GALCIT (Caltech)
Prof. Hans G. Hornung

Institute of Aerodynamics (TUM)
Prof. Dr.-Ing. N. A. Adams

Simulation of high enthalpy flow in porosities using SPH

Oliver Oberinger

Intermediate version of Master's Thesis

for presentation at

Institut Supérieur de l'Aéronautique et de l'Espace

responsible at ISAE: Jérémie Gressier (Professeur SUPAERO)

Advisors: Prof. Hans G. Hornung (Caltech)
Dipl.-Ing. Sergey Litvinov (TUM)
Prof. Dr.-Ing. Nikolaus A. Adams (TUM)

Institute of Aerodynamics Technische Universität München
Graduate Aerospace Laboratories California Institute of Technology
2010

Contents

Chapter 1

Introduction

This work aims at the numerical simulation of high enthalpy flow over a porous surface. The subject is of interest, as experiments CITATION NEEDED (Prof. Hornung) have shown that wall porosity can damp acoustic instabilities in hypersonic flows over a slender cone. A numerical simulation of this configuration is desired to furnish data for a detailed analysis of the active mechanism. The simulation method of choice for the first approach to this configuration is called Smoothed Particle Hydrodynamics (SPH).

SPH designates a meshfree method for numerical simulation of particle ensembles (respectively matter that can be modelled as an ensemble of particles as is the case for fluids). Thus, the term particle ensemble not only refers to fluids (as the name of the method would suggest) but also to solids. In fact, nowadays, SPH is employed to simulate problems in different fields, such as elasticity and fracture of solids, liquid flows and gas dynamics [18]. The initial developement of the method dates back to 1977, when Gingold and Monaghan [6] tempted to compute an astrophysical gas dynamics problem with a new, compared to grid based methods more efficient approach. Independently and simultaneously, work in this field was also carried out by Lucy who came up with the same idea [14]. One main advantage of SPH, which also is the reason why SPH was applied first to astrophysical problems, lies in the fact that it is a particle method and therefore does not need any room-fixed grid for the underlying numerical scheme. Instead the base for the calculation of derivatives are discrete points, the particles, which move with the fluid (Lagrangian approach). Dealing with high density variations and huge calculation domains, astrophysical problems, when simulated with grid-based methods, required a considerable effort in terms of mesh-refinemend. For particle methods like SPH whereas, density variations are principally resolved naturally by a more or less dense particle spacing and calculation is only performed at places where there is matter, i.e. particles. Another advantage is the relative ease of the implementation of boundaries. And this is the main reason for the application of SPH in this work, as the structure of the solid boundaries represents a significant part of the problem, when treating porosities.

To be able to perform the simulation, a 2D compressible SPH code was developped in several stages and then validated by means of various test cases. The first code that emerged from this project was a simple 1D compressible SPH code simulating a shock–tube problem. Although this code has no real use for practical applications, it is presented in this thesis with the goal to introduce the reader, who would like to familiarize himself with the implementation of SPH, to this technique.

The 2D code is based on an incompressible multi-phase SPH code, which was developed by [9] and is now in use at the Institute of Aerodynamics of the TUM. This code was adapted in several steps to the needs of the present work, the first step being the implementation of the equations used in the 1D SPH code mentioned above and the adaption of the code to treat inviscid compressible problems. At this stage the code was tested for accuracy and robustness by means of the 1D shock–tube problem. Both a 1D and a 2D particle distribution have been employed to this 1D problem. To conclude the 1D shock–tube tests, a study on the influence of perturbations in the particle distribution has been conducted, again both for 1D and 2D particle distributions. At the same point, the simulation of acoustic wave propagation has been assessed, as acoustic phenomena are essential for the final flow configuration to investigate (see above). In a next step the code was further adapted to incorporate physical viscosity and heat conduction models. The correctness of the implemented viscosity model was tested by analyzing the temporal decay of vortices using the Taylor–Green flow and the thermal conduction model was validated with a standard problem of pure heat conduction. The final test–case, which is a compressibe Courrant flow with hot– and cold–wall boundary conditions, combined all of the previously implemented models. It demonstrated that the combination of all the models would work with the required accuracy. Finally, after the systematic validation of the code as an ensemble and of all its models separately, the actual configuration of interest, the flow through porosities, is simulated.

Chapter 2

Method

2.1 Basic Idea of SPH

To illustrate the basic idea of SPH, it is helpful to first consider the governing equations of fluid dynamics for an inviscid flow, the Euler equations. Remark: As SPH has its origin in Astrophysics, and as many astrophysical problems can be modeled by the Euler equations [13], it is these equations, on which the SPH formalism is based.

The Euler equations have the form [18]

$$\frac{dA(r)}{dt} = f(A(r), \nabla A(r), r) \tag{2.1}$$

where

$$\frac{d(.)}{dt} = \frac{\partial(.)}{\partial t} + v\nabla(.) \tag{2.2}$$

is the substantial derivative.

Thus the rate of change of physical quantity depends on its spatial derivatives. The goal of any method for numerical simulation is to approximate these derivatives by information of a discrete number of points in order for a computer to be able to handle it.

The SPH method is based on the idea of approximating a function $A(x)$ by an integral interpolant

$$A_I(\mathbf{r}) = \int A(\mathbf{r}')\, W(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}' \tag{2.3}$$

where $W(\mathbf{r} - \mathbf{r}', h)$ is the so called kernel or smoothing function and h the smoothing length. If W is the delta function this interpolation is exact, otherwise it represents an approximation. As the theoretically perfect kernel, the delta function, is of no practical use, the kernels employed in practice are functions which tend to the delta function in the limit $h \to 0$. Examples for those kernel functions are a Gaussian kernel or kernels based on Schoenbergs [26]

M_n splines [18]. More details on kernels are given in section (2.2.1) and in the book on SPH published by Liu [13].

The fact of using a kernel different from the delta function constitutes the first of two approximations made by the SPH method. It is often referred to as **kernel approximation** [13]. The second approximation consists of a discretization of the integral expression in equation (2.3), thus changing the integration to a summation over all discretized elements of the domain - the particles. This allows for the determination of $A_I(\mathbf{r})$ by numerical methods. In the SPH literature, this second approximation is commonly called **particle approximation** [13].

With the relation between mass, density and volume for each discretized particle

$$m_i = \rho_i \, V_i, \tag{2.4}$$

applying the particle approximation to equation (2.3) leads to the following expression:

$$A_S(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} W(\mathbf{r} - \mathbf{r}_b, h). \tag{2.5}$$

In the same way, the derivative of the field function $fracdA(x)dx$ can be approximated to give [18] or [13]

$$\nabla A_S(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla W(\mathbf{r} - \mathbf{r_b}, h). \tag{2.6}$$

The disadvantage of the above equation (2.6) is, that for $A(\mathbf{r}) = \mathbf{const}$. the approximated derivative does not vanish. By writing

$$\nabla A = \frac{1}{\Phi}(\nabla \Phi A - A \nabla \Phi), \tag{2.7}$$

where Φ is any differentiable function, this problem can be resolved. In its SPH form equation (2.7) reads as follows:

$$\nabla_a A = \frac{1}{\Phi_a} \sum_b \frac{m_b}{\rho_b} (A_b - A_a) \nabla_a W_{ab}. \tag{2.8}$$

It can be seen that the approximation to the derivative of the field function A is now expressed by a summation over the field function itself and the derivative of the kernel function - which can be determined exactly.

Now the basic SPH formulations are introduced: both the field function itself and its spatial derivatives can be approximated using only information of a discrete number of particles. Hence, the rates of change of the states described by the Euler equations of fluid dynamics, which are of the form corresponding to equation (2.1), can be rewritten using this information. Thus one obtains the SPH formulation of the equations of fluid dynamics.

The detailed derivation of the SPH equations for fluid dynamics will not be shown here, as the interest only lies in describing the principle ideas behind the SPH method. To summarize, these are as follows: a fied-function can be described as an integral interpolation over the function itself and a kernel function, which has to be chosen in an appropriate way (c.f. section (2.2.1)). A further approximation consists in transforming the integral over the domain into a summation over the discrete elements - the particles - constituing this domain. This allows to express both the field function and its spatial derivatives in a form that depends exclusively on discrete values of the field function itself (and of the kernel function and its derivatives, all of which are known). By finally choosing an appropriate kernel function W with compact support, i.e. a finite support domain, where $W = 0$ beyond, the summation over all particles in the calculation domain can be reduced to a summation over those particles within the support domain of the particle in question.

The SPH formulation of the equations of fluid dynamics is obtained by applying the above principles and conducting different transformations to the original differential equations. Depending on the transformations used, the final SPH equations can assume a different form. In the following, a common form for the rates of change of density, velocity and internal energy based on the Euler equations, i.e. without viscosity and heat conduction, is listed [18, 13]:

$$\frac{d\rho_a}{dt} = \sum_b m_b \mathbf{v}_{ab} \nabla_a W_{ab}, \tag{2.9}$$

$$\frac{d\mathbf{v_a}}{dt} = -\sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} \right) \nabla_a W_{ab}, \tag{2.10}$$

$$\frac{de_a}{dt} = -\frac{1}{2} \sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} \right) \mathbf{v}_{ab} \cdot \nabla_a W_{ab}. \tag{2.11}$$

For the detailed derivation and for alternative formulations of these equations refer to [18, 13].

2.2 Features of an SPH code

This section presents some features of a generic SPH code and different variants of these features. Emphasis is placed on these variants that will finally be implemented in one of the two SPH codes designed and modified during this work. Points of interest are a closer description of the kernel function, different methods to perform the search for the interacting particles, as well as the implementation of different boundary conditions in SPH. Furthermore, different equations of state are introduced, depending on the nature of the flow problem to be considered (compressible/incompressible), followed by a passage explaining the need of artificial viscosity. The challenges of implementing a physical viscosity model and a heat conduction model are highlighted thereafter. Subsequently, some numerical integration schemes that have proven to be reliable for

SPH applications are discussed and the two different possibilities to calculate the density are shown. Finally a general structure of a generic SPH code is given.

Note: in this section the above mentionned features are discussed in a general perspective, independent of the particular implementation in a concrete code. The latter will be done in section (2.4.2) for the 1D code and in sections (2.5.2) and (2.5.3) for the 2D code which gives a more complete view.

2.2.1 Kernel function

Some properties of kernel functions, also called smoothing functions, and notions related to them have already been mentionned in section (2.1). In principle, every function can act as kernel function if it respects a set of criteria listed by [13]. The most important ones are (list not exhaustive):

- The smoothing function should vanish outside of a certain area around its center. This property, which is expressed in equation (2.12, is called compact support and the domain where $W \neq 0$ is the **support domain** of the smoothing function. The **support length** κh and the **smoothing length** h are not to be confused.

$$W(x - x') = 0, for|x - x'| > \kappa h \tag{2.12}$$

- The smoothing function should be normalized to 1, which ensures that constants are interpolated exactly:

$$\int W(x - x', h)dx' = 1. \tag{2.13}$$

- The smoothing function should tend towards the delta function as the smoothing length h tends to zero:

$$\lim_{h \to 0} W(x - x', h) = \delta(x - x'). \tag{2.14}$$

  This is to ensure consistency of the integral approximation, which means that the approximate expression tends to the exact expression of the integral interpolant equation for $h \to 0$.

The above list does not contain all necessary SPH kernel properties. Its complete version can be found in [13]. Based on those properties a variety of kernels are used in SPH literature. A very commonly used kernel is the M_4 kernel (cubic spline) based on Schoenberg's M_n-Splines [26].

$$M_4(x) = \begin{cases} \frac{1}{6}[(2-q)^3 - 4(1-q)^3], & \text{for } 0 \leq \text{q} \leq 1 \\ \frac{1}{6}(2-q)^3, & \text{for } 1 \leq \text{q} \leq 2 \\ O, & \text{for q} > 2. \end{cases} \tag{2.15}$$

In the work of [5] a measure of merit for kernels is developed and they came to the result that, at least in 1D, no kernel performes significantly better than the cubic spline. This is why for the 1D SPH code described in section (2.4) the cubic spline kernel was employed.

