 = 288.15$[K] and $A = -6.5 \times 10^{-3}$[K/m],
density gradient $\rho(z) = \rho_0 \left(\frac{T(z)}{T_0}\right)^{\frac{-g}{RA}+1}$ and pressure gradient $p(z) = p_0 \left(\frac{T(z)}{T_0}\right)^{\frac{-g}{RA}}$, with R the gas constant and g the gravitational acceleration. If you want this

feature you have to choose 1. A constant atmosphere without any gradient is realized with 0.

```
#Stratisfied Atmosphere [1] or constant Atmosphere [0]
0
```

1.2.7 Resolution

The resolution for all three dimensions are given here. If you want a 1D/2D simulation, you have to set the remaining two/one dimensions to 1.

```
#Basic_properties
#Resoution
#x y z
500
500
1
```

1.2.8 Dimensions

Here the dimension of the simulation is set.

```
#Dimension
2
```

1.2.9 Extend of the computational domain

The actual physical extend of your computational domain.

```
#Length scale
#meter in x,y,z
10.0
10.0
1.0
```

1.2.10 Boundary conditions

You have to specify all boundaries, even for 1D/2D simulations, although only the necessary will be applied. Zero gradients copies the last boundary cell values into ghost cells, reflecting boundaries are self-explaining and small padding means $1E-50$ values, i.e. small, are copied in the ghost cells. Inflow and Outflow boundary conditions are explained in detail in the chapter about

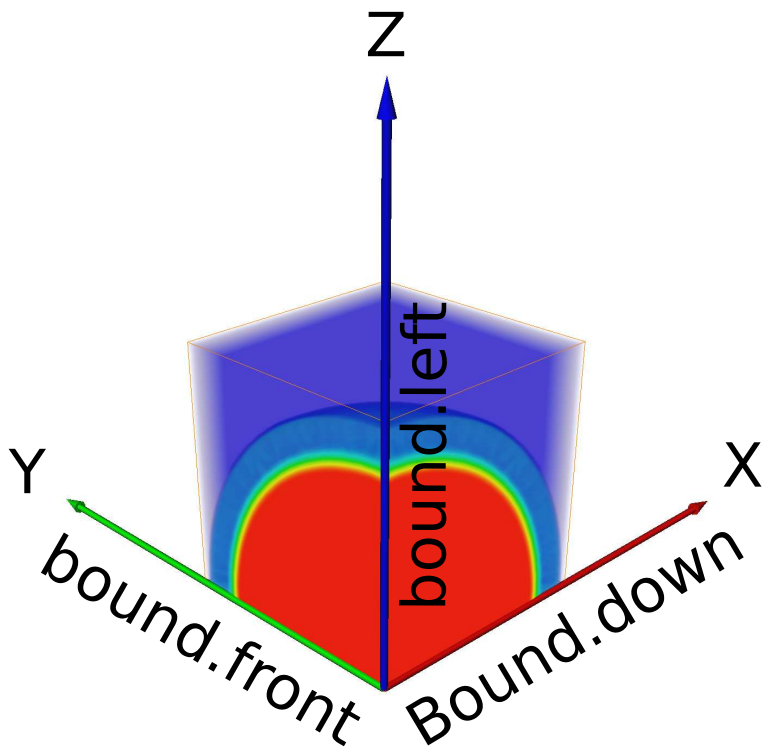


Figure 1.1: TYCHO's boundaries orientated in space.

boundary conditions. Periodic boundaries are needed often for test scenarios (e.g. Kelvin Helmholtz instability tests.)
In Fig. 1.1 the orientation of the boundaries in space is given. Note that in the case of 1D/2D simulations the boundaries for the unused axes do not have any effect, but must still be present in the parameterfile.

```
#Boundary conditions
#Boundary conditions
#0.....zero gradient
#1.....reflecting
#2.....small padding
#3.....outflow boundaries
#4.....inflow boundaries
#5.....periodic boundaries
#bound.down, bound.up,bound.left, bound.right, bound.front, bound.back
#bound.down is set automatically to reflecting in case gravity is
#switched on
```

```
0
0
0
0
0
0
```

1.2.11 Wind-flow emitting boundaries

If inflow boundary conditions are applied, the inflow velocity [m/s] and density [kg/m^3] has to be defined here. If you want to start with the constant velocity and direction over the whole computational domain you have to set the last parameter in this section to [1] for "everywhere". If you want only inflow from boundaries and no velocity in the computational domain at the beginning you have to set this parameter to [0].

```
#if inflow boundaries the inflow velocities and density
#inflow velocity [m/s], inflow density [kg/m^3]
#and inflow temperature [K] and starting everywhere[1] or only from boundaries [0].
100
1.229
300
1
```

1.2.12 Simulation timings

```
#Simulation end in minutes
1
```

Here you can define the frequency for generating output into your output directory.

```
#Output frequency in seconds
0.05
```

Here the frequency in seconds for writing TYCHO restart files are given.

```
#restart frequency in seconds
0.01
```

1.2.13 The Courant-Friedrichs-Lewy number

The Courant-Friedrichs-Lewy number, 0.3 is quite safe for this solver.

```
#CFL Number
0.3
```

1.2.14 Constant gravity background

If you want to switch a constant background gravity on, the boundary bound.down has to be set to 1 and bound.up should be set to 0. In addition a stratified atmosphere should be set (see above). Gravity acts in negative y-direction.

```
#Gravity on [1] off [0]
0
```

1.2.15 Gas constant

The specific gas constant for dry air [$\mathrm{Jkg^{-1}K^{-1}}$].

```
#Gasconstant
287.058
```

1.2.16 Gamma Factor of the gas

The Gamma factor (here for ideal gas).

```
#Gamma
1.6667
```

1.2.17 Wind-emitters

If you want to include a wind as defined in the wind marker file you have to set the parameter to 1. The difference to the inflow boundaries is the possibility to include a wind throughout the whole computational domain.

```
#Wind [on--1] / [off--0]
1
```

The speed of gas at the location of the wind source.

```
#Wind speed [m/s]
30
```

1.2.18 OpenMP

If you have a multicore/processor machine at your hand you can specify the number of parallel executed threads for the hydro solver with this parameter.

```
#Number of Threads for OpenMP
8
```

1.2.19 Obstacles

If a obstacles are needed in the simulation.

```
#With obstacle [0--off]/[1--on]
1
```

The density of the obstacle material [kg/s^3].

```
#Obstacle density [kg/m^3]
7874
```

The starting temperature of the obstacle [K].

```
#Obstacle temperature [K]
320
```

