Chapter 2

Methods

This chapter describes the methods used to model the collisions covered in this thesis. The first section gives a short review of the governing physics of collision: Fluid- and thermodynamics. Collision often can be characterized by large spatial dynamics: The mass is concentrated in bodies which only occupy a small fraction of the space to be modeled. Nevertheless it is still important to resolve the bodies well enough. Grid techniques can only handle this problem with complicated adaptive mesh refinement routines. A more elegant Lagrangian approach is *Smooth Particle Hydrodynamics* (SPH), a method in which the fluid is approximated by particles. It has the advantage that the computational elements in the form of particles follow the mass of the fluid, so the resolution is inherently where it is required. The second section of this cover different formulations of SPH. Gravity plays an important in planetary collisions. First of all gravity is the source of the kinetic energy such collisions. While for small bodies self-gravity can be neglected because it is too weak, it has to be modeled explicitly for large bodies. The third section covers the Barnes & Hut tree algorithm, which allow to implement self-gravity into a particle code. A fourth section describes an implementation of a hydrodynamic SPH code with self-gravity adapted especially for collision simulations called *SPHLATCH*. Two different versions are presented with different approaches for parallelization, one for distributed and one for shared memory computers. The last section of this chapter describes several techniques required for setting up and post-processing collision simulations.

2.1 Fluid- and Thermodynamics

2.1.1 Basics of fluid dynamics

A fluid is a classical continuum approximation of an ensemble of molecules or atoms. Characteristic variables are density ρ, specific internal energy u, temperature T, velocity $\boldsymbol{v}$ and pressure p. Conservation of mass leads directly to the continuum equation, which equals the local change density to the negative product of velocity divergence and density:

$$\frac{\partial \rho}{\partial t} + \rho \nabla \mathbf{v} = 0 \tag{2.1}$$

Momentum conservation ultimately gives us the *Navier-Stokes equation*

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \rho(\mathbf{v}\nabla)\mathbf{v} = -\nabla p + \nabla \Phi + \nu\Big(\nabla^2 \boldsymbol{v} - \frac{2}{3}\nabla\nabla \boldsymbol{v}\Big) \tag{2.2}$$

for a viscous fluid, where ν is the kinematic viscosity and Φ any external potentials acting on the fluid, for example gravity (Shore 2007). In the absence of viscosity and external potentials, the equation reduces to the momentum equation of the *Euler equations*

$$-\frac{\partial \mathbf{v}}{\partial t} = (\mathbf{v}\nabla)\mathbf{v} + \frac{\nabla p}{\rho} \tag{2.3}$$

Energy conservation in the absence of dissipative terms yields the energy equation

$$\frac{\partial}{\partial t}\Big(\rho u + \frac{1}{2}\rho \boldsymbol{v}^2\Big) + \nabla \boldsymbol{v}\Big(\rho u + \frac{1}{2}\rho \boldsymbol{v}^2 + p\Big) = 0 \tag{2.4}$$

Those equations are formulated in an eulerian form, this means in an inertial frame of reference. The fluid moves relative to this reference frame. If we choose a frame of reference at rest relative to the fluid, we get a Lagrangian description of the momentum equation 2.3 where the advection term $\boldsymbol{v}(\nabla \boldsymbol{v})$ vanishes:

$$-\frac{\partial \mathbf{v}}{\partial t} = \frac{\nabla p}{\rho} \tag{2.5}$$

The continuity equation is un-affected by the choice of a Lagrangian frame of reference, as it depends only on the velocity divergence and not the absolute velocity. The actual Lagrangian of a in-viscous fluid without an external potential is given by

$$\mathcal{L} = \int \Big(\frac{1}{2}\rho \boldsymbol{v}^2 - \rho u\Big) dV \tag{2.6}$$

2.1.2 SPH formalism

SPH is the abbreviation for *Smoothed particle hydrodynamics* and is approach to discretize fluids. The basic idea of SPH is to interpolate all continuum variables in space with an interpolant: If a variable $A(\boldsymbol{r})$ is defined at all points in space $\boldsymbol{r}$, we start with the basic equality

$$A(\boldsymbol{r}) = \int \delta(\boldsymbol{r} - \boldsymbol{r}')A(\boldsymbol{r}')\boldsymbol{dr}' \tag{2.7}$$

We now replace the δ-function with a smoothing kernel W, which has typical width h, the smoothing length, and which fulfils the following two criteria

$$\lim_{h\to 0} W(\boldsymbol{r} - \boldsymbol{r}', h) = \delta(\boldsymbol{r} - \boldsymbol{r}') \qquad \int W(\boldsymbol{r} - \boldsymbol{r}', h)\boldsymbol{dr}' = 1 \tag{2.8}$$

Equation 2.7 can be re-written as a Taylor expansion:

$$A(\boldsymbol{r}) = \int W(\boldsymbol{r} - \boldsymbol{r}', h)A(\boldsymbol{r}')\boldsymbol{dr}' + O(h^2) \tag{2.9}$$

The variable $A(\boldsymbol{r})$ is now smoothed out over the characteristic scale h. By replacing the δ-function with a kernel of finite length, an error in the order of h^2 is introduced. Now comes the particle part: The continuum is replaced by a finite set of points, *particles*, on which the quantities are defined. The integral can now be replaced by a sum with a finite number of summands:

$$A(\boldsymbol{r}) \approx \sum_j W(\boldsymbol{r} - \boldsymbol{r}_j)A_i V_j \qquad V_j = \frac{m_j}{\rho_j} \tag{2.10}$$

V_j is the particle volume and is usually given by assigning each particle a mass and dividing it by the density. The particles used in the summation are called *neighbors* and are indexed here with j. For most purposes, the variable only needs to be known at particle position i. For convenience, we write the kernel used between two particles as $W_{ij} = W(\boldsymbol{r}_i - \boldsymbol{r}_j, h)$. If the smoothing length goes to zero and the number of neighbors goes to infinity, the sum 2.10 approximates the integral 2.7 exactly. This is the *SPH limit*.

Spatial derivatives can be calculated by taking the partial derivative of the variable integral interpolant and replacing it by the sum over all neighbors:

$$\nabla A(\boldsymbol{r}_i) = \frac{\partial}{\partial \boldsymbol{r}} \int A(\boldsymbol{r}')\boldsymbol{dr}' \frac{\partial}{\partial \boldsymbol{r}} W(\boldsymbol{r}_i - \boldsymbol{r}', h) + O(h^2) \approx \sum_j A_j V_j \nabla W_{ij} \tag{2.11}$$

Rotations of variables are similarly obtained:

$$\nabla \times A_i \approx \sum_j A_i V_j (\nabla \times W_{ij}) \tag{2.12}$$

Like in in grid based schemes, there are various ways of calculating spatial derivatives in SPH. Depending on the actual application of the derivative, whether it is to get the velocity divergence or a pressure gradient, there are better suited variants which show better error behavior. They will be presented further below.

The kernel has to fulfill the two constraints given by equations 2.8. A natural choice would be a normalized Gaussian, but this function has the unpleasant property that it never goes to zero. All particles, also distant ones, would contribute to every individual SPH sum, making the method very inefficient although the distant contributions to sums would be very small. A better choice therefore are functions with compact support, which means that they go to zero at finite distance. A common choice is the cubic spline kernel

$$W(|\boldsymbol{r}|, h) = \frac{\sigma}{h^\nu \pi} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & r \leq h \\ \frac{1}{4}(2-q)^3 & h < r \leq 2h \\ 0 & 2h < r \end{cases} \qquad q = r/h \tag{2.13}$$

where ν stands for the number of spatial dimensions and $\sigma = \left(\frac{2}{3}, \frac{10}{7\pi}, \frac{1}{\pi}\right)$ for $\left(1, 2, 3\right)$ dimensions. This kernel vanishes for distances greater than $2h$, reducing SPH sums to contributing terms of only the nearest neighbors. The first derivative is given by

$$\nabla W(\boldsymbol{r}, h) = \frac{\boldsymbol{r}}{r} \frac{\sigma}{h^\nu \pi} \begin{cases} 1 - 3q + \frac{9}{4}q^2 & r \leq h \\ \frac{3}{4}(2-q)^2 & h < r \leq 2h \\ 0 & 2h < r \end{cases} \tag{2.14}$$

and is also continuous.