The unit of the kernel function W is the inverse of the volume or the corresponding 1D/2D equivalent, as can be seen easily from equation (2.40) which is only introduced below.

2.2.2 Neighboring particle search methods

To calculate the derivatives in equations (2.9)-(2.11), principally a summation over all particles has to be performed. However, as we defined kernel functions W with a compact support, i.e. $W = 0$ beyond a certain distance from the particle in question, only particles within this distance, the so called support length (see section (2.2.1)), contribute to the interpolation of the origin particle's values, and thus interact with it. Finding the interacting particles is one of the key features in an SPH implementation, as an effective interaction search method, often also called **nearest neighboring particle (NNP) search** method in the SPH literature, can considerably reduce computation time.

The most basic NNP algorithm is called **all pair search**, and as its name suggests all imaginable particle pairs are tested for interaction. This means that the calculation process consists of a double loop where the outer loop iterates over all particles and defines one of the pair's partners (often refered to as origin). For each of the (origin) particles an inner loop is performed, which iterates again over all particles thus defining the other partner (destination) of the pair. Then, the pair is tested for the interaction condition, meaning that their distance has to be smaller than the support length. In this way, for a calculation domain consisting of N particles, $N * N$ iteration steps and tests have to be executed. The calculation cost of this approach is $O(N^2)$. If the interactions are assumed to occur pairwise, i.e. if there is a spatially constant smoothing–, and therefore support–length, there is a simple but efficient way of reducing computation time. Thus, if a pair P_{ij} is an interaction, the pair P_{ji} is an interaction as well. That way half of the $O(N^2)$ iterations and tests can be eliminated, leading to a complexity of $O(N!)$. This approach is employed in the 1D SPH code as it is easy to implement[13] and the number of particles in this 1D simulation is still reasonably low in order not to have to use more sophisticated algorithms such as are introduced below.
A more effective but also more complicated method is the **linked list algorithm**. This is based on the fact that kernels with compact support are used and thus interacting particles may only be located in the neighborhood of the particle under consideration. The computation domain is divided into cells with cell size $\geq \kappa h$, ideally $= \kappa h$, and the particles are assigned to the cell they are located in [19]. In this way the cell structure serves as a book–keeping device and in case of an interaction calculation with an origin particle i not all the particles $j, j = 1, ..., N$ have to be considered but only those particles in the adjoining cells. Hence, the complexity of the original problem which is of order

$O(N^2)$ can be reduced considerably and even tends to $O(N)$ if the particles in one cell are few. Technically the cells are managed by the implementation of a linked list structure in a computational code. More details on this are given in section (2.5), as this algorithm is implemented in the 2D SPH code, and besides in [20, 8].

However, the method has a drawback if used with a spatially variable smoothing length. In this case the cell size is not ideally adapted to every particle and the algorithm looses efficiency.
Moreover, different other NPP techniques exist which, here, shall not be introduced in detail. One class of these are called **tree search algorithms** [7], which are especially robust and efficient for problems with variable smoothing length [13].

2.2.3 Boundary condition implementation

For particles at the edge of the calculation domain, the summation interpolation does not give exact results, as the integral, respectively the sum, is truncated due to the simple fact that outside the domain there are no particles [13]. This effect is called the **particle deficiency problem**.[13]. In cases where the region of interest is far from the edge of the domain and where erroneous information from the edges do not propagate to the region of interest during the time in consideration, there is no need to take this problem into account. The 1D shock-tube simulation described in sections (3.1) and (3.2) is an example of such a case. But for other applications the edges play an important role, or the calculation domain has to be continued periodically. In these cases it is necessary to define specific boundary conditions. This can be accomplished by a special boundary treatment which often includes so–called virtual particles or **ghost particles** [13]. The first ones to come up with this idea were [12]. Virtual particles are used to model solid walls by placing them inside the solid body, mirrored at the surface. For each real particle whose support domain interferes with the wall, one such particle is built. Virtual particles have the same properties as their real counterpart except for the velocity. Depending on the type of boundary condition they are supposed to represent, different velocities are assigned to them [9].

No–slip velocity boundary condition:

$$\mathbf{v}_{\mathbf{virtual}} = 2\mathbf{v}_{\mathbf{wall}} - \mathbf{v}_{\mathbf{real}} \tag{2.16}$$

Free slip velocity boundary condition:

$$\mathbf{v}_{\mathbf{virtual}}^{\mathbf{tangential}} = \mathbf{v}_{\mathbf{real}}^{\mathbf{tangential}} \tag{2.17}$$

2.2.4 Equation of state

The equation of state relates the pressure p to the density ρ and in general also the internal energy e, or equivalent. An equation of state is needed for the solution of compressible flow problems, as the energy equation is not decoupled from the mass and momentum equation unlike for incompressibly treated flows: for the latter, the density ρ is constant and therefore not a variable of the problem, leaving the pressure p and the velocity vector v as unknowns of the problem. These, in the general 3D case, four variables are faced by the four equations of mass and momentum conservation which makes the problem solvable. If the density becomes an unknown of the problem as well, further equations have to be consulted to solve the problem.

For the application of SPH to gas dynamics, the assumption of an ideal gas is often appropriate. In this case the corresponding equation of state is

$$p = (\gamma - 1)\rho e. \tag{2.18}$$

The sound speed, which is needed to calculate the artificial viscosity (see equation (2.22)), can be formulated for an ideal gas as:

$$c = \sqrt{\gamma R T} \tag{2.19}$$

or, with $c_v T = e$, $R = c_p - c_v$ and $\gamma = \frac{c_p}{c_v}$,

$$c = \sqrt{\gamma(\gamma - 1)e}. \tag{2.20}$$

Different equations of state need to be used in other media. For example, according to Monaghan [17, 18], even liquids, which are commonly approximated as incompressible (such as water) are treated as weakly compressibly is SPH. This requires a corresponding equation of state. As this equation of state does not depend on the temperature though, the energy equation is decoupled from the rest of the equations and there is no need to solve it - exactly as it is known for the truly incompressible case.

2.2.5 Artificial Viscosity

Another point that has to be taken into account for SPH simulations, especially if shock waves are present, is artificial viscosity [18]. This prevents particles from penetrating one into another in zones with high compression rates and also generally stabilizes a numerical algorithm. Artificial viscosity for numerical simulations of shock phenomena in general was first introduced by [30]. They suggested to introduce an artificial dissipation to give the shock a finite thickness to allow shocks to be calculated numerically (in the cited example with a finite difference scheme) without introduction of shock jump conditions. The first use of artificial viscosity specifically for the mesh–free SPH goes back to Lucy[14]. He suggested an artificial bulk viscosity to prevent a slow build–up of acoustic energy by integration errors in zones with low density respectively low particle presence. Finally, the form of artificial viscosity most commonly used in SPH literature [13] is the one proposed by Monaghan and Gingold [19]. They stated that the use of one of the artificial viscosities cited above (Neumann Richtmyer (adapted to SPH) or bulk viscosity) on shock problems would lead to either excessive particle oscillation or excessive smearing of the shock front. The expression for the Monaghan artificial viscosity Π_{ab}, which is added to the pressure term of equation (2.10), is - in the formulation of [15] which is equivalent to the one given in [18], but more appropriate for implementation as parts of it are later needed for automatic timestep control-

$$\Pi_{ab} = \frac{-\alpha c_{ab}\mu_{ab} + \beta\mu_{ab}^2}{\rho_{ab}} \tag{2.21}$$

where

$$\mu_{ab} = \frac{h_{ab}v_{ab}r_{ab}}{r_{ab}^2 + \epsilon h_{ab}^2} \tag{2.22}$$

v_{ab} stands for $|\mathbf{v_a}-\mathbf{v_b}|$, r_{ab} for $|\mathbf{r_a}-\mathbf{r_b}|$, c_{ab} is the averaged sound speed of the two particles in question $1/2(c_a + c_b)$. The same applies for $\rho_{ab} = 1/2(\rho_a + \rho_b)$ and $h_{ab} = (1/2(h_a + h_b))$. Remark: the use of the averaged smoothing length h_{ab} is a way to symmetrize particle interactions when h is spatially variable. In this case a particle a could be located in the support domain of a particle b but not vice versa. This would lead to an asymmetric mutual contribution of forces. To avoid this, using the averaged smoothing length between two particles is an appropriate remedy [13].

Commom choices for the two paramters figuring in equation (2.21) are $\alpha = 1$ and $\beta = 2$ [15]. The epsilon–term in equation (2.22) serves to prevent singularity if $r_{ab} = 0$ and ϵ should be chosen as $\epsilon = 0.01$ [15]. Including artificial viscosity, the SPH form of the Euler momentum equation (2.10) becomes [18]

$$\frac{\mathbf{dv_a}}{dt} = -\sum m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Pi_{ab}\right) \nabla_a W_{ab}. \tag{2.23}$$

The artificial viscosity must also be reflected in the energy equation, as viscosity transfers energy from kinetic to thermal [18]. One formulation of the energy equation which is convenient for implementation in the 1D SPH code (as a big part of the needed expression has already been calculated for equation (2.23) can be found in [13].

$$\frac{de_a}{dt} = -\frac{1}{2}\sum m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2}\right) \mathbf{v}_{ab} \cdot \nabla_a W_{ab}. \tag{2.24}$$

2.2.6 Physical Viscosity Implementation

SPH was originally developped for astrophysical use, where the Euler equations are generally an appropriate description for the typical phenomena [13]. Thus, the integration of a phsyical viscosity description in an SPH scheme was only conceived belatedly. At first sight, the most obvious way of doing so is certainly to approximate the appearing second derivatives directly by the SPH formalism, leading to second derivatives of the smoothing function. However, these second derivatives of the kernel function turned out to be very sensitive to particle disorder [2, 16], which is a major drawback and makes this formulation a far from ideal solution to the problem. Therefore, different ways of implementing physical viscoyity in SPH have evoled, each with advantages and disadvantages. A good overview was found in [1]. The basic ideas of the different formulations and their respective domain of application is briefly summarized here.

work in progress...

2.2.7 Heat conduction Implementation

Heat conduction has to be taken into account for the final application of the code, as the high enthalpy flow in the porosity will essentially be influenced by the relatively cold walls. Modelling heat conduction in SPH faces the same issues already encountered for the physical viscosity in section (2.2.6). Both being diffusion phenomena, they figure with a second derivative in the governing equations of fluid dynamics. A very robust approach of formulating the heat conduction term in SPH seems to be the one of Cleary and Monaghan [3]. They conducted extensive tests with different configurations including variable thermal conductivity k and disordered particles for which the method performed very satisfactory.

For the heat conduction term to be taken into account in the energy equation (2.11), they propose the following form

$$\frac{1}{\rho}\nabla\cdot(k\nabla T) = \sum_b \frac{4m_b}{\rho_a\rho_b}\frac{k_a k_b}{k_a+k_b}T_{ab}F_{ab}. \tag{2.25}$$

The temperature is denoted T, the thermal conductivity k and as the kernel is symmetrical, one can write

$$\nabla W(r) = rF(r) \tag{2.26}$$

2.2.8 Numerical integration scheme

As far as the time integration of the SPH equations is concerned, any stable time stepping algorithm for ordinary differential equations may be used [18]. However, for SPH simulations, the conservation properties (linear and angular momentum) of the integrator are more important than its order [18].

Thus, a standard leap–frog scheme is one possibility to perform the integration in time. The initial time step is done according to the following algorithm where $\mathbf{S}$ is a vector with components ρ, v, e.