The thermal diffusivity of the obstacles' material [m^2/s]. If this parameter is set to 0.0 no thermal-energy exchange between obstacle and fluid is calcualted.

```
#Obstacle Thermal diffusivity [m^2/s]
0.000023
```

The specific heat-capacity of the gas for the obstacle-fluid thermal exchange rate [J/kgK].

```
#Specific heat capacity gas [J/kgK]
1.0
```

The specific heat-capacity of the obstacle for the obstacle-fluid thermal exchange rate [J/kgK].

```
#Specific_heat capacity obstacle [J/kgK]
0.5
```

1.2.20 Advected marker field

A density field, which will be advected with the Hydro velocity-field.

```
#A Marker Field is advected with the HYDRO Simulation [0--off]/[1--on]
1
```

Density of the Marker field.

```
#Marker density [kg/m^3]
0.1
```

1.2.21 Gas viscosity

Viscosity of gas. Kinematic viscosity due to to Sutherland's law (in the temperature-range [200 ¡ K ¡ 1400]).

```
#Viscosity of gas [0--off]/[1--on]
1
```

1.2.22 Sound Emitter

Sound emitter inclusion in the simulation. At the place of a sound emitter (marked in the same fashion as a obstacle) a sound will be emitted. Either one pulse or a sinusodial signal with a given frequency.

```
#With_sound_emitter [0--off]/[1--on]
1
```

The sound-pressure level of the sound emitting region.

```
#Sound Pressure Level dB
100
```

In the case of one pulse [1], a pressure distrubance will be emitted at the beginning of the simulation, leading to no resolution limits. In the case of a periodic [0] sound emitting sinusodial signal the frequency has to be provided. In this case TYCHO recalculates the computationl domain extend in such a way that 100 cells span over a complete period. In this case TYCHO can simulate interference patterns in the sound maps.

```
#With_one_pulse or periodic[0--off]/[1--on]
0
```

The frequency of the sinusodial sound emitting signal.

```
#Sound Frequency Hz
10
```

The reflection coefficient of a boundary within the computational domain, linear scale. Total reflection 1, no reflection 0.

```
#Sound Reflection on Obstacle coefficient
1
```

The absorption coefficient of an oabstacle within the computational domain in dB scale.

```
#Obstacle Absorption Coefficient
20
#-------------------------------------------------------------
```

Chapter 2

Setting up Initial Condition

Basically TYCHO supports starting from initial conditions (ICs) present in files or automatically generated as defined in **make_ic.c**. If only the obstacle distributions, wind-emitters or maker fields are generated outside TYCHO, you can import just these and let TYCHO automatically set up the gas and all the rest.

2.1 Let TYCHO generate initial conditions

If you want TYCHO to generate ICs you have to edit the routine **make_ic** in the source file **make_ic.c**. TYCHO stores all hydrodynamic quantities in three dimensional arrays which are indexed in an simple way. Additionally the initial-conditions switch in the **Parameterfile** has to be set to **0**. In the following the default **make_ic**-function is shown and explained in some detail.

```
int make_ic(int x, int y, int z) {
    int i, j, k;
    double velo_x, velo_y, velo_z;

    double T0, T, A, rho0, pre0;
```

Here the constants for the implemented temperature distribution in a stratified atmosphere are defined.

```
    //T(z)=T_0 + A*(z-z0)
    T0 = 288.15; //[K]
    A = -6.5E-3; //[K/m]
```

Pressure and temperature at a certain level for the stratified atmosphere are assigned here.

```
rho0 = 1.229; // Density at sea level [kg/m^3]
pre0 = 1.013E5; //pressure at sea level [N/m^2]
```

In order to get a physical gas distribution we assign very **small** velocities to the gas.

```
velo_x = small;
velo_y = small;
velo_z = small;
```

If one needs a toy atmosphere to play around one has to switch in the **Parameterfile** the **Stratified Atmosphere** to **1**.

```
if (strat_const_atmos == 1) {
    for (i = 0; i < x; i++) {
        for (j = 0; j < y; j++) {
            for (k = 0; k < z; k++) {

            T = T0 + A * ((ymax / y) * j);

            rho[i][j][k] = rho0 * pow((T / T0), ....
                           (-grav_acc / (gasconstant * A) + 1));
            pre[i][j][k] = pre0 * pow((T / T0), ....
                           (-grav_acc / (gasconstant * A)));

            vx[i][j][k] = velo_x;
            vy[i][j][k] = velo_y;
            vz[i][j][k] = velo_z;
            }
        }
    }
}
```

If one needs a homogeneous atmosphere one needs to switch in the **Parameterfile** the **Stratified Atmosphere** to **0**.

```
if (strat_const_atmos == 0) {
    for (i = 0; i < x; i++) {
        for (j = 0; j < y; j++) {
            for (k = 0; k < z; k++) {
```

```
                    rho[i][j][k] = 1.2;
                    pre[i][j][k] = 300 * gasconstant * rho[i][j][k];

                    vx[i][j][k] = velo_x;
                    vy[i][j][k] = velo_y;
                    vz[i][j][k] = velo_z;
                }
            }
        }
    }

    return 0;
}
```

Assuming your simulation should include an obstacle build by TYCHO you have to define the shape, density and temperature in the **initiate_domain** function. An example for a 2D circle-like obstacle is given in the file **make_ic.c**. The obtacle_density and obstacle_temperature constants are specified within the **Parameterfile**.

```
/*!
 The form of the obstacle is defined in this function
 example of just a round thing.
 */
int initiate_domain(int x, int y, int z) {
    int i, j, k;

    for (i = 0; i < x; i++) {
        for (j = 0; j < y; j++) {
            for (k = 0; k < z; k++) {
                int i2 = i - x / 2.0;
                int j2 = j - y / 2.0;
                int xy = sqrt(i2 * i2 + j2 * j2);

                if (xy < x / 20) {
                    dom[i][j][k] = 1;
                    rho[i][j][k] = obstacle_density;
                    pre[i][j][k] = obstacle_temperature;

                    vx[i][j][k] = 0.0;
```

```
                    vy[i][j][k] = 0.0;
                    vz[i][j][k] = 0.0;
                } else {
                    dom[i][j][k] = 0;
                }

            }
        }
    }

    printf("Initiate domain done\n");

    return 0;
}
```

To summarize, defining your own ICs in the **make_ic**-function requires full specification of the five 3-dimensional arrays **rho, pre, vx, vy, vz**. If 2 dimensional simulations are needed, then just assign 1 to the index **k**. Remember that this code strictly uses **SI**-units. Consequently a 1D simulation needs x=1, y=1 and z=1.