The choice of the smoothing length is a trade-off. It is large compared compared to the spacing between the particles, then the variables are smoothed out over too large distances and small features cannot be resolved. If the smoothing length is too small, the variables are not sufficiently smoothed and become very noise, ultimately leading to instabilities. As a golden rule, the smoothing length should be chosen, such as the number of neighbors is roughly 50 inside a radius of $2h$.

2.1.3 standard SPH

The most widely used version of SPH uses the density and particle mass to calculate the particle volumes ($V_j = m_j/\rho_j$) and the density to interpolate all variables. The density for the particle with index i is given by the simple particle sum

$$\rho_i = \sum_j m_j W_{ij} \tag{2.15}$$

The velocity divergence $\nabla \boldsymbol{v}$ could in principle be written as

$$\nabla \boldsymbol{v}_i = \sum_j \frac{m_j}{\rho_j} \boldsymbol{v}_j \nabla W_{ij} \tag{2.16}$$

but this form has the disadvantage of being asymmetric and not conserving linear momentum when used in the Euler equation. It is better to apply the *second golden rule* of SPH (Monaghan 1992) and to re-write the velocity divergence term with the density placed inside the operators:

$$\nabla \boldsymbol{v} = \frac{1}{\rho}\Big(\nabla(\rho \boldsymbol{v}) - \boldsymbol{v}\nabla\rho\Big) \tag{2.17}$$

so that the corresponding SPH sum becomes

$$\nabla \boldsymbol{v}_i = \frac{1}{\rho_i}\sum_j \frac{m_j}{\rho_j}\rho_j \boldsymbol{v}_j \nabla W_{ij} - \frac{1}{\rho_i}\boldsymbol{v}_i \sum_j \frac{m_j}{\rho_j}\rho_j \nabla W_{ij} = \frac{1}{\rho_i}\sum_j m_j \boldsymbol{v}_{ij} \nabla W_{ij} \tag{2.18}$$

with $\boldsymbol{v}_{ij} = \boldsymbol{v}_j - \boldsymbol{v}_i$, which is a symmetric form for the velocity divergence. If we take the partial time derivative of 2.15, where only the particle positions in the Kernel depend explicitly on time, we get:

$$\frac{\partial \rho_i}{\partial t} = \sum_j m_j \Big(\frac{\partial}{\partial t}\boldsymbol{r}_{ij}\Big)\nabla W_{ij} = \sum_j m_j \boldsymbol{v}_{ij} \nabla W_{ij} \tag{2.19}$$

So this equation together with the velocity divergence equation 2.18 fulfills the continuity equation 2.1 for each particle exactly.

The equations of motion in standard SPH can be derived in two different ways: With the inviscous equation of motion of the Navier-Stokes equation 2.3, a symmetric formulation for the pressure gradient term $\nabla p/\rho$ can be found with the trick used above for the velocity divergence. A more elegant way is to derive the equation of motion directly from the Lagrangian given by

equation 2.6 whose SPH formulation is

$$\mathcal{L} = \sum_j m_j \Big(\frac{1}{2} \boldsymbol{v}_j^2 - u(\rho_j, s_j) \Big) \tag{2.20}$$

where u is the specific internal energy of a fluid particle depending only on its density and specific entropy s. The Euler-Lagrange equations for a single particle with index i is

$$\frac{d}{dt} \Big(\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_i} \Big) - \frac{\partial \mathcal{L}}{\partial \boldsymbol{r}_i} = 0 \tag{2.21}$$

which yields

$$m_i \frac{d\boldsymbol{v}_i}{dt} = \sum_j m_j \frac{\partial u_j}{\partial \rho_j} \frac{\partial \rho_j}{\partial \boldsymbol{r}_i} \tag{2.22}$$

for constant entropy, which is the case in the absence of shocks. The first law of thermodynamics for a particle is given by

$$\frac{u_j}{\rho_j} = \frac{p_j}{\rho_j^2} \tag{2.23}$$

and the divergence of the density is directly given by

$$\frac{\partial}{\partial \boldsymbol{r}_i} \rho_j = \sum_k m_k \frac{\partial}{\boldsymbol{r}_i} W_{jk} = \sum_k m_k \nabla W_{jk} (\delta_{ji} - \delta_{ki}) \tag{2.24}$$

where the kernel vanishes except if $i = j$ and if $i = k$. So equation 2.22 yields

$$m_i \frac{d\boldsymbol{v}_i}{dt} = \sum_j m_j \frac{p_j}{\rho_j^2} \sum_k \nabla_i W_{jk} (\delta_{ji} - \delta_{ki}) = m_i \sum_j m_j \Big(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \Big) \nabla W_{ij} \tag{2.25}$$

and leaves us with a symmetric SPH sum for the acceleration of a particle in absence of shocks:

$$\frac{d\boldsymbol{v}_i}{dt} = \sum_j m_j \Big(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \Big) \nabla W_{ij} \tag{2.26}$$

The derivative of the kernel is anti-symmetric ($\nabla W_{ij} = -\nabla W_{jj}$), therefore each interacting particle pair fulfils Newtons third law *actio et reactio* and linear momentum is conserved. Angular momentum is also conserved, which can be shown by taking the time derivative of the $\frac{d}{dt} \sum_i (m_i \boldsymbol{r}_i \times \boldsymbol{v}_i)$ where some index magic (Price 2004) again shows that all terms cancel each other out in the sum over all particles.

Except for isothermal calculations, the energy of the fluid is non-constant, so we also need a

equation for the change in internal energy due to compressional heating. The rate of change in specific internal energy is given by

$$\frac{du}{dt} = -\frac{p}{\rho}\nabla \boldsymbol{v} \tag{2.27}$$

Using the velocity divergence SPH sum 2.18 yields

$$\frac{du_i}{dt} = \frac{p_i}{\rho_i^2}\sum_j m_j \boldsymbol{v}_{ij}\nabla W_{ij} \tag{2.28}$$

An SPH sum of this equation consistent with the equation of motion from above can be obtained by applying the golden rule again and putting the density inside the divergence operators:

$$\frac{du}{dt} = -\nabla\left(\frac{p\boldsymbol{v}}{\rho}\right) + \boldsymbol{v}\nabla\left(\frac{p}{\rho}\right) \tag{2.29}$$

The corresponding SPH sum yields the asymmetric equation

$$\frac{du_i}{dt} = -\sum_j m_j \frac{p_j \boldsymbol{v}_j}{\rho_j}\nabla W_{ij} + \boldsymbol{v}_i \sum_j m_j \frac{p_j}{\rho_j^2}\nabla W_{ij} = \sum_j m_j \boldsymbol{v}_{ij}\frac{p_j}{\rho_j^2}\nabla W_{ij} \tag{2.30}$$

A symmetric formulation can be gained by taking the average of equations 2.28 and 2.30:

$$\frac{du_i}{dt} = \frac{1}{2}\sum_j m_j \boldsymbol{v}_{ij}\left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W_{ij} \tag{2.31}$$