For $n = 0$:

$$\mathbf{S_{\frac{1}{2}}} = \mathbf{S_0} + \frac{1}{2}dt\left(\frac{d\mathbf{S}}{dt}\right)_0 \tag{2.27}$$

$$\mathbf{x_1} = \mathbf{x_0} + \mathbf{v_{\frac{1}{2}}}\,dt. \tag{2.28}$$

For any further time step $n \geq 1$:

$$\mathbf{S_n} = \mathbf{S_{n-\frac{1}{2}}} + \frac{1}{2}dt\left(\frac{d\mathbf{S}}{dt}\right)_{n-1} \rightarrow \left(\frac{d\mathbf{S}}{dt}\right)_n \tag{2.29}$$

$$\mathbf{S_{n+\frac{1}{2}}} = \mathbf{S_{n-\frac{1}{2}}} + dt\left(\frac{d\mathbf{S}}{dt}\right)_n \tag{2.30}$$

$$\mathbf{x_1} = \mathbf{x_0} + \mathbf{v_{\frac{1}{2}}}\,dt. \tag{2.31}$$

Another possibility for time integration is a predictor corrector scheme, as used by [10]. The 2D SPH code structure being based on their work, this integration scheme is also implemented in this program.

In the predictor step a guess $\hat{\mathbf{S}}$ for the value $\mathbf{S}$ at $n+1$ is made by simply applying a normal Euler step:

$$\hat{\mathbf{S}}_{\mathbf{n+1}} = \mathbf{S_n} + dt\left(\frac{d\mathbf{S}}{dt}\right)_n \tag{2.32}$$

$$\mathbf{x_{n+1}} = \mathbf{x_n} + \mathbf{v_n}\,dt. \tag{2.33}$$

This guess is then corrected by a central difference scheme to update the values at $n+1$:

$$\mathbf{S_{n+1}} = \mathbf{S_n} + dt\left(\frac{d\mathbf{S}}{dt}\right)_{n+\frac{1}{2}}. \tag{2.34}$$

The values at $n+\frac{1}{2}$ needed to calculate $\left(\frac{d\mathbf{S}}{dt}\right)_{n+\frac{1}{2}}$ are obtained by taking the mean value of S_n and $\hat{\mathbf{S}}_{\mathbf{n+1}}$:

$$\mathbf{S_{n+\frac{1}{2}}} = \frac{1}{2}(\mathbf{S_n} + \hat{\mathbf{S}}_{\mathbf{n+1}}). \tag{2.35}$$

2.2.9 Choice of time step

As any other numerical method, SPH gets instable, if the time step dt for the numerical integration is not choosen correctly (i.e. small enough). This fact was first examined by Courant, Friedrichs and Lewy [4] who studied the convergence behaviour of discretized partial differential equations, i.e difference equations. They found out that for hyperbolic difference equations, the spatial and temporal discretization steps need to meet a certain condition in order for the difference equation to converge to its corresponding differential equation. This conditon is generally known as the Courant-Friedrichs-Lewy (CFL) condition, which gives a necessary criterion for the stability of numerical difference schemes. In 1D it can be formulated as

$$\frac{u\delta t}{\delta x} \leq C \tag{2.36}$$

C being a constant. To principally make its meaning clear, it can be interpreted as follows: the numerically possible propagation - in terms of distance - of any information during one timestep, which is $\propto \delta x$, must be superior to the physical propagation of phenomena of the underlying problem, which is $\propto u\delta$, u being a propagation velocity charactieristic to the problem. If this is not given, the simulation can not correctly reproduce the physical behaviour.

The principle of the CFL-Condition can also be applied to SPH [21], which is not a finite difference scheme, but where a characteristic numerical propagation can still be described using the smoothing length as length scale: in one timestep information can propagate at maximum a distance of κh, which corresponds to the support length. On the other hand, the characteristic propagation speed for the physical problem is the soundspeed c. In addition to the Courant-condition, two more criteria that influence the timestep selection must be considered [15], one based on the force per unit mass f and the other based on the viscosity, be it physical or artificial.

These three conditions can be written as [21, 15]

$$dt_f = \min_i \left(\frac{h}{f_i} \right) = min \left(\frac{h}{\frac{du}{dt}} \right) \tag{2.37}$$

$$dt_{cv} = \min_i \left(\frac{h}{c_i + 0.6(\alpha c_i + \beta \max_j \mu_{ij})} \right) \tag{2.38}$$

where equation (2.37) defines the maximum admissible timestep regarding the forces and equation (2.38) the maximum admissible timestep for the combined Courant and viscosity criterion. The indices i, j go over all particles, the parameters α and β are the same as for the artificial viscosity (see section (2.2.5)) and μ_{ij} is the viscosity. In the case of an exclusively artificial viscosity, μ_{ij} is given by equation (2.22).

The applicable time step for the simulation corresponds to the minimum of the above two, including a security factor

$$dt = \min(dt_f, dt_{cv}) \tag{2.39}$$

This can be implemented in an SPH program and so the applicable timestep is automatically adapted to the given situation at each individual timestep. Note concerning the implementation: in equation (2.37), there can be a singularity in the denominator if the mass–specific force per particle, i.e. its acceleration, is zero. Although this does not affect the validity of the above equations, there are issues when implemented. To prevent this singularity, a very small number (e.g. $1E^{-35}$) is added to the denominator at implementation.

2.2.10 Summation Density and Continuity Density approach

One way to update the density with the SPH method is simply to integrate the density change rate (equation (2.9) for each time step, just like it is done for velocity and internal energy. Because of the character of the SPH formulation, it is also possible to obtain the density in another way. Taking the interpolation equation (2.5) and supposing A to be the density ρ gives an expression for the interpolation - the smoothing - of the density of a particle a by summing over all particles b within the support domain [18].

$$\rho(r) = \sum_b m_b \, W(r - r_b, h). \tag{2.40}$$

The first method where density is obtained by integrating equation (2.9) is refered to as the **continuity density approach**, as the density is obtained by integrating the SPH form of the continuity equation. The second method, where density is obtained by summation, or smoothing, is called the **summation density approach** [13]. Both methods are commonly used in SPH codes, and they are often even both implemented in one same code.

2.2.11 Basic structure of an SPH code

Any implementation of an SPH code possesses a certain basic structure defined by the nature of the SPH equations. Figure (2.1) together with the description in this paragraph shall show the interdependence of the constituent parts of an SPH code in principle. The actual control–flow of a specific SPH code may differ in detail from figure (2.1) and can be seen from the more detailed flow diagrams and descriptions for the specific code.

Remark: the structure described in the following paragraph applies to any (serial, not parallel) SPH code (no matter if compressible/incompressible or viscous/inviscid). The equations referred to, however, are the SPH form of the Euler equations and thus are only valid for inviscid problems.

Figure (2.1) shows that the first essential part of an SPH program is the setting up of the initial conditions in form of an initial particle distribution, where each particle has the desired properties. Since the choice of the quantities given as initial conditions is not being unique, there are several possibilities in defining them. For example giving m and ρ defines the particle spacing, which corresponds to the volume, and giving the spacing and ρ fixes m. Which way is selected for a specific program is explained in the corresponding section. Thereafter the program enters the main computation loop and the operations executed therein are repeated for each time step. Among those operations is the creation of the virtual boundary particles (section (2.2.3), not shown in figure (2.1)) which then need to be included in the nearest neighbouring particle search (section (2.2.2)). After having determined all the interactions for all particles, the smoothing function and its derivative can be computed for each interaction pair. Then, depending on the choice of density calculation mode (section (2.2.10)), the program sequence differs slightly. With the summation density approach, the density is obtained by summation (equation (2.40)), and, once density is calculated, the forces on the particles may be determined. Those forces allow for the computation of the rate of change of momentum. Remark: the calculation of forces and rate of chage of momentum can be considered as one single step, all included in equation (2.10). The rate of change of energy is determined from equation (2.11). For the continuity density approach, the calculation of forces is conducted the same way as before. But the rate of change of density has to be determined as well according to equation (2.9), as density will be obtained by integration. Once all necessary change rates are known, the values can be updated to the next time step, see section (2.2.8) and the loop can be restarted. After the actual simulation is finished, an output module is in charge of making data available in a form that is exploitable by the user.

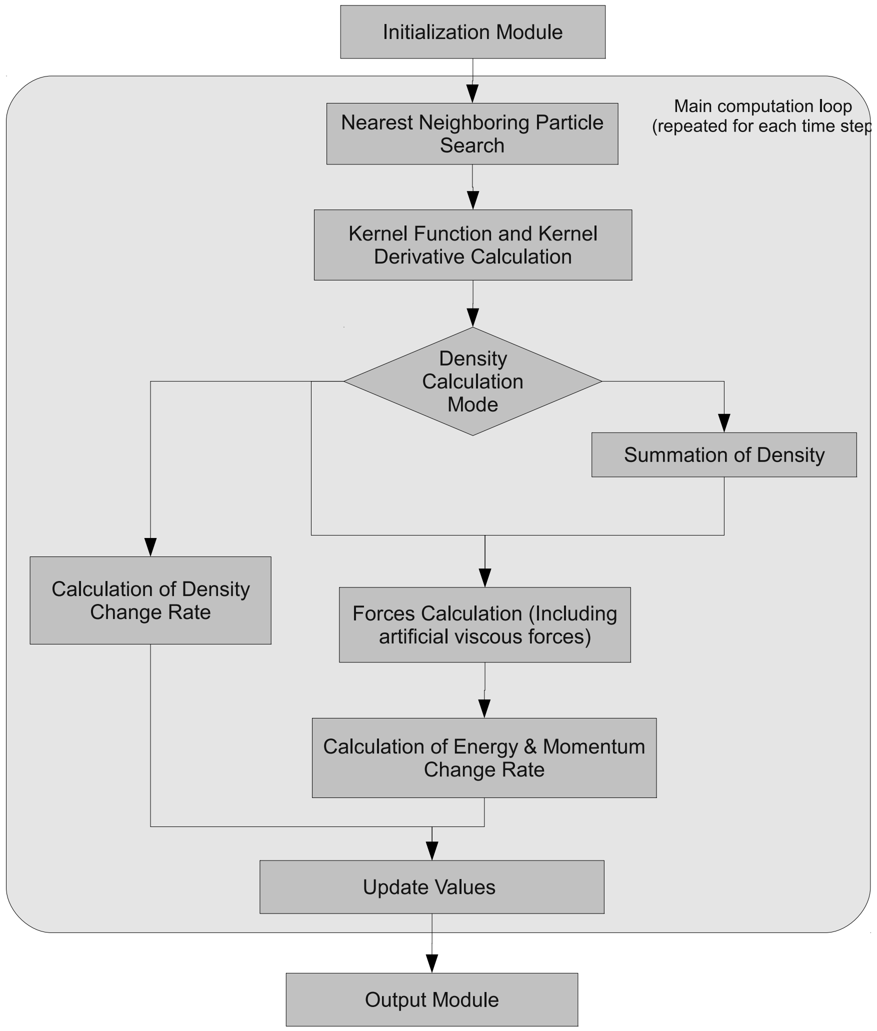

Initialization Module
Main computation loop
(repeated for each time step
Nearest Neighboring Particle Search
Kernel Function and Kernel Derivative Calculation
Density Calculation Mode
Summation of Density
Calculation of Density Change Rate
Forces Calculation (Including artificial viscous forces)
Calculation of Energy & Momentum Change Rate
Update Values
Output Module

Figure 2.1: Basic structure of an SPH code (adapted from [13])

2.3 Presentation of test cases

This section introduces the different test cases, which are used to validate the developped code at the different stages. The general problem setup is presented as well as analytical solutions in case they exist. Otherwise a numerical reference solution is provided.