2.2 Read in your own generated files

Applying your own generated ICs is possible through TYCHO's native file format, which is explained in detail in the chapter about supported filetypes. If you want to start with self-generated ICs or with files from another TYCHO simulation, you have to specify the appropriate files including the paths and the appropriate switches in the **Parameterfile**. The switch dealing with the initial velocity file presence **#Initial velocity field file present [1] or not [0]** allows you to choose if TYCHO reads in the given velocity ICs or ignores them. In case you want to ignore them you still have to give a dummy filename for the initial velocities. The IC read-in routine will extract the simulation time, the resolution and the counter from the given files, therefore you can basically restart TYCHO from every output fileset written in TYCHO's native fileformat.

2.3 Read in the obstacle distribution only

If you want to set up all hydrodynamic quantities by TYCHO as specified in the **make_ic**-function, but you want to create the obstacle distribution exter-

nally (e.g. with the **domain_composer.m** in the tools directory), you can specify in the **Parameterfile**. In this case only the initial-obstacle file will be processed, all the other files in the initial conditions section in the **Parameterfile** will be ignored, although dummy filenames have to bee given. The positions of the obstacles in the computational domain are defined by specifying the value 1 in the cells representing the obstacle and 0 everywhere else within the obstacle-array. The density and pressure values in the **rho and pre** array will be taken from the obstacle-density and obstacle-temperature parameters specified in the **Parameterfile**. The obstacle-temperature is needed for the thermal diffusion routine.

2.4 Read a wind marker file

In case one needs a wind source (like a ventilator) the feature has to be switched in the **Parameterfile** and a wind marker file has to be specified in the initial-files section. The wind-file provides the code basically the location and direction of wind sources, i.e. a constant velocity will be given to the gas at the wind marker positions. The wind direction is specified by the following numbers: 2 the +x direction, 3 the -x direction, 4 the +y direction,5 the -y direction, 6 the +z direction and 7 the -z direction.

2.5 Read a marker file for advection

In order to probe the gas-flow in the computational domain a density marker field can be advected with the hydrodynamic velocities. First the advection parameter in the **parameterfile** has to be set to 1 and an initial marker field has to be provided. The file provides simply an integer with the value 1 at the location of the marker-field. In the simulation a marker density will be assigned to the marker-field regions, which will than be advected.

2.6 Restarting mechanisms in TYCHO

In the **Parameterfile** a duration in seconds for writing restart files can be given. If the simulation has written restart files beforehand one can very easily restart a TYCHO simulation by giving an additional argument to the tycho executable up, e.g. **./tycho parameterfile restart**. TYCHO will pick up the existing restartfiles and restart the simulation from that point. If you want to restart from another timestep you can do that by reading in TYCHO output files in the TYCHO native fileformat and setting the

appropriate parameters in the **Parameterfile**. That means that if you write output to other filetypes then TYCHO's native fileformat, restarting from checkpoints is the only option. Note that at restart the **parameterfile** will be processed in addition, which allows you to alter boundary conditions or other parameters within the simulation at restart points. If you want to restart without any chances leave the **parameterfile** unchanged. The restart files have to be located in the output directory.

Chapter 3

The Boundary Conditions

TYCHO comes with five different boundary conditions to choose from, namely **Zero Gradient**, **Reflecting**, **Small Number Padding**, **Outflow**, **Inflow** and **Periodic**. These boundary conditions can be attached separately to all edges of the computational domain. The exact definition of the conditions can be found in the file **set_boundary.c**. In Fig. 3.1 the handling of the velocity vector for all different boundary conditions is given. In the case of the **Small Padding** boundary conditions the density, pressure and total energy is set to small (i.e. 1.0E-50) values in the ghost cells. The other boundary conditions copy density, pressure and total energy in the ghost cells, as defined in **set_boundary.c** . A special behavior is implemented for the **Inflow/Outflow** boundary conditions. In the first case a constant inflow of gas with a velocity, density and temperature specified in the **Parameterfile** is realized and in the second case gas which leaves the computational domain will not enter it again, because backflow of gas is prevented.
In the case of obstacles in the computational domain TYCHO will split the calculations into two parts, i.e. boundary at the computational domain and boundary at the obstacles. The boundaries at the obstacles are always reflecting.
In the following example source section taken from **set_boundary.c**, the handling of the left and right edges of the computational domain are shown. In principle one can easily implement own boundary conitions in the same way. The integers nmin and nmax stand for the first and last cells in the gas part of the computational domain. TYCHO has six ghost cells for the PPM Method. The arrays dx, xa, dx0 and xa0 contain the spacing and the positions of the left faces of the cells in code units.

```
/*!
 Reflecting boundary condition
```

```
 */
int left_boundary_reflecting(int nmin, int nmax,
    double *rho_1D, double *eng_1D, double *pre_1D,
    double *vx_1D, double *vy_1D, double *vz_1D,
    double *xa0, double *dx0, double *xa, double *dx,
    int flag) {

    int n;

    for (n = 1; n < 7; n++) {
        rho_1D[nmin - n] = rho_1D[nmin + n - 1];
        pre_1D[nmin - n] = pre_1D[nmin + n - 1];
        eng_1D[nmin - n] = eng_1D[nmin + n - 1];

        vx_1D[nmin - n] = -1 * vx_1D[nmin + n - 1];
        vy_1D[nmin - n] = -1 * vy_1D[nmin + n - 1];
        vz_1D[nmin - n] = -1 * vz_1D[nmin + n - 1];

        dx[nmin - n] = dx[nmin + n - 1];
        xa[nmin - n] = xa[nmin - n + 1] - dx[nmin - n];
        dx0[nmin - n] = dx0[nmin + n - 1];
        xa0[nmin - n] = xa0[nmin - n + 1] - dx0[nmin - n];
    }

    return 0;
}

/*!
 Reflecting boundary condition
 */
int right_boundary_reflecting(int nmin, int nmax,
    double *rho_1D, double *eng_1D, double *pre_1D,
    double *vx_1D, double *vy_1D, double *vz_1D,
    double *xa0, double *dx0, double *xa, double *dx,
    int flag) {
    int n;

    for (n = 1; n < 7; n++) {
        rho_1D[n + nmax] = rho_1D[nmax + 1 - n];
        pre_1D[n + nmax] = pre_1D[nmax + 1 - n];
        eng_1D[n + nmax] = eng_1D[nmax + 1 - n];
```

```
        vx_1D[n + nmax] = -vx_1D[nmax + 1 - n];
        vy_1D[n + nmax] = -vy_1D[nmax + 1 - n];
        vz_1D[n + nmax] = -vz_1D[nmax + 1 - n];

        dx[n + nmax] = dx[nmax + 1 - n];
        xa[n + nmax] = xa[nmax + n - 1] + dx[nmax + n - 1];
        dx0[n + nmax] = dx0[nmax + 1 - n];
        xa0[n + nmax] = xa0[nmax + n - 1] + dx0[nmax + n - 1];
    }

    return 0;
}
```