2.1.4 Resolving shocks

Shocks are changes of the characteristics of a fluid on the spatial scale of the molecules mean-free path. Discontinuities arise in the fluid approximation, usually in density, pressure and temperature. Because the sharp changes cannot be resolved properly, numerical schemes tend to react to these discontinuities with unphysical oscillations behind the shock front. Several approaches exist to tackle this problem: Godunov-schemes solve the Riemann problem on both sides of the shock exactly between computational elements and in a second step superpose all pair-wise solutions for each element. In case of SPH this can be done by solving the Riemann problem for each particle against its neighbors (Monaghan 1997). Another approach is the von Neumann and Richtmyer viscosity: By introducing a small amount of viscosity, an *artificial viscosity*, the shock front is smoothed out and the oscillations disappear. This can be implemented in SPH by adding additional terms to the momentum and energy equation. The most commonly used variant is

that given by Monaghan (1992):

$$\left.\frac{d\boldsymbol{v}_i}{dt}\right|_{AV} = \sum_j m_j \frac{-\alpha c_{ij}\mu_{ij} + \beta\mu_{ij}^2}{\rho_{ij}} \nabla W_{ij} \qquad \mu_{ij} = \frac{h\boldsymbol{v}_{ij}\boldsymbol{r}_{ij}}{\boldsymbol{r}_{ij}^2 + 0.01h^2} \quad \text{if} \quad \boldsymbol{r}_{ij}\boldsymbol{v}_{ij} < 0 \tag{2.32}$$

where ρ_{ij} is the averaged density between two particles and μ_{ij} is a signal velocity with an order of magnitude of the speed of sound. The artificial viscosity term is only applied in situations of compression, where $\boldsymbol{r}_{ij}\boldsymbol{v}_{ij} < 0$. The choice of constants is usually $\alpha = 1$ and $\beta = 2\alpha$. The β-term is the actual von Neumann and Richtmyer term and becomes dominant in strong shocks where the velocity difference $\boldsymbol{v}_{ij}$ between two particles in the shock front becomes large. μ_{ij} is called the signal velocity between two particles. Momentum conservation is still guaranteed thanks to the symmetric form of the artificial viscosity term.

The contribution of artificial viscosity to the internal energy can be found by requiring that artificial viscosity itself should be energy conserving. The specific energy change due to the dissipative is given by

$$\left.\frac{de_i}{dt}\right|_{AV} = \left.\frac{du_i}{dt}\right|_{AV} + \boldsymbol{v}_i \left.\frac{d\boldsymbol{v}_i}{dt}\right|_{AV} = 0 \tag{2.33}$$

and should be zero. This leads to the thermal contribution

$$\left.\frac{du_i}{dt}\right|_{AV} = -\sum_j m_j \boldsymbol{v}_{ij} \frac{-\alpha c_{ij}\mu_{ij} + \beta\mu_{ij}^2}{\rho_{ij}} \nabla W_{ij} \tag{2.34}$$

with the same choice for the signal velocity μ_{ij} and numerical constants as in equation 2.32.

A disadvantage of this approach is that this viscosity also acts outside of shocks in situations of weak compression or shear flows. There exist several approaches to fix this issue by detecting shocks more cleverly: One approach by Morris and Monaghan (1997) tries to detect shocks more cleverly than by simply checking for compression. Another approach tries to limit artificial viscosity in shears flows (Balsara 1995), which is useful for example when modeling astrophysical disks. In collision simulations increased viscosity due to artificial viscosity is hardly a problem, therefore we don't include such fixes in the code which will be presented later.

2.1.5 variable smoothing length

In order to meet the requirement of the optimum number of neighbors $N_{N,opt} = 50$ neighbors per particle, the smoothing has to be adjusted for each particle individually. For constant particle masses the density is proportional to the particle density δ_i, which again is inverse proportional

to the particle volume and therefore to the inverse third power of the smoothing length in 3D:

$$\rho_i \sim \delta_i \sim \frac{1}{V_i} \sim \frac{1}{h_i^3} \tag{2.35}$$

So the rate of change of the smoothing length can be obtained by using the continuity equation 2.21 and yields

$$\frac{dh_i}{dt} = -\frac{h_i}{3\rho_i}\frac{d\rho_i}{dt} = \frac{1}{3}h_i \nabla \boldsymbol{v}_i \tag{2.36}$$

This keeps the number of neighbours exactly constant, if the particle density changes isotropically. Usually this is not the case and the number of neighbors (N_N) becomes either too large or too small. In such cases it is useful to overcorrect the smoothing length with the global maximum of velocity divergence $\nabla \boldsymbol{v}_{max} = \max_i \nabla \boldsymbol{v}_i$:

$$\frac{dh_i}{dt} = h_i(k_1 \nabla \boldsymbol{v}_{max} + k_2 \frac{1}{3}\nabla \boldsymbol{v}_i - k_3 \nabla \boldsymbol{v}_{max}) \tag{2.37}$$

$$\begin{aligned} k_1 &= \tfrac{1}{2}(1 + \tanh -\tfrac{N_N - N_{N,min}}{5}) \quad & N_{N,min} &= \tfrac{2}{3}N_{N,opt} \\ k_2 &= 1 - k_1 - k_3 & & \\ k_3 &= \tfrac{1}{2}(1 + \tanh \tfrac{N_N - N_{N,max}}{5}) \quad & N_{N,max} &= \tfrac{5}{3}N_{N,opt} \end{aligned} \tag{2.38}$$

If the number of neighbours is within $\frac{2}{3}N_N$ and $\frac{5}{3}N_N$, the expression 2.36 is dominant. If it's below or above the optimal number, the change rate of the smooting length is over- or undercorrected with the maximum velocity divergence. In practice, this scheme keeps the number of neighbors in reasonable limits, without introducing any instabilities.

2.1.6 Integration

Together with an equation of state which gives us the pressure $p_i(u_i, \rho_i)$ and as a side effect also the speed of sound $c_i(u_i, \rho_i)$, we now have a closed set of equations describing the fluid which can be integrated in time. A common choice due to its simplicity and low number of derivation steps required is the predictor-corrector scheme (Press 2002). For the integration of particle positions we use the second-order form, which yields the prediction step

$$\boldsymbol{r}_i^{(p)} = \boldsymbol{r}_i^{(0)} + \frac{1}{2}(3\boldsymbol{v}_i^{(0)} - \boldsymbol{v}_i^{(-1)})\Delta t \tag{2.39}$$

$$\boldsymbol{v}_i^{(p)} = \boldsymbol{v}_i^{(0)} + \frac{1}{2}(3\boldsymbol{a}_i^{(0)} - \boldsymbol{a}_i^{(-1)})\Delta t \tag{2.40}$$

and the correction step

$$\boldsymbol{r}_i^{(1)} = \boldsymbol{r}_i^{(0)} + \frac{1}{2}(\boldsymbol{v}_i^{(p)} - \boldsymbol{v}_i^{(0)})\Delta t \tag{2.41}$$

$$\boldsymbol{v}_i^{(1)} = \boldsymbol{v}_i^{(0)} + \frac{1}{2}(\boldsymbol{a}_i^{(p)} - \boldsymbol{a}_i^{(0)})\Delta t \tag{2.42}$$

The integrator requires the evaluation of the derivative $\boldsymbol{a}_i(\boldsymbol{r}_i, \boldsymbol{v}_i)$ for two times. As the predicted variables $(\boldsymbol{r}_i^{(p)}, \boldsymbol{v}_i^{(p)})$ and previous variables $(\boldsymbol{r}_i^{(-1)}, \boldsymbol{v}_i^{(-1)})$ are never used at the same time, it is sufficient to store only one additional set of variables. This integrations scheme is therefore also very memory conserving.

Variables defined by first-order differential equations like the internal energy, use the first-order prediction step

$$u_i^{(p)} = u_i^{(0)} + \frac{1}{2}(3\dot{u}_i^{(0)} - \dot{u}_i^{(-1)})\Delta t \tag{2.43}$$

and the correction step

$$u_i^{(1)} = u_i^{(0)} + \frac{1}{2}(\dot{u}_i^{(p)} - \dot{u}_i^{(0)})\Delta t \tag{2.44}$$

The timestep is limited by several factors. Stability analysis of SPH yields the *Courant-Friedrich-Lax* (short: CFL) condition

$$\Delta t_{CFL} = \mathcal{C}_{CFL} \min_i \frac{h_i}{c_i} \tag{2.45}$$

where $\mathcal{C}_{CFL}$ is a constant chosen between $\mathcal{C}_{CFL} = 0.1 \ldots 0.4$. All other integrated variables require, that the their maximal change per time step is not more than their own order of magnitude. In order to avoid arbitrarily small time step when a variable goes to zero, a minimal value for such quantities is introduced. This yields for example for the smoothing length the time step

$$\Delta t_h = \mathcal{C} \min_i \frac{\dot{h}_i}{h_i + h_{min}} \tag{2.46}$$

Other variables integrated in that way and therefore requiring a time step criterion are specific internal energy u_i or the density ρ_i in case the continuity equation is integrated.