The reason why this section is placed already here is, that the test problems are sometimes referred to from the code section (2.4),(2.5) and it is preferable if they are introduced before.

Details on initializing the flow–field, running the simulations and post–processing are not given in this section but only in section (2.6), after having introduced the numerical codes. The present section merely povides the basis by introducing equations and the general ideas necessary to perform the actual simulation and the post-processing.

2.3.1 1D shock–tube

As an exact solution is known, the 1D shock-tube is a very common test–case for compressible numerical techniques, especially in the form used by Sod [28].

The problem consists of two initially separated (e.g. by a diaphragm) gas–filled tube sections, both in general with different properties, at least different pressure. When at the instant t_0 (initial situation shown in fihure (2.2)) the separation is removed, a shockwave is propagating into the low–pressure area and a rarefaction is traveling into the high pressure area, both adjusting the pressure to the same intermediate value. This means that at a time $t_1 > t_0$ the situation can be presented as in figure (2.3) where the nomenclature from Anderson [11] is adopted.

In figure (2.3), areas 2 and 3 are separated by the so called contact discontinuity, which is the fluid interface generated by the removal of the diaphragm. While in general, the states and properties of the fluids in 2 and 3 are different, pressure p and velocity u at both sides of the discontinuity have to be the same:

$$\begin{aligned} p_2 &= p_3 \\ u_3 &= u_2 \end{aligned} \tag{2.41}$$

The analytical solution to the shock–tube problem can be found using the shock relations and the isentropical relations and is taken from Anderson [11] as well. It will be presented here, as the post processing program, which is described below, is based on this solution.

First the unknown quantities in areas 2 and 3 are calculated (states 1 and 4 are completely known). Then the location of all area boundaries is determined at a certain instant $t_1 > t_0$. Finally the properties inside the rarefaction (located between area 3 and 4) can be obtained.

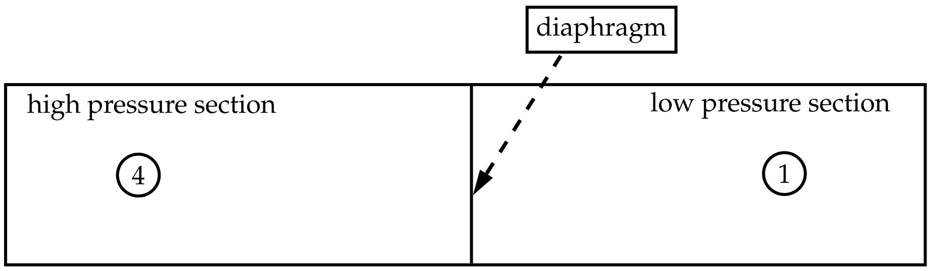


Figure 2.2: Shock–tube at initial time t_0

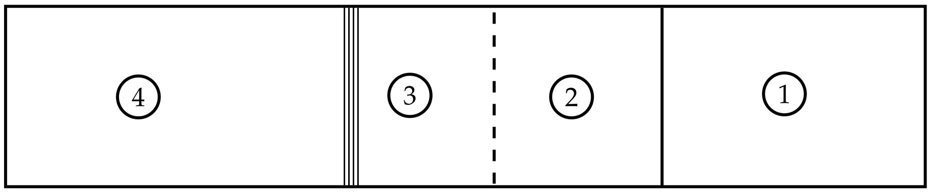


Figure 2.3: Shock–tube at time $t_1 > t_0$

Determination of quantities in areas 2 and 3

With the relation

$$\frac{p_4}{p_1} = \frac{p_2}{p_1}\left(1 - \frac{(\gamma_4 - 1)\frac{c_1}{c_4}\left(\frac{p_2}{p_1} - 1\right)}{\sqrt{2\gamma_1\left(2\gamma_1 + (\gamma_1 + 1)\left(\frac{p_2}{p_1} - 1\right)\right)}}\right)^{\left(\frac{-2\gamma_4}{\gamma_4 - 1}\right)} \tag{2.42}$$

the unknown pressure p_2 can be determined iteratively, where the sound speed for an ideal gas is calculated according to equation (2.19).

For the further proceeding, the ratio of specific heats γ is assumed constant (i.e. same gas in 1 and 4 and besides, no variation of γ with temperature)

$$\gamma_4 = \gamma_3 = \gamma_2 = \gamma_1 = const \tag{2.43}$$

The density in area 2 can be determined from state 1 with the shock relations

$$\rho_2 = \rho_1 \frac{1 + \frac{\gamma+1}{\gamma-1}\frac{p_2}{p_1}}{\frac{\gamma+1}{\gamma-1} + \frac{p_2}{p_1}} \tag{2.44}$$

Applying the isentropical relations from state 4, one obtains the density in area 3

$$\rho_3 = \left(\frac{p_3}{p_4}\right)^{\frac{1}{\gamma}} \rho_4 \tag{2.45}$$

The internal energy in areas 2 and 3 is obtained by the ideal gas equation of state (2.18).

The velocity of the shock traveling into area 1 is

$$W = c_1 \sqrt{\frac{\gamma+1}{2\gamma}\left(\frac{p_2}{p_1}-1\right)+1} \tag{2.46}$$

With the applicaton of the continuity equation, the velocity induced in area 2 u_2 by the shock is

$$u_2 = W\left(1-\frac{\rho_1}{\rho_2}\right) \tag{2.47}$$

According to equation 2.41, the velocity in area 3 u_3 is the same.

Determination of the area boundaries at a certain instant

As geometrical information comes into play, the position of the diaphragm in the space has to be indicated and is assumed to be $x_{dia} = 0$ and x is counted positively towards the right, i.e. in direction of the low pressure area (area 1)). The following positions have to be determined (see also figure 2.3:

(a) begin rarefaction (delimits area 4)
As the begin of the rarefaction propagates into area 4 with the corresponding (local) soundspeed, its position is

$$x_{rar,beg} = -c_4 t \tag{2.48}$$

(b) end rarefaction (delimits area 3)
The end of the rarefaction propagates with the corresponding (local) soundspeed relative to the fluid in 3 which itself has a velocity u_3. The desired position is therefore

$$x_{rar,end} = (u_3 - c_3)t \tag{2.49}$$

(c) contact discontinuity
The contact discontinuity moves with the velocity u_3 and therefore

$$x_{CD} = u_3 t \tag{2.50}$$

(d) shock
The expression for the propagation velocity is given by equation (2.46) and therefore

$$x_{shock} = Wt \tag{2.51}$$

Determination of the properties inside the rarefaction

Inside the rarefaction the flow evolves in an isentropic manner, which means the isentropic relations can be employed. Furthermore, the theory of characteristics gives the velocity at a certain point inside the rarefaction. Again all equations are taken from Anderson [11] and the coordinate system remains unchanged: $x = 0$ corresponds to diaphragm position and therefore is origin of the rarefaction.

To obtain the velocity inside the rarefactions, the following equation is used

$$u_{rar} = \frac{2}{\gamma + 1} \left(c_4 + \frac{x}{t} \right) \tag{2.52}$$

The density at a certain point inside the rarefaction is calculated with

$$\rho = \rho_4 \left(1 - \frac{\gamma - 1}{2} \frac{u}{a_4} \right)^{\frac{2}{\gamma - 1}} \tag{2.53}$$

and for the pressure, the isentropic relation in the following form is used

$$p = p_4 \left(\frac{\rho}{\rho_4} \right)^{\gamma} \tag{2.54}$$

Finally, the internal energy is computed with the equation of state (2.18).

2.3.2 Acoustic wave propagation

Acoustic phenomena play an important role for the actual problem to be investigated within this project (see chapter (1)). It is therefore necessary to get an idea on how the code reproduces these phenomena. A crucial issue in this context is the damping of oscillating or propagating waves due to numerical dissipation: many numerical codes have an intrinsic dissipation, introduced by the numerical scheme, even when the actually implemented equations are inviscid. This numerical dissipation acts like a real viscous dissipation and diminuishes the kinetic energy of the flow. The comparison of the simulation results against the following test cases, whose theoretical solution is always obtained assuming an inviscid flow (i.e. no dissipation), aim at quantifying this damping. A further point to be investigated by means of these test cases is the capability of the code to exactly reproduce the right periodical time T of the propagating/oscillating wave.

In order to quantify the effect of viscosity in the simulation results (no matter what nature it is), a damping factor is introduced as the relative change in amplitude A of two consecutive periods i and $i + 1$

$$f_d = \frac{A_i - A_{i+1}}{A_i} \tag{2.55}$$

where the amplitude A may be measured either in terms of pressure p or velocity u.

For the case of a damping force that is proportional to the velocity, the damping factor is constant. This fact is important with regard to the artificial viscosity that can be included in the code (see section (2.2.5)). A closer look at the formulation for the artificial viscosity (equation (2.21)) yields

$$\Pi_{ab} \propto v_{ab} \tag{2.56}$$

. This above statement is made assuming ν from equatio (2.21) constant.

2.3.2.1 1D stationnary oscillating wave

The configuration of an oscillating wave is taken from [18], where it is used to study the dispersion relation for an infinite one–dimensional gas. In the present context though, the same configuration is judged suitable for the study of damping and evolution of T.

The sinusoidal oscillation of wavelength λ is initialized by a velocity profile with amplitude of 5% of sound speed

$$u(x) = 0.05c \sin\left(\frac{2\pi x}{\lambda}\right) \tag{2.57}$$

in the domain $0 \leq x \leq \lambda$ with periodical boundary conditions on both sides.

For a simple harmonic wave, the theoretical periodical time T is given as [29]

$$T = \frac{\lambda}{c} \tag{2.58}$$

and serves as a reference for the comparison with the simulation results.

2.3.2.2 1D infinite traveling wave

This test case is similar to the above one with the difference that now the wave travels in one direction through the space. A theoretical description for this kind of wave can be found using the wave equation for a velocity potential. This is a linearized equation and therefore valid only for small disturbances, which for the present application is justified. In Stewart [29], the expressions for the fluctuation (denoted by δ) of the desired quantities around their mean values (denoted by the subscript 0) are directly given and read as follows:

- velocity evolution

$$\delta u(x,t) = kA' \sin(k(ct-x)) \tag{2.59}$$

- density evolution

$$\delta\rho(x,t) = \rho_0 \frac{k}{c} A' \sin(k(ct-x)) \tag{2.60}$$

- pressure evolution

$$\delta p(x,t) = kc\rho_0 A' \sin(k(ct-x)) \tag{2.61}$$

- evolution of the displacement $\xi = x - x_0$

$$\xi(x,t) = -\frac{A'}{c}\cos(k(ct-x)) = -\frac{A'}{c}\sin\left(k\left(ct - x + \frac{\pi}{2k}\right)\right) \tag{2.62}$$

A' denotes the amplitude of the underlying velocity potential and therefore determines the amplitude of the other quantities as well. It can be any random constant with the restriction that the disturbances in ρ and p remain small compared to their mean value (linear theory). The variable k is the wavenumber and defined as $k = \frac{2\pi}{\lambda}$.

Furthermore one can note that the displacement is $\pi/2$ or $90°$ in phase ahead of the other quantities. This fact will be important when thinking about the initialization of a finite propagating wave (single wave train).

For the initialization of the flowfield at the instant $t = 0$, the above equations (2.59)-(2.62) will be employed (see section (2.6)).

2.3.2.3 1D wave train with reflection and interference

For the testing of reflection and interference, the problem as described below is simulated.

But first, some thoughts on the initialization of a single 1D wave train are presented. Goal is to initialize a 1D wave train in one part of the domain while the rest shall be initialized with unperturbed flow. For this to make any sense, the size of the domain has of course to be superior to the wavelength λ of the wave train.

First attempts to initialize the single wave train using the equations for a continuous (infinite) traveling wave (2.59)-(2.62) have turned out to be problematic due to the phase shift between ρ, p, u on the one hand and the displacement ξ on the other hand. As the wave does not continue infinitely but is simply cut off after one wavelength, there will always be a discontinuity in one of the quantities.