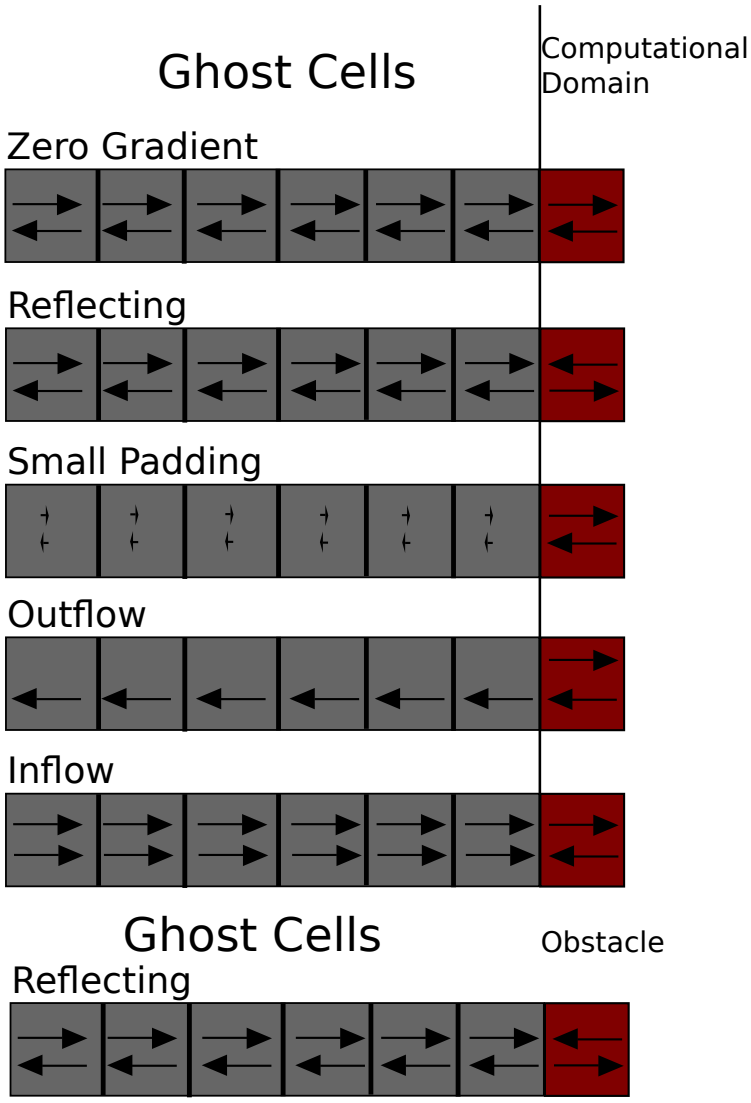


Figure 3.1: The handling of the velocities for the differnent implemented boundary conditions at the left edge of the computational domain.

Chapter 4

Viscosity and thermal diffusion and obstacle-fluid thermal-energy exchange

TYCHO calculates the influence of viscosity, if the corresponding switch is set in the parameterfile. The viscosity is derived in an explicit way, therfore the timestep calculation is altered to satisfy $dt < 0.5\frac{dx^2}{\nu}$, where ν is the kinematic viscosity [m^2/s]. The kinematic viscosity is a function of the gas' temperature and approximated ba Sutherland's law in the temperature regime [200 ¡ K ¡ 1400]. Velocity-fluxes in the hydro-core are altered by terms $F = -\nu\nabla v$. Special emphasis is given in the seperation of the computational domain in obstacle and fluid parts. Note that the routine requires more memory due to the viscosity calculation.
Thermal diffusion in obstacles is calculated in an explicit operator-splitting way. The Heat transfer from the obstacle to the gas is realised by providing the code the specific heat-capacities for the gas and the obstacle. In the routine **alpha_heat_transfer** the heat-transfer coefficient between obstacle and gas is derived as a function of gas velocity at the boundary. Change this function to serve your problem best.

Chapter 5

Sound emitters within the computational domain

Purpose of this feature is the investigation of sound-propagation in the computational domain. If enabled, TYCHO uses information provided in the sound-emitter initial-condition file. This file stores the loaction of a sound emitter within the computational domain, in the same manner as the boundary intitial condition file. In the case of a single pressure-pulse simulation, an over-pressure with the given sound-pressure level is emitted at the beginning of the simulation. With this information a sound-map, i.e. a dB map, is calcuated at each timestep. In the case of no-single sound pulse a sinusodial signal with the given frequency and sound-pressures level is constantly emitted at the location of the sound-emitter. In this case TYCHO has to recalcuated the extend of the computational domain in such a way the the given resolution can describe the sound wave in a computational-stable way. TYCHO will inform the user at startup about the new extend, so keep in mind that this can change the extend dramatically. In this case one can investigate interference patterns of sound waves propagating through the computational domain. The sound-reflection coefficient describes the amount of pressure reflected at the boundaries [1 total - 0 no reflection], whereas the boundary absorption coefficient [dB] gives the amount pressure gradient transported by the boundary. The applications tychoBCGEN and tychBCG3D assists you for placing sound-emitters in the simulation.

Chapter 6

The Filetypes TYCHO offers

In order to provide you some flexibility, you can choose between different filetypes for output data. Currently supported are

- TYCHO's native fileformat
- VTK RECTILINEAR_GRID BINARY files
- AMIRA MESH files
- or IFRIT uniform scalar data files.

6.1 The native TYCHO filetype

The native TYCHO filetype is a raw binary file in double precision. TYCHO writes density, temperature, velocity, obstacles, pressure on obstacles and marker-field files as output. The velocity files are written in the order v_x^i, v_y^i and v_z^i for a given cell$i$. The obstacles distribution files as well as the marker-field files are written in raw binary files with integer precision. Like for all filetypes,they are stored in the directory specified by the parameterfile. For each timestep defined by the output-frequency a set of files are generated with an ascending number in the filename and the extension **.tyc**. The header in each file has a size of 200 bytes and gives information about the content, the simulation time in seconds, resolution in human-readable form and a intern filecounter. An example Header for a TYCHO density file.

```
TYCHO Density File
1.7e-05
200
200
```

```
1
323
```