The predictor-corrector scheme has the disadvantage of a global time step. Other integrators like the leap-frog integrator allows hierarchical time stepping, due to their inherent symmetry of the sub-steps. In most cases the time step is limited by the CFL-criterion 2.45, which depends on the smoothing length and the speed of sound. For an ideal gas at constant entropy and when using particles of a fixed mass, the time step depends on density as

$$\Delta t_{CFL} \sim \frac{h}{c} \sim \frac{1}{\sqrt[3]{\rho}}\sqrt{\frac{\rho}{p}} \sim \frac{1}{\sqrt[3]{\rho}}\frac{1}{\sqrt{u}} \sim \frac{1}{\rho^{2/3}} \tag{2.47}$$

When particle density varies over several orders of magnitudes, the time step does so too. This if for example the case for cosmological simulations, where the collapse of dilute collapse gas to hot dense clumps results in highly different time steps. In this case, only hierarchical schemes the actual used time step for a particle is in the same order of magnitude as the time step allowed by the various criterions, allows an integration with reasonable computational effort.

For collision events in the gravity regime, the time steps don't vary too much. Solids have similar speed of sound and smoothing lengths, also under compression. The formation of dilute vapor is not an issue, because particles in vapor state have much larger smoothing lengths than solid length, while retaining a comparable speed of sound, resulting in an even larger time step. A hierarchical time stepping scheme is therefore not necessary for the simulations of collisions.

2.1.7 miscible SPH

Standard SPH interpolates all variables weighed according to density. This works very well for single-phase fluids, but introduces difficulties with multi-phase fluids with large density contrasts. In reality, density can be discontinuous along boundaries, for example for fluid droplets embedded in a gas.

Big density contrasts normally arise, when two fluid with different equations of state interact and have different densities for the same pressure and thermodynamical state. This difference can be orders of magnitude large like for an atmosphere on top of a liquid or a solids. For material in the same phase the density difference is usually not so big, but can still be easily a factor of a magnitude for example for water ice and solid iron ($\rho_0 \approx 0.9 g/cm^3$ vs. $7.8 g/cm^3$).

The actual problem of standard SPH and density contrasts arises for fluids with rather stiff equations of state, such as fluids and solids. Even a small deviation from the zero-pressure density ρ_0 results in a large pressure.

Figure 2.1 demonstrates this with an actual radial profile of a two-fluid 3D sphere in equilibrium, this means that all particles are at rest. The structure models an early proto-Earth of $0.9 M_\oplus$ with a $30 wt\%$ core of iron and a $70 wt\%$ mantle of silicate (SiO_2). A thin black line indicates the equilibrium solution of a 1D Lagrangian code (see section 2.4.2). Ideally the SPH particles would represent this solution in 3D. The left plots show the structure calculated with standard SPH. Particles have the actual density and pressure within a small error at most radii, except for the boundary between the core and the mantle, where there are differences in density leading to a huge pressure deviation (green circles). In standard SPH the iron particles obtain their density by interpolating over their neighbors, which include silicate particles with a considerably lower density for the same pressure. This leads to an under-estimated density at the boundary and ultimately to a pressure much too low. The same effect works the other way round for the silicate

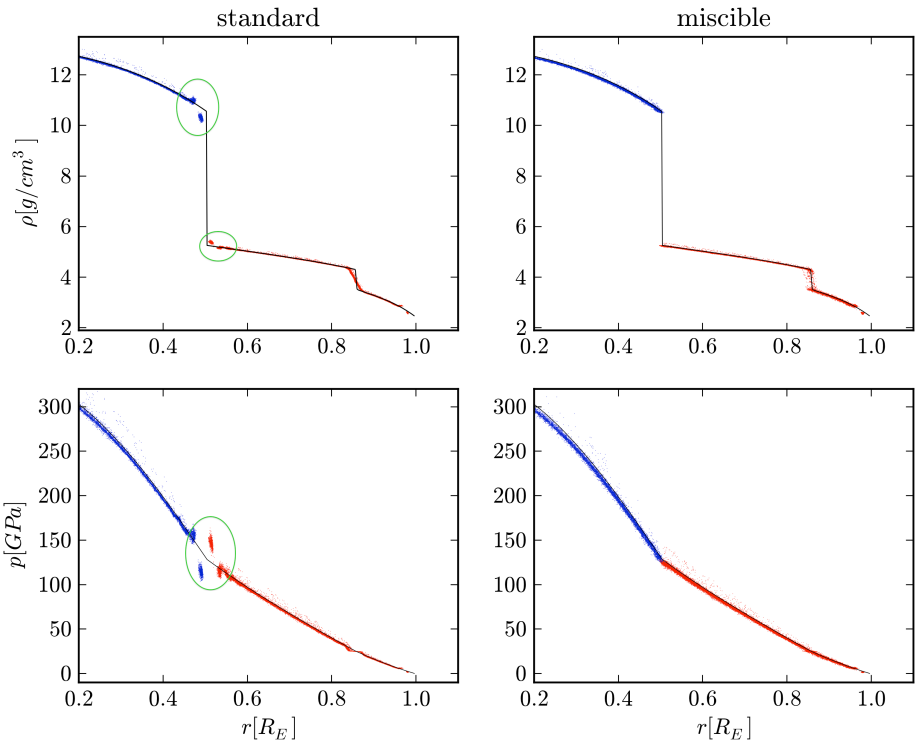

Figure 2.1: Radial structure of a proto-Earth body of $0.9M_\oplus$ with a $30wt\%$ iron core and a $70wt\%$ silicate mantle. Blue dots show the values of iron particles, red dots show silicate particles. The upper plots show density and the lower plots the resulting pressure. The structure on the left plots is calculated with standard SPH and the one on the right plots uses the alternative *miscible SPH* formulation. Both structures are relaxed, which means that the residual particles velocities are very small compared to free-fall velocities inside such a gravitational potential. As in standard SPH density is a smooth value along the boundary between the iron core and the silicate mantle, the iron particles will have an under-density at the boundary, while the silicate particles have an over-density (green circles). This results in anomalous pressures at the boundary. Miscible SPH circumvents this issue by allowing non-smoothed density at the boundary.

particles, which obtain an over-density and therefore also have a too large pressure. By smoothing out the density at the boundary, it reaches unnatural values.

A method first proposed by (Ott and Schnetter 2003) and also mentioned by Solenthaler and Pajarola (2008) and Price (2004) drops the fundamental principle of SPH of interpolating all quantities with density 2.7 and uses the particle density instead

$$\delta_i = \sum_j W_{ij} \tag{2.48}$$

The actual density is then obtained by

$$\rho_i = m_i \delta_i \tag{2.49}$$

While the particle density is still smoothed out, the density can be discontinuous and vary largely if particles of different masses interact. The direct result of this formulation can be seen on the right plots in figure 2.1. The density follows correctly the required values also at the boundary and as a result also the pressure becomes continuous in the correct way.