If, for example, the initialization is done with a smooth transition of ρ, p, u from the unperturbed flow, there is a discontinuity in the displacement, and vice versa. This discontunuity (be it in ρ, p, u or in the displacement) constitutes a new disturbance which again is origin of new waves. So, no neat wave train could be created this way. Thus, a different way to generate the wave had to be found and it turned out that the periodic movement of a wall boundary was an appropriate mean to do so.

Now that single wave trains can be produced, the test case is introduced: A domain with size l_{dom} and limited by two solid walls (at $x = 0$ and $x = l_{dom}$) is considered. The left hand wall at $x = 0$ has the ability to perform a movement according to a sinusoidal velocity profile, when desired. The periodical time T of the wave train is chosen in a way that its wavelegth is $\lambda = l_{dom}/4$. The condition on T is therefore

$$T = \frac{l_{dom}}{4c_0} \tag{2.63}$$

The following passage explains the temporal course of the theoretically exact test problem

- At $t_0 = 0$ the left hand side wall begins to move according to

$$u_{wall} = u_{max} \sin(\omega t) \tag{2.64}$$

  where

$$\omega = 2 * \Pi / T u_{max} = 0.05 c_0 \tag{2.65}$$

- At $t_1 = T$ the wall stops moving, meaning that exactly one wave train has been generated, which now travels through the domain.
- At $t_2 = l_{dom}/c_0$ the first part of the wave train hits the right hand side wall and starts being reflected. Simultaniously, the LHS wall starts moving again according to same equation as above, generating a new wave.
- At $t_3 = t_2 + T$ the LHS wall stops moving and the generation of the second wave is finished. The first wave is now completely reflected from the RHS wall.
- At $t_4 = t_3 + T$ the two waves begin to interfere. (Both wave trains reach the middle of the domain)
- At $t_5 = t_4 + T/2$ the interference of the waves is at maximum, i.e. both waves interfere along their entire wavelength.
- At $t_6 = t_4 + T$ the wave interference is over.
- At $t_7 = t_6 + T$ both waves hit the respective walls. The test problem ends at this instant.

2.3.3 Taylor–Green flow

WORK IN PROGRESS

2.3.4 Pure heat–conduction

WORK IN PROGRESS

2.3.5 compressible Couette–flow

WORK IN PROGRESS

2.4 1D SPH code

2.4.1 Short overview

This code has a relatively simple structure, yet it features all the components a more sophisticated SPH computational program would incorporate. Both the summation density approach and the continuity density approach are implemented (see section (2.2.10)). By computing a 1D shock–tube problem, the code is used to verify the the basic compressible SPH Euler equations which are then implemented in the more complex 2D program.

As the analytical solution for this problem is well known, the shock–tube is a very common test case for 1D compressible codes [28] The problem setup and the results obtained are discussed in the corresponding section.

2.4.2 Components and structure

2.4.2.1 The data structures

To facilitate the implementation and the handling of the code, two data structures are defined.

The `parameters` data structure

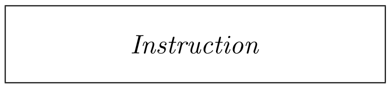
This data structure is defined in the header file `parameters.h` and contains all parameters needed to set up the problem. Among those parameters are geometrical values, the initial conditions, simulation length and time step as well as parameters defining the fluid properties.

The `simulation` data structure

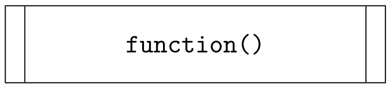
Defined in the file `simulation.h`, this structure is made up of all variables that change in the course of the simulation. To begin with, these are the values of interest like position, velocity, density, pressure, energy, etc.. Furthermore there are variables needed as intermediate results, such as the smoothing length and its derivative for example. Most of the variables contained in the simulation data structure are needed one per particle. Therefore those variables are defined as vectors with a size that corresponds to the number of particles in the system.

2.4.2.2 The functions

As the size of the 1D SPH program is not too big, it admits of the description of every single function used. The textual description is not complete but only comprises particularities and features that are considered worth mentionning. A Nassi-Schneidermann-Diagram (NSD) [22] complements each description and shows the control flow for each function as well as the interdependence of the individual functions. For the sake of a clear illustration, the NSD does not display every loop of the program; frequently, loops over all particles and the interaction list are ommitted (the instruction 'calculate ... for each particle' for



(a) normal Instruction



(b) function call

Figure 2.4: The NSD process symbols

example means the implementation of a loop over all particles, i.e., all components of the corresponding variable's vector). Consulting the comments in the source code is recommended if a more detailed understanding is necessary or desired.

Nassi-Schneidermann-Diagram - a short introduction

A NSD is employed in this section as it allows for a more compact description of the code structure within limited space than a conventional flow diagram and, in addition, offers a range of further advantages [22]. It is read from top to bottom, with different box symbols representing different kinds of instructions or control elements. The symbols used during the description of the functions below are the following ones:

The process symbol This symbol represents simple instructions for which during their execution no analysis has to be made. Examples are assignments or inputs/outputs. A variation of the conventional process box is the symbol for the call of an external function.

The iteration symbol The shown iteration symbol represents loops, where the condition is tested each time before entering the body. This control struct can be implemented by a while loop or a for loop. An example is shown in figure (2.4.2.2).

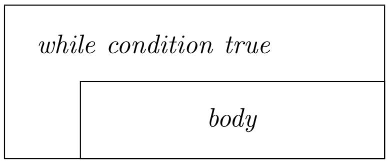
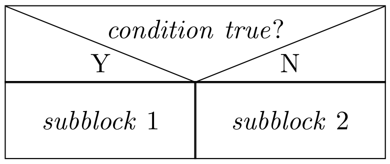
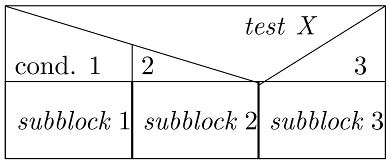


Figure 2.5: The NSD iteration symbol

The decision symbols Control structures that depending on a condition enter different branches are represented by the decision symbols. A binary decision symbol corresponding to an if-else statement in programming languages is shown in (FIGURE...). For multi branch control structs, such as if-ifelse-...-else or switch-case, (FIGURE...) provides the corresponding symbol (in the example: 3 branches).



(a) binary decision



(b) multiple decision

Figure 2.6: The NSD decision symbols

Remark: The NSD flowchart language provides the possibility to describe programs at their level of implementation. In the following section however, it is used in a more abstract way and instructions or conditions are verbalized.

The Main function

As its name suggests, `main.cpp` is the main function of the program. It is entered upon execution of the program, declares and initializes the data structures and calls the functions needed. Its structure is given in figure (2.4.2.2).

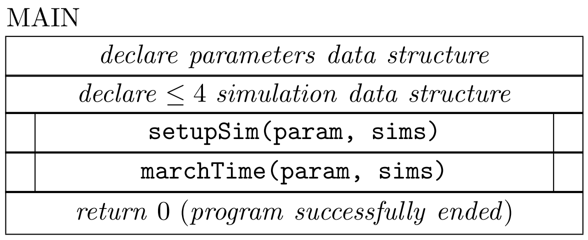
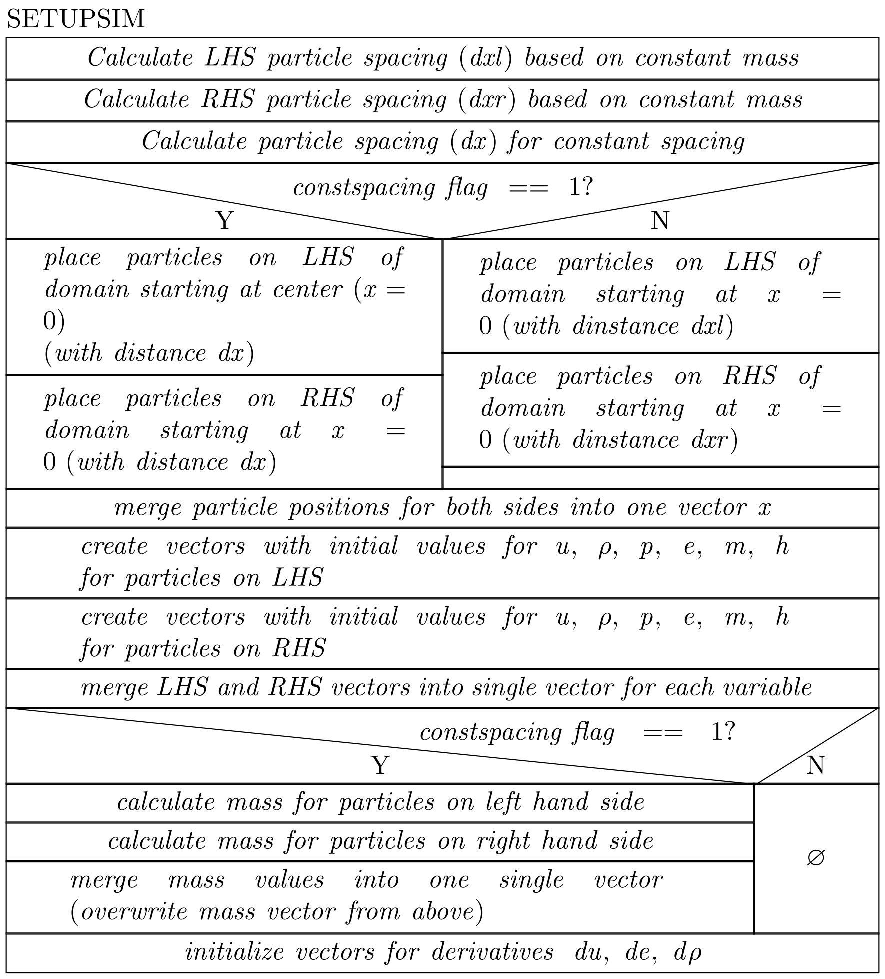

MAIN
declare parameters data structure
declare ≤ 4 simulation data structure
setupSim(param, sims)
marchTime(param, sims)
return 0 (program successfully ended)

Figure 2.7: The structure of the main–function

The setupSim function

Implemented in the file `setupSim.cpp` with the corresponding header `setupSim.h`, this function represents the 'initialization module' shown in figure (2.1). More precisely, it first sets up the initial particle distribution for the shock–tube problem: depending on the user's selection made by setting the flag 'constspacing', which is a member of the `parameters` data structure, this is done in two different ways: One possibility is to have an initial particle distribution with the same spacing in the high density area of the shock-tube as in the low density area (flag `constspacing = true`). In this case though, the mass of the particles has then to be adapted to be coherent with the spacing and the initial value of the density. A second possibility is a particle distribution resulting of a spatially constant particle mass (flag constspacing= false). According to the initial value of the density, the particles will then be further apart in the low density area than in the high density area. For the further course of the program it is essential that each particle has a unique mean of identification. For more sophisticated 2D or 3D codes, this issue is resolved by explicitely assigning each particle an identification number. For this 1D code however, particle identification is already implicitely given due to the way, the particles and their information are stored: the position, consisting only of an x value, of each particle is stored as one component of a position vector X. Therefore the particles can be clearly identified by the number of the vector component they are stored into.

Once the particles are set up, they are assigned their initial values for the shock-tube problem as choosen in the `parameters.h` file.