The C-routine writing the native TYCHO density file is shown here:

```
/*!
 In this routine a TYCHO density file is written.
 */
int write_tyc(int x, int y, int z, int counter) {
    FILE *fd;
    char filename[600];
    int i, j, k, tmp1;
    double tmp;

    sprintf(filename, "%srho_%i.tyc", output_dir, counter);
    fd = fopen(filename, "w");
    if (fd == NULL) {
        printf("-----------------------------------\n");
        printf("The output directory does not exist\n");
        printf("-----------------------------------\n");
        exit(13);
    }
    fprintf(fd, "TYCHO Density File\n%g\n%i\n%i\n%i\n%i\n",
            time_sim, x, y, z, counter);
    fseek(fd, 200, SEEK_SET);
    for (i = 0; i < x; i++) {
        for (j = 0; j < y; j++) {
            for (k = 0; k < z; k++) {
                tmp = rho[i][j][k];
                if (isnan(tmp)) {
                    printf("NaN in density array %i %i %i\n", i, j, k);
                    exit(0);
                }
                fwrite(&tmp, 1, sizeof (double), fd);
            }
        }
    }
    fclose(fd);

  return 0;
}
```

6.2 VTK RECTILINEAR_GRID BINARY files

The VTK RECTILINEAR_GRID BINARY files can be easily visualized with visualization packages like PARAVIEW (http://www.paraview.org/) or VISIT (https://wci.llnl.gov/codes/visit/). More information on this filetype can be found at http://www.vtk.org/.

6.3 AMIRA MESH Files

The AMIRA MESH files can be directly visualized with AMIRA (http://www.amira.com/) or AVIZO
(http://www.vsg3d.com/avizo/overview). These tools are commerical products. Information about the filetype can be found at
http://amira.zib.de/mol/usersguide/HxFileFormat_AmiraMesh.html.

6.4 IFRIT uniform Scalar Data File

The IFRIT package is a freely available visualization tool, capable to visualize scalar, vector and tensor data. It can be downloaded at
http://sites.google.com/site/ifrithome/.
The fileformat is explained in detail at
http://sites.google.com/site/ifrithome/Home/documentation/file-formats.

Chapter 7

Visualization of TYCHO data

To visualize the results of a TYCHO simulations PARAVIEW in combination with the VTK output is not a bad starting point. PARAVIEW can be obtained from
http://www.paraview.org/paraview/resources/software.html for Linux, Windows and MacOS platforms. Some Linux distributions,like FEDORA or UBUNTU, provide repository packages for PARAVIEW. In the Figs. 7.1 and 7.2 exemplory visualizations of a TYCHO simulations with PARAVIEW and AMIRA are shown.

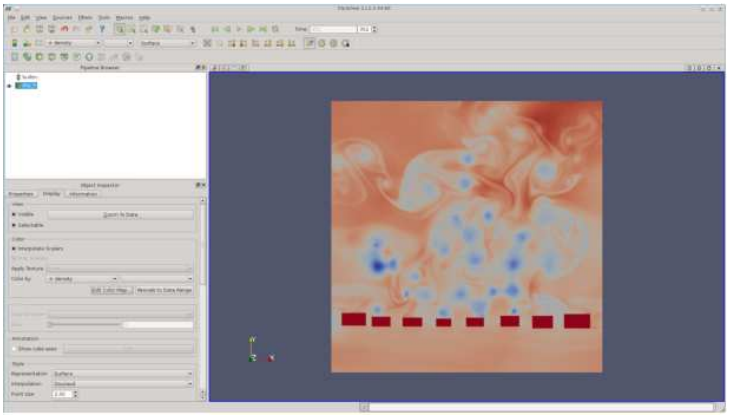


Figure 7.1: TYCHO simulation output visualized with PARAVIEW.

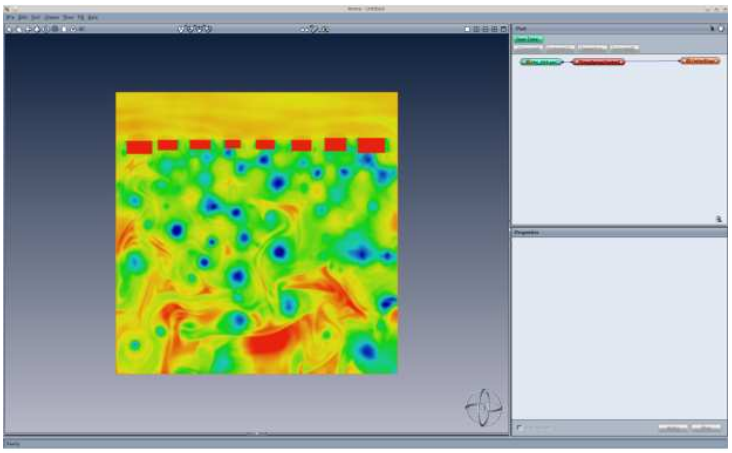


Figure 7.2: TYCHO simulation output visualized with AMIRA.

Chapter 8

The domain-marker and wind composer OCTAVE scripts

To provide the user with a simple starting point for one's own simulations, two OCTAVE scripts are provided to set up 2D initial conditions for TYCHO. The scripts are meant to generate initial obstacle and marker-field distributions for TYCHO 2D simulation. The scripts **domain_composer.m** and **marker_composer.m** are located in the tools directory and can be used with the freely obtainable **OCTAVE**. GNU OCTAVE is a high-level interpreted language, primarily intended for numerical computations and can be obtained at
http://www.gnu.org/software/octave/. Both scripts are kept very simple and provide a starting point to play around with the functionality of TYCHO. The functions of the scripts are:

- Setting the resolution of you 2D simulations.
- Including obstacles and/or wind-emitters.
- Setting the direction of the wind-emitters.
- Marking some gas.
- Setting location and shape of the markers.

The scripts are desigend to be rather self-explaining in their usage. Just execute them and have fun.

Chapter 9

Some examples

TYCHO comes with three small examples. In the parameterfile subdirectory the **parameterfiles** for these examples can be found. For three of them initial conditions are present in the subdirectory **initial_conditions**.
One example is a 2D Kelvin-Helmholtz instability, as shown in Fig. 9.1. The initial conditions are set up with a density contrast, two layers of lower density gas with a constant velocity and a small velocity disturbance at the contact layers of the two density gradients. The velocity disturbance is defined as follows in the **make_ic.c** file.

```
if (dimension == 3) {
if (j == y / 4) vy[i][j][k] =
sin(xmax / x * i) * cos(zmax / z * k);
if (j == 3 * y / 4) vy[i][j][k] =
-1 * sin(xmax / x * i) * cos(zmax / z * k);
}
if (dimension == 2) {
if (j == y / 4) vy[i][j][k] = sin(xmax / x * i);
if (j == 3 * y / 4) vy[i][j][k] = -1 * sin(xmax / x * i);
                }
```