The other SPH sums can be derived in an analog way as for standard SPH. By taking the time derivative of 2.49 and applying the same trick for symmetrizing derivative sums, we get a new SPH formulation of the continuity equation

$$\frac{d\rho_i}{dt} = m_i \sum_j \boldsymbol{v}_{ij} \nabla W_{ij} \tag{2.50}$$

To derive the momentum and energy equations we use the same procedure as for standard SPH and get

$$\frac{d\boldsymbol{v}_i}{dt} = -\frac{1}{m_i} \sum_j \left(\frac{p_i}{\delta_i^2} + \frac{p_j}{\delta_j^2} \right) \nabla W_{ij} \tag{2.51}$$

$$\frac{du_i}{dt} = \frac{1}{2m_i} \sum_j \boldsymbol{v}_{ij} \left(\frac{p_i}{\delta_i^2} + \frac{p_j}{\delta_j^2} \right) \nabla W_{ij} \tag{2.52}$$

The artificial viscosity term depend directly on density and need to be symmetric in order to conserve linear momentum. Requiring total energy conservation, the contribution to the momentum equation yields

$$\left. \frac{d\boldsymbol{v}_i}{dt} \right|_{AV} = -\frac{1}{m_i} \sum_j \frac{-\alpha c_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\delta_{ij}} \nabla W_{ij} \tag{2.53}$$

and similarly to the energy equation

$$\left.\frac{du_i}{dt}\right|_{AV} = \frac{1}{2m_i}\sum_j \boldsymbol{v}_{ij}\frac{-\alpha c_{ij}\mu_{ij} + \beta\mu_{ij}^2}{\delta_{ij}}\nabla W_{ij} \tag{2.54}$$

with the same factors μ_{ij}, α and β as in equation 2.32.

The failure of standard SPH at boundaries can introduce difficulties in resolving Kelvin-Helmholtz instabilities properly, as noted by Agertz et al. (2006). We will compare the ability to resolve this instability between standard and miscible SPH in chapter 5 about the Moon-forming giant impact.

Another situation where standard SPH fails are mixed states between materials, for example if solid particles are embedded in gas particles. This occurs in the simulations of impacts into proto-Jupiter-like structures, where a solid impactor and, after the collision, solid debris travels through the gaseous envelope of the target.

2.1.8 solid state SPH

SPH can also be used for modelling solids. For an elastic solid, the isotropic pressure of a fluid is replaced with the full stress tensor in order to additionally model shear stresses. Brittle failure due to cracks can be modeled with a statistical approach Grady and Kipp (1980). An SPH implementation of this is presented in Benz and Asphaug (1994). Chapter 4 uses an solid state SPH code with the features mentioned above. The code is described in detail in Nyffeler (2006) and will not be discussed any further here.

2.1.9 fluid SPH summary

Table 2.1 shows a summary of the sums of the three different SPH formulations presented above. On modern computers, the computationally most expensive thing is to find the neighbors and fetch their variables. It is therefore useful, to collect several SPH sums which are independent of each other results to a single summation step. In case of standard SPH this means for example, that the 2nd SPH summation step searches the neighbors for particle with index i and then calculates $\nabla \boldsymbol{v}_i$, $\frac{du_i}{dt}$ and $\frac{d\boldsymbol{v}_i}{dt}$ at the same time. Standard and miscible SPH both require two summation steps, in order to calculate the physical and particle density respectively. In case the density is an integrated variable, only one summation step is required.

variant	standard	integrated density	miscible
1st SPH sums	$\rho_i = \sum_j m_j W_{ij}$	ρ_i is integrated	$\delta_i = \sum_j W_{ij}$ $\rho_i = m_i \delta_i$
2nd SPH sums	$\nabla \boldsymbol{v}_i = \sum_j \frac{m_j}{\rho_j} \nabla W_{ij}$ $\frac{d\boldsymbol{u}_i}{dt} = \frac{1}{2} \sum_j m_j \boldsymbol{v}_{ij} \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} + \Pi_{ij} \right) \nabla W_{ij}$ $\frac{d\boldsymbol{v}_i}{dt} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} + \Pi_{ij} \right) \nabla W_{ij}$ $\frac{d\rho_i}{dt} = \rho_i \nabla \boldsymbol{v}_i$		$\nabla \boldsymbol{v}_i = \frac{1}{m_i} \sum_j \nabla W_{ij}$ $\frac{d\boldsymbol{u}_i}{dt} = \frac{1}{2m_i} \sum_j \boldsymbol{v}_{ij} \left(\frac{p_i}{\delta_i^2} + \frac{p_j}{\delta_j^2} + \Pi_{ij} \right) \nabla W_{ij}$ $\frac{d\boldsymbol{v}_i}{dt} = -\frac{1}{m_i} \sum_j \left(\frac{p_i}{\delta_i^2} + \frac{p_j}{\delta_j^2} + \Pi_{ij} \right) \nabla W_{ij}$
art. visc.	$\Pi_{ij} = \frac{-\alpha c_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\rho_{ij}}$		$\Pi_{ij} = \frac{-\alpha c_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\delta_{ij}}$
	$\mu_{ij} = \frac{h_{ij} \boldsymbol{v}_{ij} \boldsymbol{r}_{ij}}{\boldsymbol{r}_{ij}^2 + 0.01 h^2}$		

Table 2.1: Overview of the sums for different SPH variants

2.1.10 Equations of state

The Navier-Stokes equation together with the continuity equation only present a closed system of equations, if an equation of state (EOS) delivering $p(\rho, s)$ is provided. Instead of specific entropy, the specific energy is often used as the second thermodynamic state variable. The speed of sound defined as $c_S = \sqrt{\partial p / \partial \rho}$ is used for the CFL criterion (equation 2.45) and in the artificial viscosity (equation 2.32).

For an ideal gas, the equation of state and the speed of sound is given by

$$p = (\gamma - 1) \rho u \qquad c_s = \sqrt{\frac{\gamma p}{\rho}} \qquad \gamma = \frac{c_p}{c_v} \tag{2.55}$$

where γ is the ratio of specific heats or the adiabatic index. For a monoatomic gas it is $\gamma = \frac{5}{3}$, whereas for a solar mixture of roughly three quarters of molecular hydrogen and a quarter of atomic helium it is $\gamma \approx 1.53$ (Nelson et al. 2000). The ideal gas EOS can also be written in the form

$$p = s \rho^\gamma \tag{2.56}$$

where s is the specific entropy (Springel 2005).

For condensed matter in collisions processes, the ideal gas approximation breaks down and more complex equations of state are required. For geological materials a popular choice is the ANEOS software package. It uses the basic approach of minimizing the Helmholtz free energy $F(\rho, T)$ for a given pair of density ρ and temperature T. Various analytical terms contributing to the Helmholtz free energy for ionization, phase changes and low temperatures are included and and guarantee a smooth behavior at phase boundaries (Thompson 1990). Pressure and entropy

are derived from the Helmholtz free energy:

$$p = \rho^2 \frac{\partial F(\rho, T)}{\partial \rho}\Big|_T \qquad S = -\frac{\partial F(\rho, T)}{\partial T}\Big|_\rho \tag{2.57}$$

Figure 2.2 shows isobars for different materials obtained by ANEOS. The jumps in entropy at certain temperatures show phase changes, whereas in between phase changes the isobar follows approximately analytical functions. Water at $50bar$ for example (light blue line, figure 2.2, right plot) follows a non-trivial curve in the solid phase, until melting kicks in around $300K$. Then the isobar follows again a straight line as expected for a liquid with a constant ratio of specific heats. Another entropy jump for vaporization occurs slightly below $400K$. Then in the vapor phase, the isobar follows again a straight line as expected for a more or less real gas. Above $1000K$ dissociation of the water molecules starts and the isobar follows a more complicated curve again.