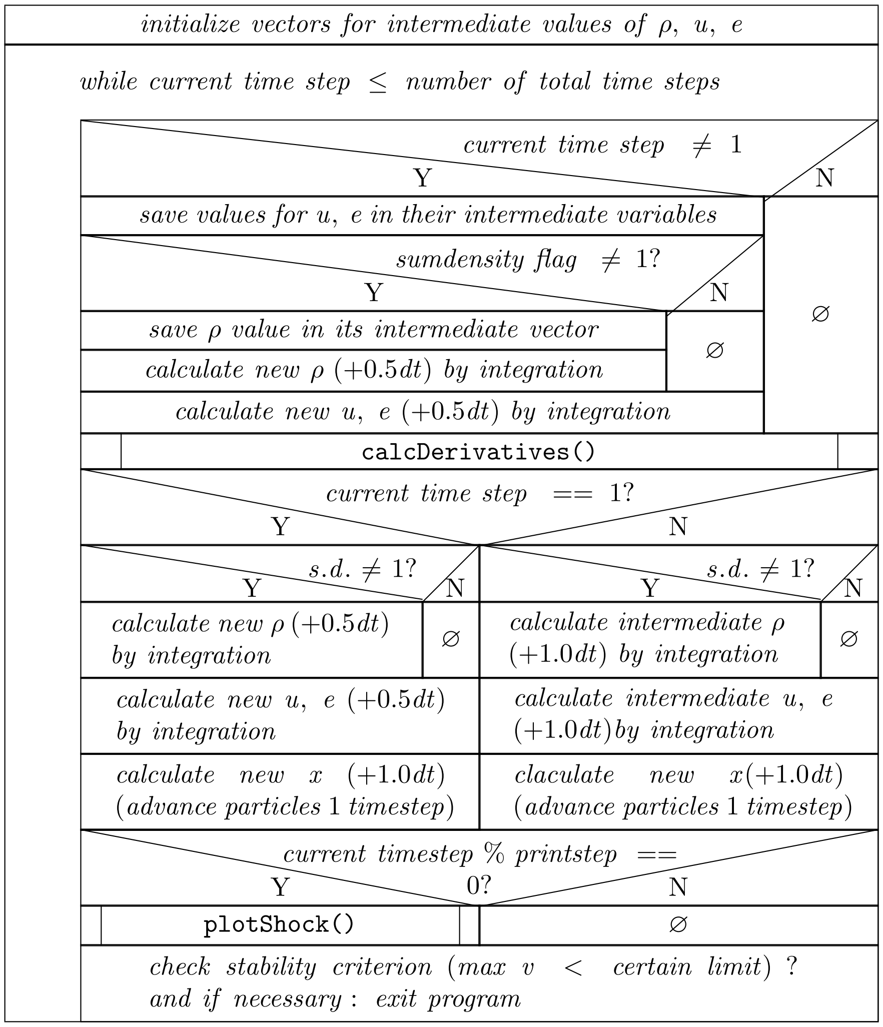
```
SETUPSIM
Calculate LHS particle spacing (dxl) based on constant mass
Calculate RHS particle spacing (dxr) based on constant mass
Calculate particle spacing (dx) for constant spacing
constspacing flag == 1?
    Y:
        place particles on LHS of domain starting at center (x = 0) (with distance dx)
        place particles on RHS of domain starting at x = 0 (with distance dx)
    N:
        place particles on LHS of domain starting at x = 0 (with dinstance dxl)
        place particles on RHS of domain starting at x = 0 (with dinstance dxr)
merge particle positions for both sides into one vector x
create vectors with initial values for u, ρ, p, e, m, h for particles on LHS
create vectors with initial values for u, ρ, p, e, m, h for particles on RHS
merge LHS and RHS vectors into single vector for each variable
constspacing flag == 1?
    Y:
        calculate mass for particles on left hand side
        calculate mass for particles on right hand side
        merge mass values into one single vector (overwrite mass vector from above)
    N:
        ∅
initialize vectors for derivatives du, de, dρ
```

Figure 2.8: The structure of the setupSim–function

The marchTime function

The marchTime function conducts the numerical integration using the leapfrog scheme described in section (2.2.8). If the program runs in the continuity density mode (flag sumdensity = false), the density is integrated as well, whereas if the program runs in the summation density mode (flag sumdensity = true) the density is smoothed. The latter operation is not performed directly in this function but in one of the (sub)functions that it calls.

MARCHTIME



```
initialize vectors for intermediate values of $\rho$, $u$, $e$
while current time step $\leq$ number of total time steps
    current time step $\neq$ 1
    Y:
        save values for $u$, $e$ in their intermediate variables
        sumdensity flag $\neq$ 1?
        Y:
            save $\rho$ value in its intermediate vector
            calculate new $\rho$ $(+0.5dt)$ by integration
        N:
            $\varnothing$
        calculate new $u$, $e$ $(+0.5dt)$ by integration
    N:
        $\varnothing$
    calcDerivatives()
    current time step == 1?
    Y:
        s.d. $\neq$ 1?
        Y:
            calculate new $\rho$ $(+0.5dt)$ by integration
        N:
            $\varnothing$
        calculate new $u$, $e$ $(+0.5dt)$ by integration
        calculate new $x$ $(+1.0dt)$ (advance particles 1 timestep)
    N:
        s.d. $\neq$ 1?
        Y:
            calculate intermediate $\rho$ $(+1.0dt)$ by integration
        N:
            $\varnothing$
        calculate intermediate $u$, $e$ $(+1.0dt)$by integration
        claculate new $x(+1.0dt)$ (advance particles 1 timestep)
    current timestep % printstep == 0?
    Y:
        plotShock()
    N:
        $\varnothing$
    check stability criterion (max $v$ < certain limit) ?
    and if necessary : exit program
```

Figure 2.9: The structure of the marchTime–function, the abbreviation s.d. in the if–statement designates the sumdensity flag

The calcDerivatives function

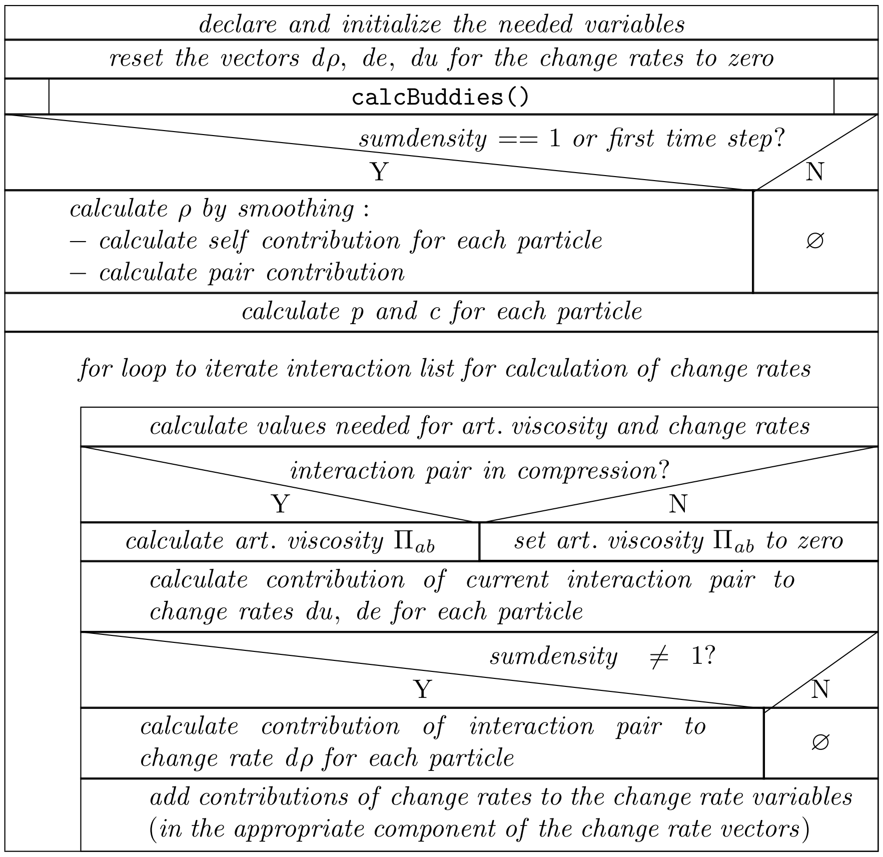
This function performs the calculation of the derivatives for the velocity v (equation (2.23)), the internal energy e (equation (2.24)) and, if the continuity density approach is selected, the density ρ (equation (2.9)). Included in equations (2.24) and (2.23) is the artificial viscosity term, which is basically calculated according to equation (2.21). For the specific case of a shock-tube problem however, it is usual to apply artificial viscosity only in zones where particles are approaching each other and to set it to zero in zones where particles are receding. This ensures that the artificial viscosity is used for shocks and not for rarefactions [18]. As the described 1D SPH code is exclusively applied to a shock-tube problem, this fact has to be taken into account. The criterion for not–receding particles being

$$v_{ab} \cdot r_{ab} \leq 0 \tag{2.66}$$

the final expression for the artificial viscosity in this 1D shock-tube program is

$$\Pi_{ab} = \begin{cases} \text{eq. (2.21)}, & \text{for } v_{ab} \cdot r_{ab} \leq 0 \\ 0, & \text{for } v_{ab} \cdot r_{ab} > 0 \end{cases} \tag{2.67}$$

CALCDERIVATIVES



```
declare and initialize the needed variables
reset the vectors $d\rho$, $de$, $du$ for the change rates to zero
calcBuddies()
sumdensity == 1 or first time step?
  Y:
    calculate $\rho$ by smoothing :
    – calculate self contribution for each particle
    – calculate pair contribution
  N:
    ∅
calculate p and c for each particle
for loop to iterate interaction list for calculation of change rates
    calculate values needed for art. viscosity and change rates
    interaction pair in compression?
      Y:
        calculate art. viscosity $\Pi_{ab}$
      N:
        set art. viscosity $\Pi_{ab}$ to zero
    calculate contribution of current interaction pair to
    change rates du, de for each particle
    sumdensity $\neq$ 1?
      Y:
        calculate contribution of interaction pair to
        change rate $d\rho$ for each particle
      N:
        ∅
    add contributions of change rates to the change rate variables
    (in the appropriate component of the change rate vectors)
```

Figure 2.10: The structure of the calcDerivatives–function

The calcBuddies function

This function conducts the nearest neighboring particle search as described in section (2.2.2) using the pairwise interaction method. If two particles i and j constitute an interaction pair, their unique particle ID (see section (2.4.2.2)) is stored in the same component of the vectors pairi and pairj respectively. pairi and pairj belong to the `simulation` data structure.