A second example is a small **Toytown** to demonstrate TYCHO's handling of obstacles in the wind flow. At the beginning an obstacle distribution is read in (please adapt the **parameterfile** for your own pahts for the files and the output. A constant wind-flow from the left to the right blows through the "city". One can study the turbulence, the regions with high velocity and the resulting pressure on obstacles as shown in Fig. 9.2. Another example includes two wind-generators (e.g. ventilators or turbines) generating a wind-flow in opposite directions. In front of the ventilators a stream-splitting obstacle is placed and marker-fields are set up in that scenario. See Fig. 9.3

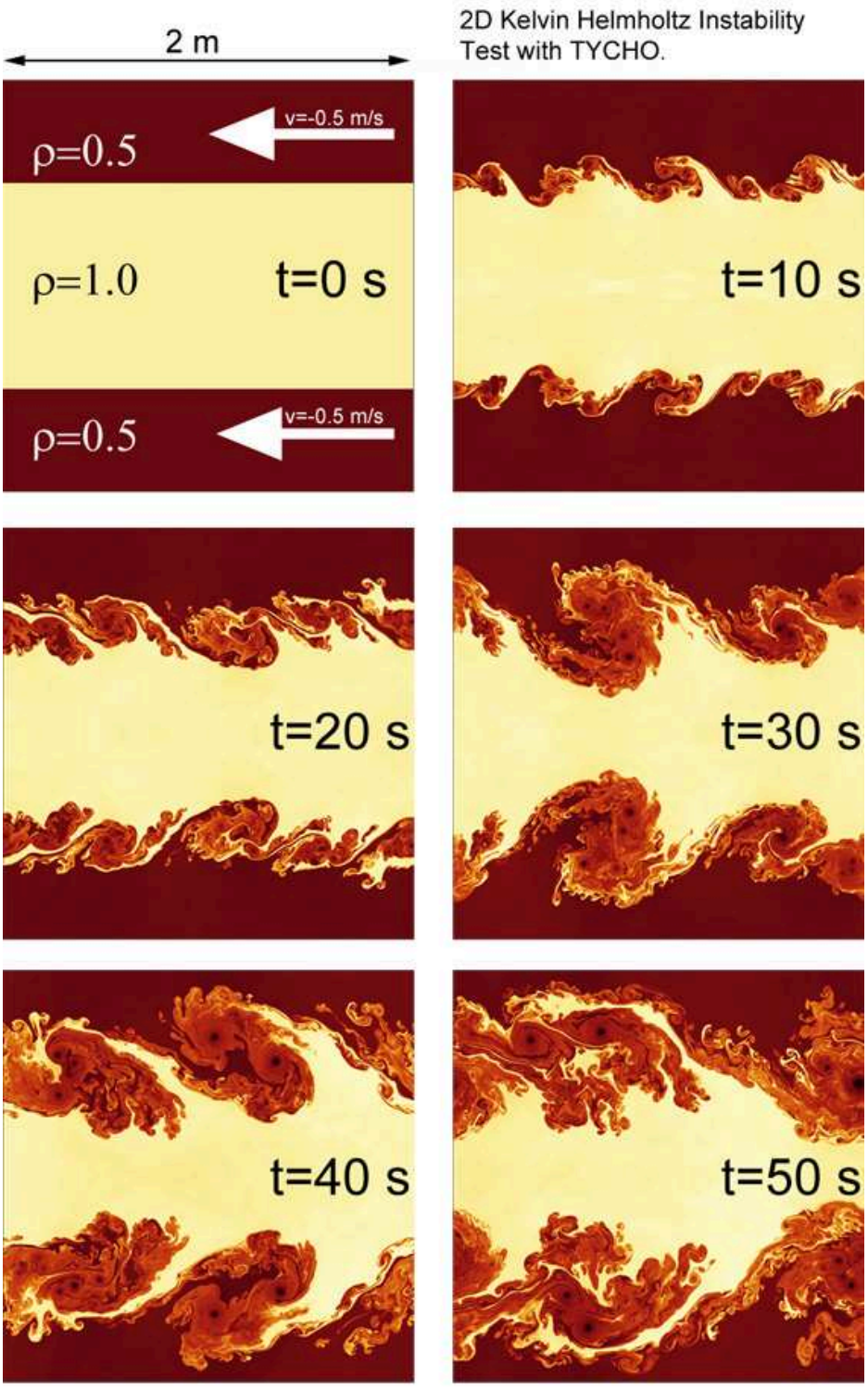


Figure 9.1: TYCHO simulation of a 2D Kelvin Helmholtz instability.

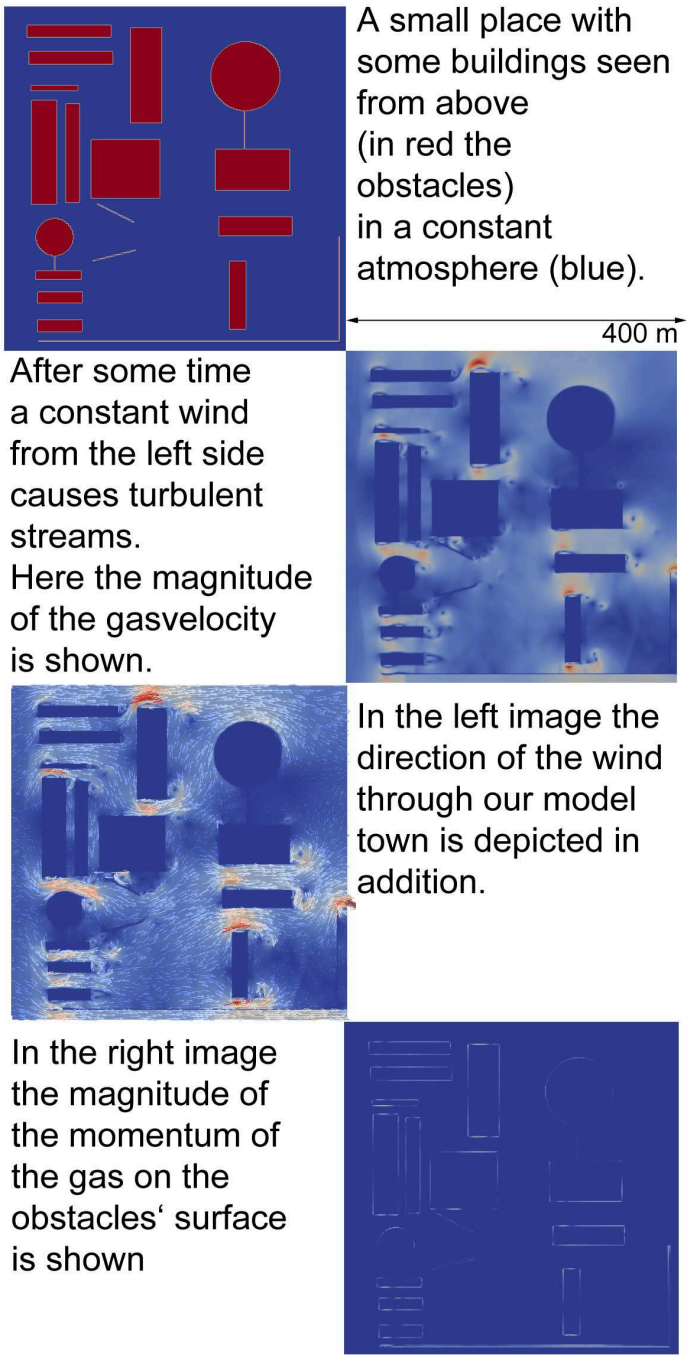


Figure 9.2: TYCHO simulation of a wind in a toy-town.