The original version of ANEOS treats the gas phase as a mixture of mono-atomic gases. Compared to a gas of molecules, the monoatomic gas has a very high internal energy and entropy for a given temperature. So in the mono-atomic case vaporization only starts at comparably high energies. Melosh (2007) tackled this problem by modifying original ANEOS for SiO_2 and introducing molecular species in the gas phase. This modified version is called M-ANEOS. Figure 2.3 shows expected gas mixtures for three different pressures as a function of temperature. Vaporization starts by producing a mixture mainly made up of SiO and O_2 gas. Dissociation of these molecules only happens at much higher temperatures. The difference between the two models can be seen by comparing the isobars in figure for dunite (ANEOS, no molecules, green line) and for SiO_2 (M-ANEOS, with molecules, red line). The two materials are comparable silicates. At both pressures, the entropy jump for vaporization for SiO_2 starts at a temperature roughly $2000K$ lower compared to dunite. Vaporization of solid SiO_2 in collisions as a consequence of shock heating occurs, when the particle velocities exceed $7 - 8km/s$ (Melosh 2007).

ANEOS uses as (ρ, T) as the thermodynamical state variables as input parameters, whereas the hydrodynamical equations for a fluid require (ρ, u) or (ρ, s) as state variables, depending on whether the energy or the entropy equation is solved and integrated. Specific energy and entropy are both returned by ANEOS. Luckily both quantities $u(\rho, T)$ and $s(\rho, T)$ are monotonic increasing functions of temperature, so the correct temperature for given specific energy or entropy can be found through a root finding algorithm. The Brent Rooter (Press 2002) converges usually within a few iteration steps, for a temperature range covering a few K to a few ten thousand K and a precision of 3×10^{-8}.

ANEOS dates back to a time, when computer memory was sparse and it was not possible to store large tables of complicated functions. Calling ANEOS itself therefore incorporates numer-

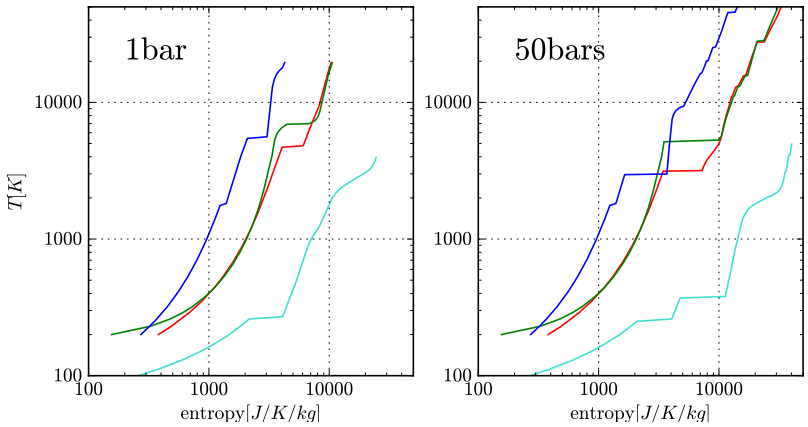

Figure 2.2: Isobars for different material in the ANEOS equation of state. The blue line depicts iron, the green line dunite, the red line SiO_2 and the turquoise line shows H_2O. The flat regions where temperature does not increase for a entropy increase are phase changes.

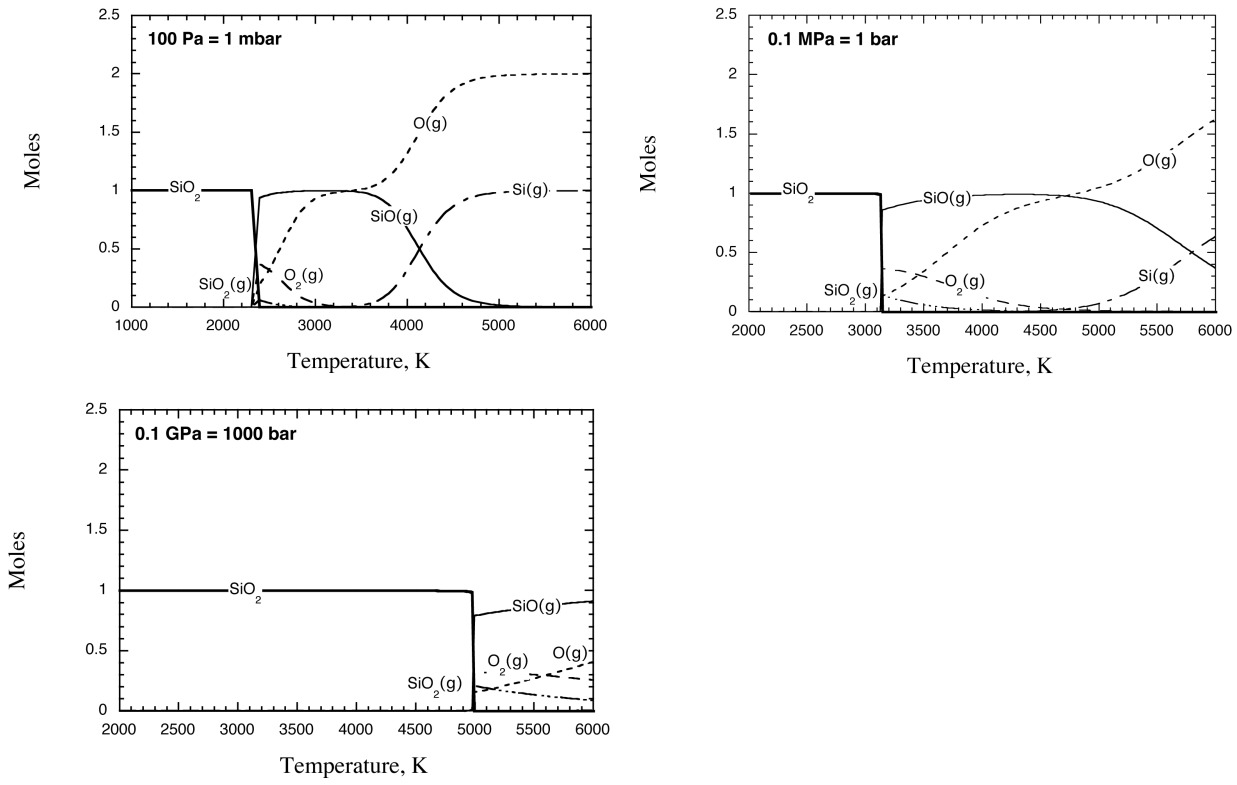

Figure 2.3: Figure 4 from Melosh (2007) showing the present species depending in pressure and temperature for the SiO_2 M-ANEOS equation of state.

ous internal iteration routines. This iterations over iterations make a single EOS call for given (ρ, u) or (ρ, s) computationally relatively costly. Nowadays memory is abundant. Therefore we tabulate ANEOS, so that EOS calls simply reduce to finding suitable points in the parameter space and interpolating between them. Figures 2.4 - 2.7 show the tables for four different material. The parameter space is given by $(\log \rho, \log u)$ for the density-energy table and $(\log \rho, \log u)$ for the density-entropy table. Each material has a reference density ρ_0 at standard conditions. The important shock processes in planetary system collisions between solid or liquid bodies occur at densities roughly inside an order of magnitude above the reference density. At the same time the table needs to cover vaporized material which has a density many magnitudes below the reference value. In order to fulfill both requirements, the table is split into a high- and a low-density region. The high density region spans roughly 2.5 orders of magnitude starting from $100kg/m^3$ with 500 grid points. The low density region covers the vapor and solid-vapor mixture regions and goes all the way down to $10^{-8}kg/m^3$ with only 100 grid points. This low number of grid points over such a large density range is no problem, because all variables behave sufficiently smooth in this regions and interpolation delivers good agreements to the theoretical values. Specific energy and entropy are tabulated in both tables with 1500 points in the logged value. In the rare case, that a value outside of the table is requested, the code falls back to the iteration routine used to create the tables. Figures 2.4 - 2.7 show the phase state information returned by ANEOS. Note that for mixed states, the single density used by ANEOS is the average density for all phases together. So for example in case of fluid droplets embedded into gas of the same material, ANEOS calculates the density as the average density of both the droplets and the gas.