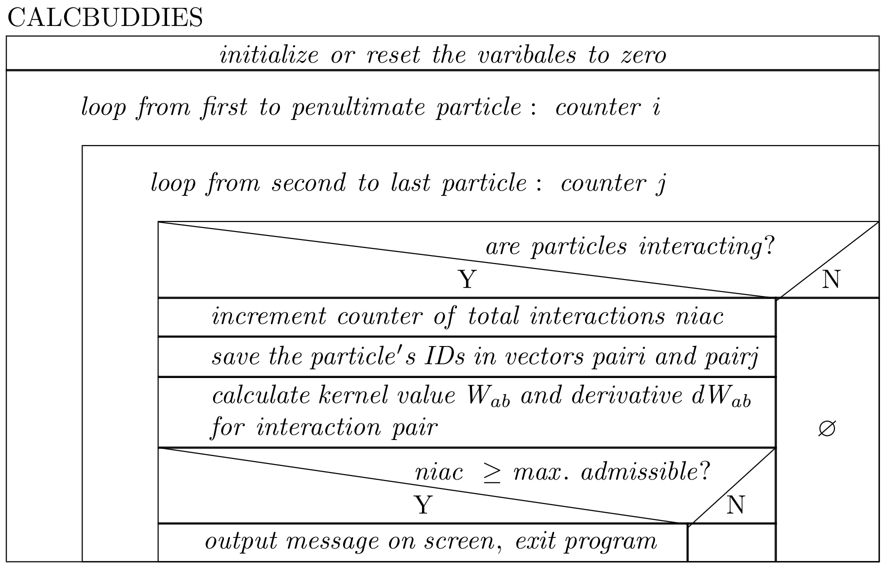

CALCBUDDIES
initialize or reset the varibales to zero
loop from first to penultimate particle : counter i
loop from second to last particle : counter j
are particles interacting?
Y
N
increment counter of total interactions niac
save the particle's IDs in vectors pairi and pairj
calculate kernel value W_{ab} and derivative dW_{ab} for interaction pair
∅
niac ≥ max. admissible?
Y
N
output message on screen, exit program

Figure 2.11: The structure of the calcBuddies–function

The W function

The W() function implements the smoothing kernel for the SPH method, which in this case is the cubic spline kernel (see section (2.2.1)). For implementation purposes the expression of this kernel function (eq. (2.15)) has been expanded resulting in

$$M_4(x) = \begin{cases} \frac{2}{3} - q^2 + 0,5q^3, & \text{for } 0 \leq \text{q} \leq 1 \\ \frac{1}{6}(2-q)^3, & \text{for } 1 \leq \text{q} \leq 2 \\ O, & \text{for q} > 2 \end{cases} \tag{2.68}$$

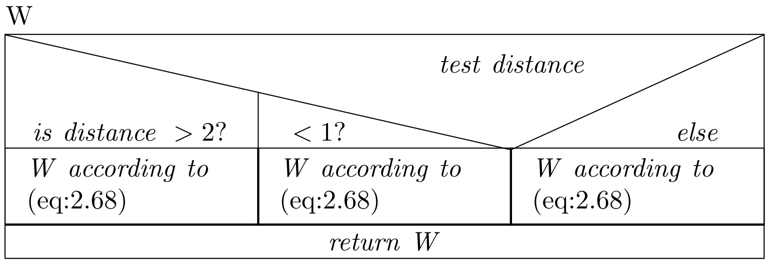


Figure 2.12: The structure of the W–function

The dW function

The derivative of the smoothing kernel $\frac{dW}{dx}$ (in 1D) is calculated by this function. Based on the definition of the cubic Spline Kernel(2.15), this derivative can be determined easily as

$$M_4(x) = \begin{cases} 2q + 1,5q^2, & \text{for } 0 \leq \text{q} \leq 1 \\ -\frac{1}{2}(2-q)^2, & \text{for } 1 \leq \text{q} \leq 2 \\ O, & \text{for q} > 2 \end{cases} \tag{2.69}$$

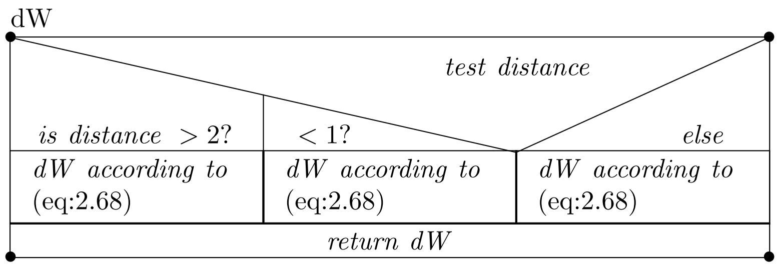
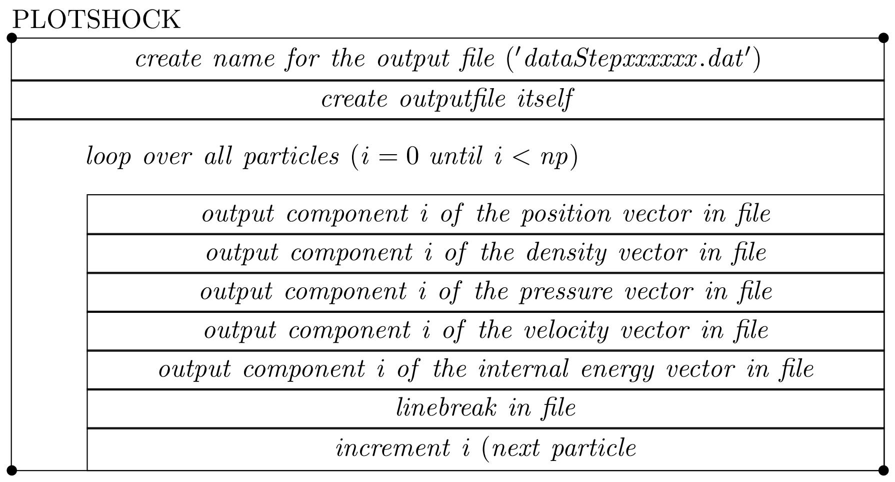

dW
test distance
is distance > 2?
< 1?
else
dW according to (eq:2.68)
dW according to (eq:2.68)
dW according to (eq:2.68)
return dW

Figure 2.13: The structure of the dW–function

The plotShock function

This function corresponds to the output module as shown in figure (2.1) and writes the simulation results at a certain time step in a file named `dataStepxxxxxx.txt`, where xxxxxx indicates the corresponding number of timesteps. The frequency of the output can be choosen by editing the parameter `printstep` of the `parameter` data structure. Each output file has the following format:

$x \; \rho \; p \; u \; e$



```
PLOTSHOCK
create name for the output file ($'dataStepxxxxxx.dat'$)
create outputfile itself
loop over all particles ($i = 0$ until $i < np$)
    output component $i$ of the position vector in file
    output component $i$ of the density vector in file
    output component $i$ of the pressure vector in file
    output component $i$ of the velocity vector in file
    output component $i$ of the internal energy vector in file
    linebreak in file
    increment $i$ (next particle
```

Figure 2.14: The structure of the plotShock–function

2.4.3 How to install and run the program

Installation. An actual installation is not necessary for the 1D SPH–code, but only the directory `1DSPH` has to be present on the computer which is supposed to use the code. The directory can either be copied from the attached disc or downloaded from the public git–repository 1DSPH. Git (www.github.com) is a code–hosting and collaboration management tool. The git–repository can be cloned to the local disc by typing the command (provided git is installed on the local computer):

WORK IN PROGRESS

2.5 2D SPH code

2.5.1 Short overview

The 2D SPH code is more sophisticated than the formerly introduced 1D code. Features such as different types of boundary conditions, a more complex and more efficient neighbouring particle search allgorithm, physical viscosity description and heat conduction, as well as an automatic time stepping control are implemented. Thus, this program is more flexible and can be used to treat problems of real scientific and practical importance.

Just like the 1D SPH code, the 2D code is implemented in C/C++. While the 1D code is still using a functional programming approach, the 2D code follows the object-oriented programming paradigm.

But before starting to describe the structure of the code including all the objects and their interaction (subject of the following paragraphs), a short introduction on the file structure of the whole repository, which containins code, pre– and post–processing scripts, results etc. is given.

The main folder is called `sph-blitz` and contains a number of sub–folders. Those that are important for the use of the program, are briefly detailed here. The others contain different add–ins and third–party libraries for the program, but their detailed knowledge is not required. While information on how to install and run the program is given in the corresponding section (2.5.4), the present description shall be considered as a first orientation guide. It should help the user to get a rough idea on where to find what.

- `src`
  This folder contains the source code of the C++ program. It has again several subfolders, some of which with further source-files that have been grouped together. Furthermore it might contain a folder `html`. This folder hosts the code–documentation, which has automatically been created by a corresponding program. If this folder is not existing, one has to run this program to recreate the most up–to–date documentation of the code. Refer to further below in this section for more information. Finally there is also a folder `outdata`, where the simulation results are written to. This folder might not exist by default. In this case, it is automatically created during simulation.

- `scripts`
  Scripts necessary for pre–processing (generation of initial particle distributions) as well as for post–processing are arranged in this folder.

- `gnuplot`
  Gnuplot being the name of a plotting–software, this folder contains gnuplot–scripts for visualization of the results.

- `results`
  The post–processed results for all the simulations performed and presented in the results–section (3.2) can be found in this folder.

Some of the folders and subfolders contain `README`–files, which can be consulted for more precise information and instructions on the corresponding subject. But before having a look at these, it is recommended to read over section (2.5.4), as many issues are adressed there.

If one desires to acquire detailed understading of this code without previous knowledge, the following roadmap is suggested.

How to approach the code without previous knowledge

1. go through section (2.2) to get an idea on which parts a code principally contains and how its general structure looks like (in particular, consider figure(2.1)).

2. for a first approach to this specific code, consult sections (2.5.2) and (2.5.3).

3. to broaden and deepen understanding of implementation–specific details, an automatic source code documentation system is set up. It provides very extensive documentation (almost to source code level) of all classes, methods and attributes used in the code and even shows dependencies among them by means of call- and caller-graphs for every method (i.e. which methods call the considered method and which methods does the considered method call). This documentation tool is called doxygen and is used in the following way: Run doxygen in a shell placed in the `sph-blitz/src`-folder to generate the documentation in form of html-files. The command, provided doxygen is installed on the computer, is:

   ```
   doxygen
   ```

   The html-files are written in a newly created folder `sph-blitz/src/html`. Open the main-page of the documentation (index.html) in a browser. The corresponding command supposing one uses for example a firefox browser is:

   ```
   firefox html/index.html &
   ```

   Now, one can navigate through the documentation.

4. the last step might be (provided such detailed knowledge of the code is desired), to directly have a look in the source files. There is additional documentation by comments which are not exported to doxygen.

2.5.2 Components

The different classes and some essential methods are briefly described, to show how the features generally introduced in section (2.2) are concretely implemented. For a more complete view of the various classes, their methods and all their attributes, it is recommended to use doxygen, the automatic code documentation tool (see section 2.5.1).

WORK IN PROGRESS

2.5.3 Structure

WORK IN PROGRESS

2.5.4 How to install and run the program

WORK IN PROGRESS

2.6 Simulation Setup and validation methods for different test cases

To provide users with a quick and complete overview of all the settings employed for the different test cases, they are summarized in two tables for each test case – one with general simulation settings and one with information on the initial particle distribution and particle properties. This is all contained in the respective simulation setup paragraph. Furthermore, there is a paragraph entitled validation methods for each test case. It introduces the respective post–processing procedure and the logic of validation.

2.6.1 Employed Norm Definition

Where necessary, the simulation results are compared quantitatively to a reference solution. The error for a certain scalar magnitude B (B might for example be the density ρ, the internal ernegy e, the etc.) is computed using the L_1, L_2 and L_∞ norms as used in [24]. The local error E_i for each particle i is calculated by

$$E_i = (B_{ref} - B_{simu}) \tag{2.70}$$

where the subscribts *ref* and *simu* denote the reference and simulation solutions respectively. Then the error norms can be calculated as

$$\begin{aligned} L_1 &= \frac{1}{|B_{char}|}\,||E||_1 = \frac{1}{|B_{char}|}\left(\frac{\sum_i |E_i|\, C_i}{\sum_i C_i}\right) \\ L_2 &= \frac{1}{|B_{char}|}\,||E||_2 = \frac{1}{|B_{char}|}\left(\frac{\sum_i E_i^2\, C_i}{\sum_i C_i}\right)^{\frac{1}{2}} \\ L_\infty &= \frac{1}{|B_{char}|}\,||E||_\infty = \frac{1}{|B_{char}|}\max|E_i| \end{aligned} \tag{2.71}$$

The normalization factor $|B_{char}|$ is a characteristic value of the quantity B. This value is specific to the problem. Furthermore, the above given definition (eq. (2.71)) is a general formulation allowing for a weighted error norm calculation. The weight factor C_i can be any reasonable variable. For example taking the particle volume ($C_i = V_i$) one obtains a volume weighted norm. Taking $C_i = 1$ simply gives the ordinary norm definition.