for some visualizations of this configuration. In the last example two wind-generators (e.g. ventilators or turbines) generating a wind-flow in the same direction through a tunnel. Several circle-shaped marker-fields are set up in that scenario. See Fig. 9.4 for some visualizations of this configuration.

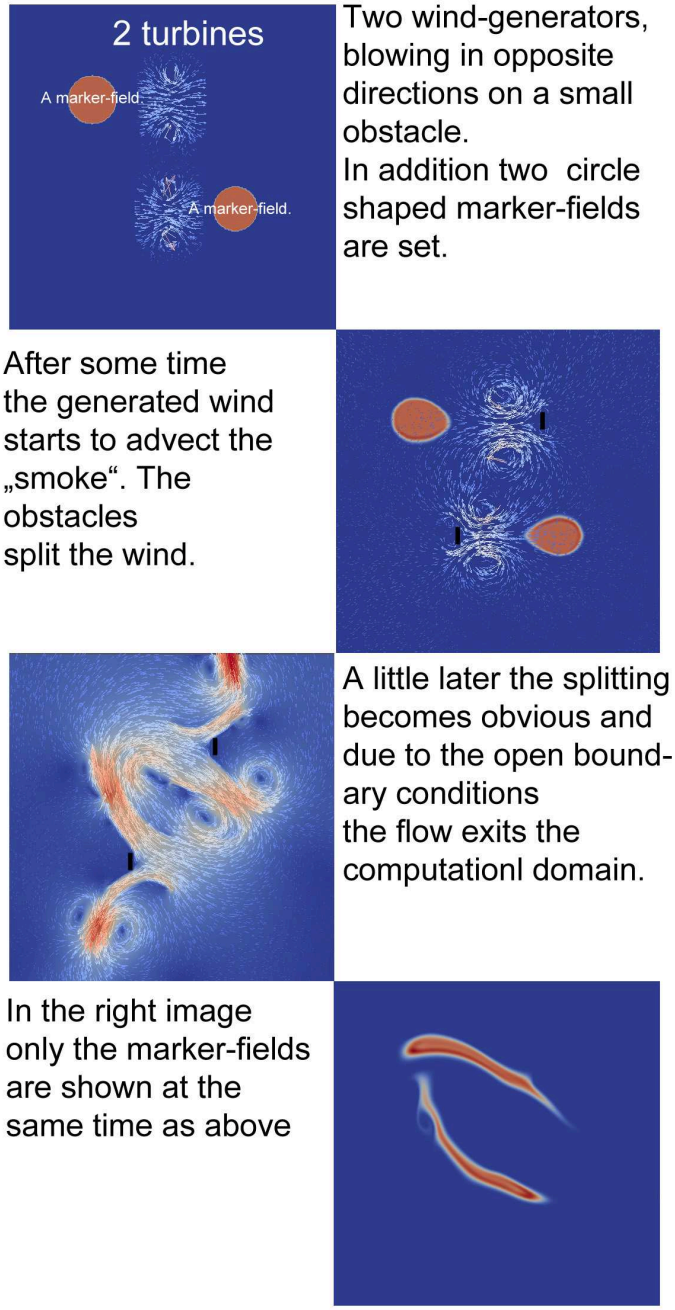
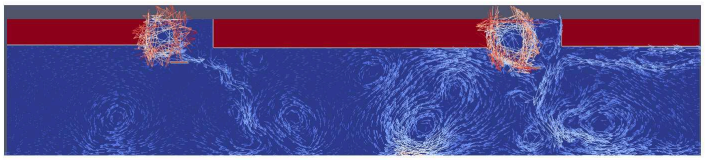
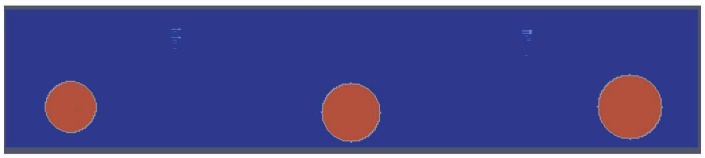


Figure 9.3: TYCHO simulation of two wind-generators.

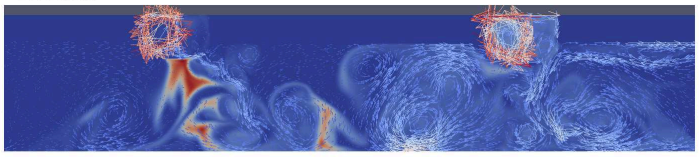
Two wind-generators, blowing in the same direction through a tunnel shortly after simulation start. The velocity field in addition to the obstacle distribution is shown.



The same quantities as is shown after some time. The turbulent behavior of the gas is now visible.



The velocity-field and three circle-shaped marker-fields are shown.



The same image as above after some time is depicted above. The mixing of the marked area is clearly visible.

Figure 9.4: TYCHO simulation of two wind-generators in a tunnel.

Chapter 10

The Sod Shock-Tube in 1D

The Sod shock tube problem, named after Gary A. Sod, is a test for the accuracy of the Riemann solver. In Fig. 10.1 and 10.2 the initial conditions for the 1D case are shown. The initial conditions are set up as defined here

$$\begin{pmatrix} \rho_l \\ p_l \\ v_l \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 0.0 \end{pmatrix}, \begin{pmatrix} \rho_r \\ p_r \\ v_r \end{pmatrix} = \begin{pmatrix} 0.15 \\ 0.1 \\ 0.0 \end{pmatrix} \tag{10.1}$$

After 0.2s the solution of the density and pressure are shown in Fig. 10.3 and Fig. 10.4. The typical regions are clearly visible, i.e. the rarefaction wave, the contact discontinuity and the shock discontinuity. More information on this test scenario can be found at
http://en.wikipedia.org/wiki/Sod_shock_tube.