2.2 Gravity

A self-gravitating continuum with the density field $\rho(\boldsymbol{r})$ has the gravitational potential $\Phi(\boldsymbol{r})$ which fulfills the Poisson equation for gravity

$$\nabla(\nabla\Phi(\boldsymbol{r})) = \Delta\Phi(\boldsymbol{r}) = 4\pi G\rho(\boldsymbol{r}) \tag{2.58}$$

and acts on the fluid through the external potential term in the Navier-Stokes equation 2.2. Grid codes often calculate the potential by taking the Fourier transform of the Poisson equation, which yields directly the Fourier transform of the potential which again can be transformed back into real space.

When we approximate the density field by discrete particles like in SPH, self-gravity reduces to the N-body problem, where the potential and the acceleration due to gravity for particle i is

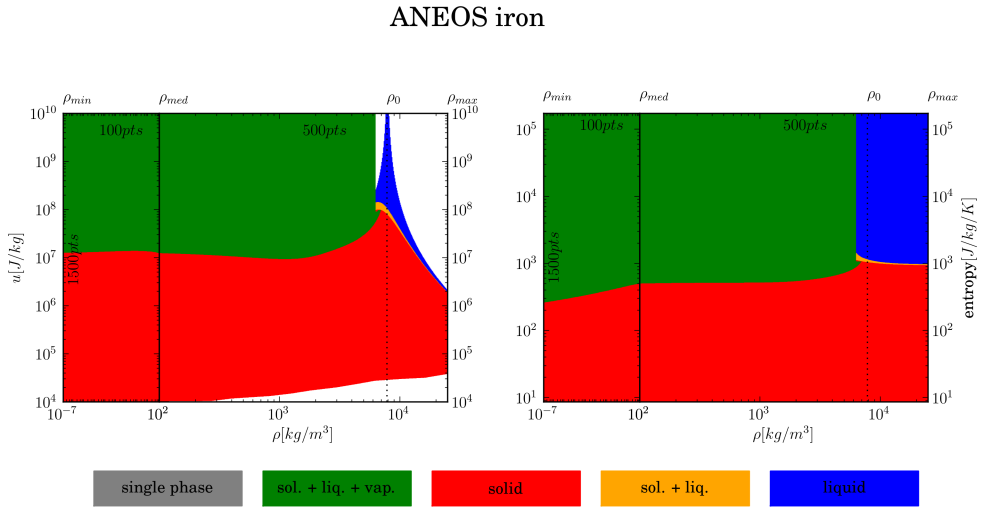

Figure 2.4: Look-up tables for the ANEOS equation of state for iron. The left plot shows the table using density and specific energy as the parameter space, while the right plot shows the table using density and specific entropy. Both tables are divided into two sub-tables. A small table with covers the low-density region and uses 100 grid points equally spaced in logged density from ρ_{min} to ρ_{med} and 1500 points in logged specific energy. In the higher density region from ρ_{med} to ρ_{max}, the density resolution is increased by using 500 grid points in logged density, in order the resolve the phase transitions well enough. The same sub-division also applies to the density-entropy table, except that instead of logged specific energy logged specific entropy is tabulated. The colors in the plot indicate the phase information returned by ANEOS. Solid is red. Orange indicates material where partial melting occurs and both a solid and a liquid phase exists. For an increase in energy or entropy, the material liquifies completely, indicated by blue color. Green color indicates the mixed state, where solid, liquid and vapor phases co-exist at the same time. White regions are invalid.

ANEOS H_2O

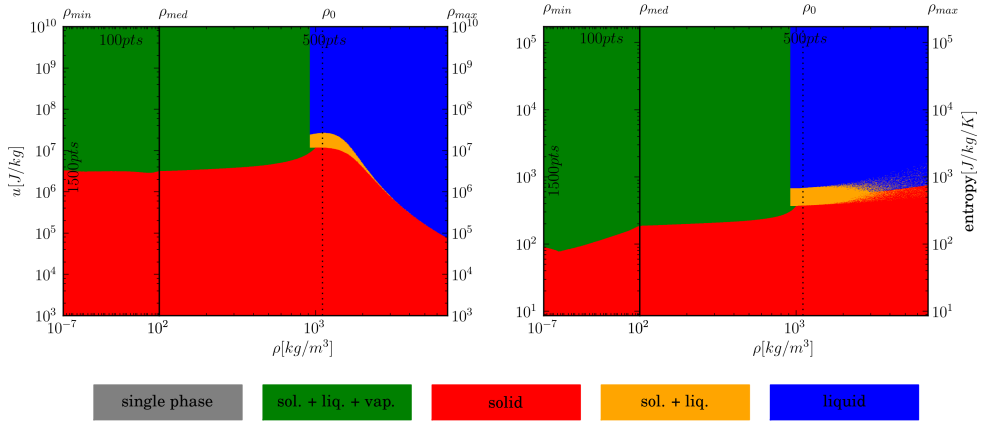

Figure 2.5: The same tables as in 2.4 for H_2O

ANEOS dunite

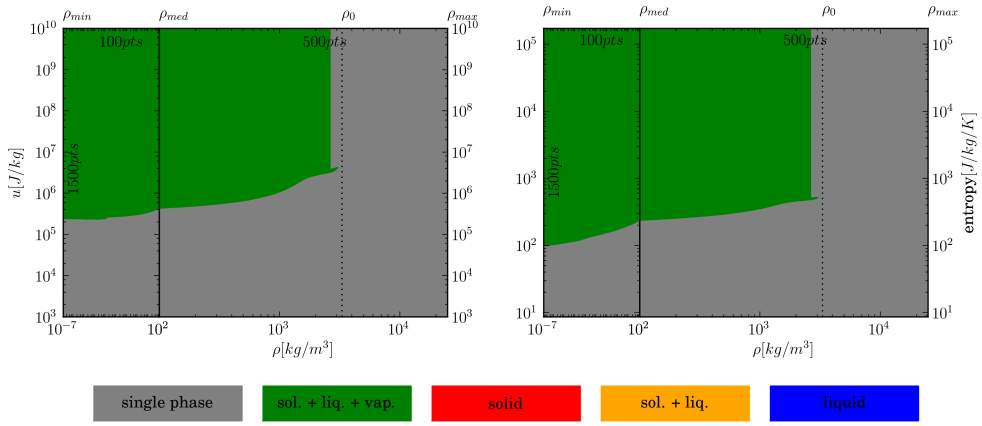

Figure 2.6: The same tables as in 2.4 for *dunite*. In the grey zones, ANEOS returns valid pressures and sound speeds, but does not return any particular phase information.

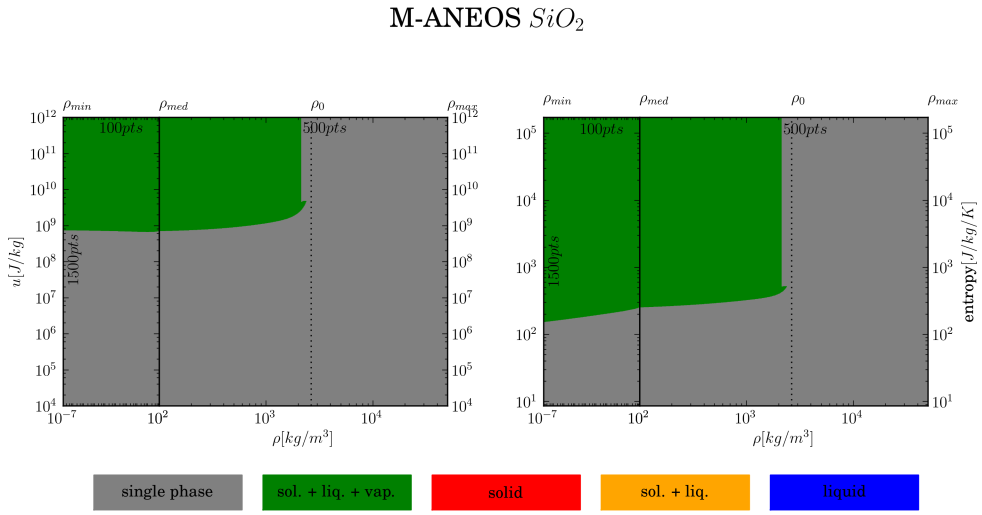

Figure 2.7: The same tables as in 2.4 for the modified M-ANEOS for SiO_2

given by

$$\Phi_j = -G \sum_{i \neq j} \frac{m_i}{r_{ij}} \qquad \boldsymbol{a}_j = -G \sum_{i \neq j} \frac{m_i}{r_{ij}^2} \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{2.59}$$

Calculating self-gravity with this sum is called direct-summation and involves for a each of N particles a sum with N force terms. So the complexity of this simple algorithm is $O(N^2)$. By making use of the symmetry of the force terms, only half of the force terms need to be evaluated, but the algorithm still retains its $O(N^2)$ complexity. This method is only practical if the problem can be satisfactorily modeled by a small number of particles (a few thousand) and specially designed computers are used (e.g. GRAPE or GPUs). An example of such an application is the evolution and accretion of a proto-lunar disk into a Moon as done by Kokubo et al. (2000). For problems using millions of particles, the direct summation is infeasible.