2.6.2 1D SPH code

The simulation setup for the shock–tube test problem is already integrated in the 1D SPH–code, i.e. no pre–processing and the like has to be done before running the code. The parameter–setting, which is used for the simulation, from which the sample result presented in section (3.1) is obtained, are the following:

WORK IN PROGRESS

2.6.3 2D SPH code

Chapter 3

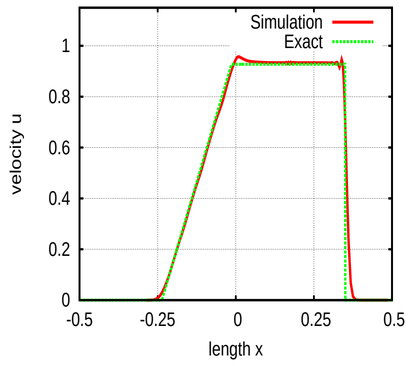
Results

This chapter presents the results obtained with the developed codes. The focus clearly lies on the 2D code which is finally used to simulate the porosity configuration. However, as the 1D code has be introduced in the chapter before, an example of its application to the 1D shock-tube–problem is shown in the following section. It serves for a first qualitative discussion of the shock–tube results that can be obtained with SPH. The 2D code result section mainly features the results of the previously introduced test–cases (2.3) and finally a simulation of the flow in a porosity.

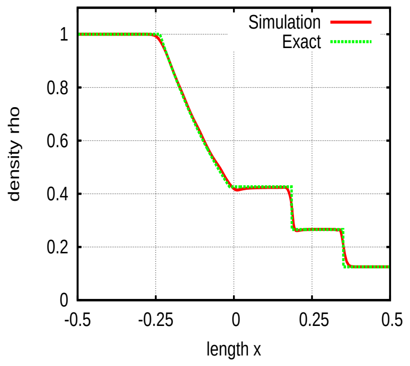
3.1 1D SPH code

As mentionned above, the 1D SPH–code, which only serves demonstration purposes and does not have any practical use, incorporates the simulation setup for the shock-tube–problem. The code is developed to get familiarized with the SPH implementation and to provide a very simple playground to experiment and find the right set of SPH equations for compressible flows. Figure (3.1) shows the results of a simulation with a constant smoothing length of $h = 0.0065$ and a constant mass particle distribution, i.e. with different particle spacings on the LHS and RHS of the shock–tube problem (see paragraph (2.4.2.2)) for an introduction of the two shock–tube–specific particle distributions, namely with constant mass or constant spacing. The complete simulation settings for this example can be found in section (2.6.2).

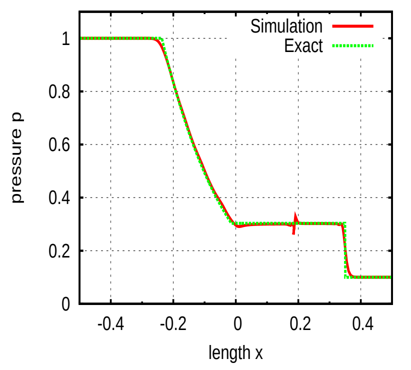
While quantitative evaluation of the shock–tube results will only be done with the 2D code, some qualitative statements can already be made at this place. Comparison with SPH–solutions from literature shows that the profiles presented in figure (3.1) are typical. Taking for example the results of [13] or [20] one can observe the same tendencies: the shock front is resolved within several (2-3) smoothing lengths. The same applies for the contact discontinuity in the variables where there are discontinuities (ρ, e). Furthermore the fact that the internal energy is overestimated at the high energy side of the contact



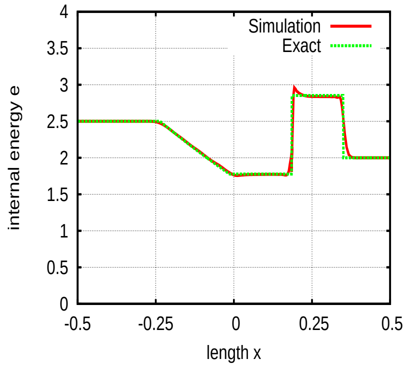
(a)



(b)



(c)



(d)

Figure 3.1: Example of a shock-tube result obtained with the 1D SPH–code for a simulation time of $t = 0.2$ contrasted with the exact analytical solution (see section (2.3.1)): ((a)) velocity profile; ((b)) density profile; ((c)) pressure profile;((d)) profile of the internal energy.

discontinuity seems to be typical as well and a remedy is found by smoothing the initial energy profile before starting the actual calculation [18, 25]. On the other hand by smoothing initial conditions, one looses accuracy in the resolution of the rarefaction [25]. The oscillations after the shock front (notably in the velocity profile) are normal as well and by the way are one of the reasons why an artificial viscosity has to be used for shock simulations: besides preventing particles from interpenetrating at the shock front, another main function of the arificial viscosity is to dissipate these velocity oscillations [18, 27]. Finally, the little blib in the pressure profile at the contact discontinuity is known in literature as well. It can be removed by introducing a small amount of heat diffusion into the energy equation [15]. The logic behind the use/effect of this artificial heat diffusion term is the following: the artificial viscosity, which is also taken into account in the energy equation (see section (2.2.5)), may produce excessive heating [27]. This phenomenon is commonly refered to as wall–heating errors (originally discovered by [23]), and it can be significantly reduced by introducing an atrificial heat diffusion term to the internal energy equation. The exact form of this term however, differs from author to author. Monaghan [15] for example only suggests a constant value of heat diffusion to be added for the shock–tube case. More sophisticated, variable heat diffusion terms are proposed by Price [25] and Sigalotti [27]. However, as the wall–heating error only occurrs for strong shocks and as there are no shocks at all expected in the application of this code, which is a high enthalphy flow through porous meida, this "trick" is not implemented.

The first qualitative evaluation of the simulation results implies that the equations used for the compressible 1D SPH code are the good ones and encourages the integration of these equations in the more complex 2D SPH code, where extensive quantitative evaluation is performed.

3.2 2D SPH code

3.2.1 shock–tube test case

DONE – NOT YET INTEGRATED IN REPORT

3.2.2 linear acoustic waves

DONE – NOT YET INTEGRATED IN REPORT

3.2.3 Taylor–Green flow

ALMOST DONE – NOT YET INTEGRATED IN REPORT

3.2.4 Pure Heat–Conduction

DONE – NOT YET INTEGRATED IN REPORT

3.2.5 Compressible Couette–flow

STILL TO BE DONE

Chapter 4

Conclusion

As the project is not finished until december, it may be to early to draw a final conclusion. However, I am optimistic to be able to simulate a configuration including porosities by the end of the project. I used the time so far to become familiarized with the SPH–method and all the tools that go with an activity in numerics (including Linux and Co.). As a first step I developed a little 1D SPH–code. Then I tried to understand the 2D incompressible multiphase SPH code, which I had to modify. This accomplished, I started implementing the equations which I had identified with the little 1D code as the right ones for compressible flows and adapted the 2D code to that effect. The shock–tube test case and the acoustic wave propagation simulations showed that so far everything was correct. Finding an appropriate formulation for the physical viscosity, i.e. compressible and for variable viscosity, turned out to be difficult for SPH. I first implemented a constant viscosity formulation and ran the Taylor–Green test–case. The results are not yet exact, but I have understood the error. Independently, I have implemented a heat conduction model in the energy equation which has already been tested and which works very well. Besides finding the error in the viscosity implementation, one big test–case remains: the compressible Couette–flow, which combines all phenomena. If this gives the right results, the final simulation of a porous configuration may be conducted. I am really curious if this will work out... we will know by december.

List of Figures

Bibliography

[1] Mihai Basa, Nathan J. Quinlan, and Martin Lastiwka. Robustness and accuracy of sph formulations for viscous ?ow. *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS*, 60:1127–1148, 2009.

[2] L. Brookshaw. *The Stability of Binary Systems and Rotating Stars.* PhD thesis, Monash University, 1986.

[3] Paul W. Cleary and Joseph J. Monaghan. Conduction modelling using smoothed particle hydrodynamics. *J. Comput. Phys.*, 148(1):227–264, January 1999.

[4] R. Courant, K. Friedrichs, and H. Lewy. ber die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100:32?–74, 1928.

[5] David A. Fulk and Dennis W. Quinn. An analysis of 1-d smoothed particle hydrodynamics kernels. *Journal of Computational Physics*, 126:165–180, 1996.

[6] R A Gingold and J J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, 1977.

[7] Lars Hernquist and Neal Katz. Treesph - a unification of sph with the hierarchical tree method. *Astrophysical Journal Supplement Series*, 70:419–446, 1989.

[8] Roger W. Hockney and James W. Eastwood. *Computer simulation using particles.* Taylor & Francis, Inc., 1988.

[9] X.Y. Hu and N.A. Adams. A multi-phase SPH method for macroscopic and mesoscopic flows. *J. Comput. Phys.*, 213(2):844–861, 2006.

[10] X.Y. Hu and N.A. Adams. An incompressible multi-phase SPH method. *J. Comput. Phys.*, 227(1):264–278, 2007.

[11] Jr. John D. Anderson. *Modern Compressible Flow: With Historical Perspective.* McGraw-Hill Science/Engineering/Math, 2002.

[12] L. D. LIBERSKY, A. G. PETSCHEK, T. C. CARNEY, J. R. HIPP, and F. A. ALLAHDADI. High-strain lagrangian hydrodynamics - a 3-dimensional sph code for dynamic material response. *J. Comput. Phys.*, 109(1):67–75, November 1993.

[13] Gui-Rong Liu and M. B. Liu. *Smoothed Particle Hydrodynamics - a mesh-free particle method*. World Scientific, 2003.

[14] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, 1977.

[15] J. Monaghan, J. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30:543–574, 1992.

[16] J. J. Monaghan. An introduction to sph. *Computer Physics Communications*, 48(1):89–96, 1988.

[17] J. J. Monaghan. Simulating free surface flows with sph. *J. Comput. Phys.*, 110:399–406, 1994.

[18] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports in Progress in Physics*, 68(8):1703–1759, 2005.

[19] J. J. Monaghan and R. A. Gingold. Shock simulation by the particle method sph. *Journal of Computational Physics*, 52:374–389, 1983.

[20] J. J. Monaghan and H. Pongracic. Artificial viscosity for particle methods. *Applied Numerical Mathematics*, 1:187–194, 1985.

[21] J.J. Monaghan. On the problem of penetration in particle methods. *J. Comput. Phys.*, 82(1):1–15, 1989.

[22] I. Nassi and B. Shneiderman. Flowchart techniques for structured programming. *SIGPLAN Not.*, 8(8):12–26, 1973.

[23] W.F. Noh. Errors for calculations of strong shocks using an arti?cial viscosity and an arti?cial heat ?ux. *Journal of Computational Physics*, 72:78, 1978.

[24] Igor L. Novak, Fei Gao, Yung-Sze Choi, Diana Resasco, James C. Scha?, and Boris M. Slepchenko. Di?usion on a curved surface coupled to di?usion in the volume: Application to cell biology. *Journal of Computational Physics*, 226:1271?1290, 2007.

[25] D. Price. *Magnetic fields in Astrophysics*. PhD thesis, University of Cambridge, 2004.

[26] I. J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions: part a. *Quarterly Appl. math.*, IV:45–99, 1946.

[27] Leonardo Di G. Sigalotti, Hender Lopez, Arnaldo Donoso, Eloy Sira, and Jaime Klapp. A shock-capturing sph scheme based on adaptive kernel estimation. *Journal of Computational Physics*, 212:124–149, 2006.

[28] G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27:1–31, 1978.

[29] George W. Stewart and Robert B. Lindsay. *Acoustics A Text on Theory and Applications*. D. van Nostrand Company, Inc., 1930.

[30] J. VonNeumann and R. D. Richtmyer. A method for the numerical calculation of hydrodynamic shocks. *Journal of Applied Physics*, 21:232–237, 1950.