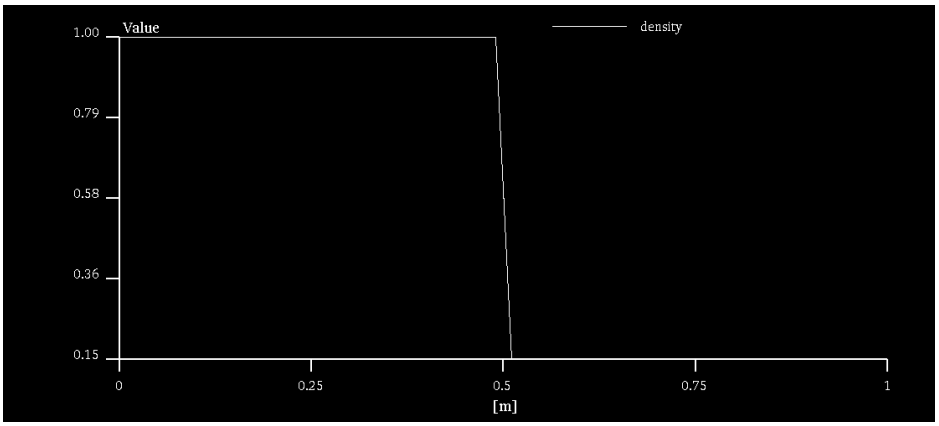


Figure 10.1: The initial conditions for the 1D Sod Shock-Tube problem (density distribution shown).

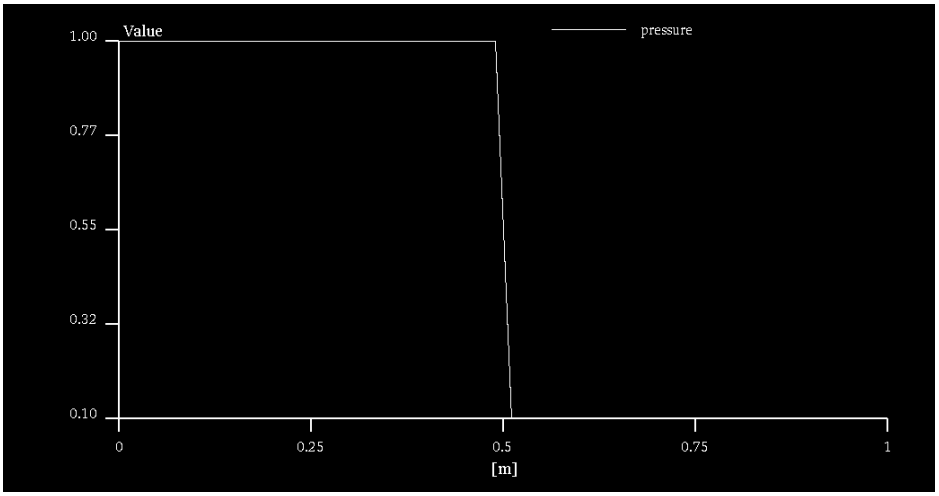


Figure 10.2: The initial conditions for the 1D Sod Shock-Tube problem (pressure distribution shown).

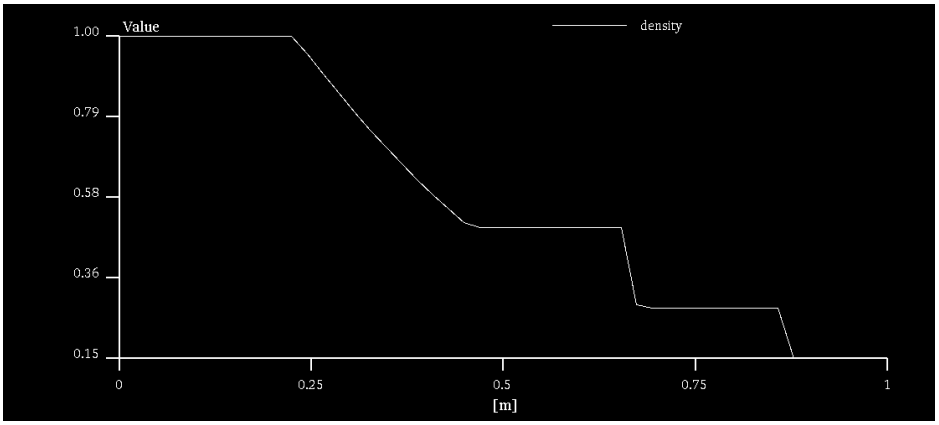


Figure 10.3: The 1D Sod Shock-Tube problem after 0.2 s of evolution. The density of the gas is shown.

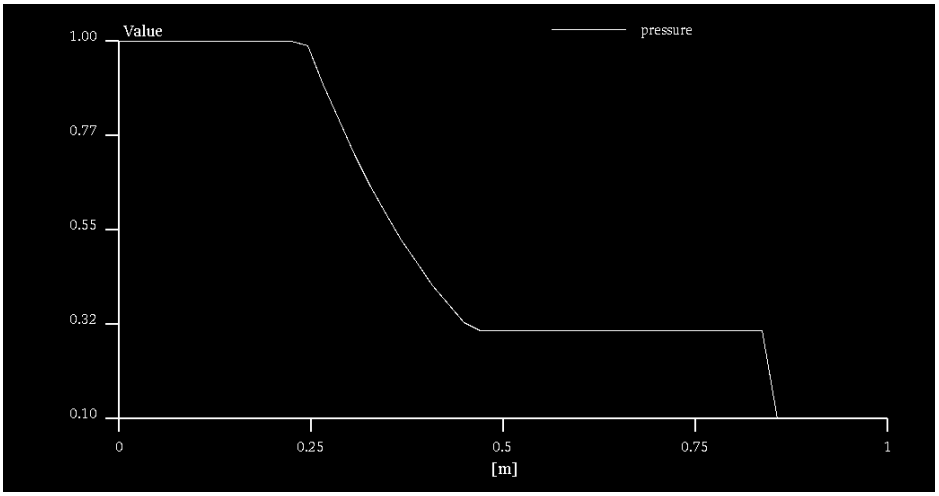


Figure 10.4: The 1D Sod Shock-Tube problem after 0.2 s of evolution. The pressure of the gas is shown.

Chapter 11

Copyright and legal information

TYCHO is freely available to everyone. You are welcome to download it and do whatever you want with it. I would appreciate, however, if you would acknowledge the package in you publications/work and if you send me information for what purpose you use the code. The code is based on the freely available VH-1 package. Keep in mind that this code does not come with any guarantee. If you need help or you want to highlight a bug/error (which I would appreciate) you can contact me via the channels provided at the website of this project.