2.2.1 Multipole approximation

Different methods exist to avoid the prohibitively high computational complexity of the direct summation method mentioned above. They make use of two properties of a self-gravitating system of more or less equal mass particles: First, the strong r^{-2} dependence of the gravitational field on distance means that near particles with a certain mass exert a much larger attraction than particles further away with a comparable mass. Secondly, the field of more distant particles can be approximated as background field, which does not have to be resolved down to the contribution

of every single particle. Instead, distant particles can be summarized into clusters and their field be approximated as the one of single particles with the cluster's mass m_C at the cluster's center of mass $\boldsymbol{r}_C$:

$$\boldsymbol{a}_j = -G \sum_{i \,\epsilon\, C} \frac{m_i}{r_{ij}^2} \frac{\boldsymbol{r}_{ij}}{r_{ij}} \approx -G \frac{m_C}{r_{Cj}} \frac{\boldsymbol{r}_{Cj}}{r_{Cj}} \quad \text{when } r_{ij} \gg r_{jk} \quad \forall j,k \,\epsilon\, C \tag{2.60}$$

$$m_C = \sum_{i \,\epsilon\, C} m_i \quad \boldsymbol{r}_C = \frac{1}{m_C} \sum_{i \,\epsilon\, C} \boldsymbol{r}_i m_i \quad \boldsymbol{r}_{Cj} = \boldsymbol{r}_j - \boldsymbol{r}_C \tag{2.61}$$

This is the *monopole approximation*. In more general terms the total mass of the cluster is the rank-0 monopole tensor M with the center of mass X

$$M = \sum_c m_c \tag{2.62}$$

$$\boldsymbol{X} = \frac{1}{M} \sum_c m_c \boldsymbol{r}_c \qquad \text{for } M > 0 \tag{2.63}$$

We use the same indices as in McMillan and Aarseth (1993), where i, j number vector components and c indexes particles in a cluster or clusters in a group of clusters. The next highest non-zero multipole term is the traceless rank-2 quadrupole tensor

$$Q_{ij} = \sum_c m_c \big(3 r_{i,c} r_{j,c} - \boldsymbol{r}_c^2 \delta_{ij}\big) \tag{2.64}$$

where $r_{i,c} = \boldsymbol{X}_i - \boldsymbol{r}_{i,c}$ denotes the i-th component of the distance vector between a cluster's particle c and the center of mass of the cluster. The rank-3 octupole tensor can be reduced to a rank-2 tensor

$$S_{ij} = \sum_c m_c \big[5\big(3 - 2\delta_{ij}\big) r_{i,c}^2 - 3\boldsymbol{r}_c^2\big] r_{j,c} \tag{2.65}$$

and the single rank-0 component

$$S_{123} = 15 \sum_c m_c r_{1,c} r_{2,c} r_{3,c} \tag{2.66}$$

Higher order moments are usually not used and skipped here. The field of a group of cluster can be approximated by summing up the multipole moments of the individual clusters c. The mass can simply by summed up:

$$M = \sum_c M_c \tag{2.67}$$

The quadrupole tensor now also includes the quadrupole moments of the individual cluster:

$$Q_{ij} = \sum_c M_c\big(3r_{i,c}r_{j,c} - \boldsymbol{r}_c^2\delta_{ij}\big) + Q_{ij,c} \tag{2.68}$$

and finally the octupole tensor yields:

$$\begin{aligned} S_{ij} = &\sum_c M_c\big[5\big(3-2\delta_{i,j}\big)r_{i,c}^2 - 3\boldsymbol{r}_c^2\big]r_{j,c} + 5\big(1-\delta_{ij}\big)r_{i,c}Q_{ij,c} \\ &+ \frac{5}{2}r_{j,c}Q_{ii,c} - \sum_l \big[r_{l,c}Q_{jl,c}\big] + S_{ij,c} \end{aligned} \tag{2.69}$$

$$S_{123} = 15\sum_c M_c r_{1,c}r_{2,c}r_{3,c} + \frac{5}{3}\big(r_{1,c}Q_{23,c} + r_{2,c}Q_{31,c} + r_{3,c}Q_{12,c}\big) + S_{123,c} \tag{2.70}$$

The center of mass of the group of clusters is simply given by

$$\boldsymbol{X} = \frac{1}{M}\sum_c M_c\boldsymbol{X}_c \qquad M > 0 \tag{2.71}$$

The gravitational potential of a cluster can now be approximated by its multipole expansion:

$$\phi(\boldsymbol{r}) = G\Bigg(\underbrace{-\frac{M}{r}}_{monopole} \underbrace{-\frac{Q_{ij}}{r^3}\frac{r_i r_j}{2r^2}}_{quadrupole} \underbrace{-\frac{S_{ij}}{r^4}\frac{r_i^2 r_j}{2r^3} + \frac{S_{123}}{r^4}\frac{r_1 r_2 r_3}{2r^3}}_{octupole} + O\Big(\frac{1}{r^7}\Big)\Bigg) \tag{2.72}$$

and so the acceleration for a point mass at $\boldsymbol{r}$ in this potential yields

$$\begin{aligned} a_k = -\nabla_k\phi(\boldsymbol{r}) \approx -G\Bigg(&\underbrace{\frac{M}{r^2}\frac{r_k}{r}}_{monopole} + \underbrace{\frac{Q_{ij}}{r^4}\Big(\frac{\delta_{ik}r_j}{r} + \frac{5r_i r_j r_k}{2r^3}\Big)}_{quadrupole} \\ &+ \underbrace{\frac{S_{ij}}{r^5}\Big(\frac{\delta_{ik}r_i r_j}{r^2} + \frac{\delta_{jk}r_i^2}{2r^2} - \frac{7r_i^2 r_j r_k}{r^4}\Big) + \frac{S_{123}}{r^5}\Big(\frac{\delta_{1k}r_2 r_3 + \delta_{2k}r_3 r_1 + \delta_{3k}r_1 r_2}{2r^2} - \frac{7r_1 r_2 r_3 r_k}{2r^4}\Big)}_{octupole}\Bigg) \end{aligned} \tag{2.73}$$

Note that particles can be treated like clusters with vanishing multipole moments.

2.2.2 Barnes & Hut Tree algorithm

Barnes and Hut (1986) describe an algorithm, where particles are grouped into cluster with the help of a Tree. Before we look at the algorithm in detail, we need to discuss a few terms related

to trees: A graph is a set of elements represented by *nodes*. A tree is a rooted, directed graph in which any two nodes are connected by one and only one path. One node is defined as the *root node*. Each node can have a number of children[1] nodes and has exactly one parent, expect for the root node which has no parent. Nodes are located at a certain depth. The root node per definition has depth 0 and the children of a node inherit the depth of the parent increased by one. Figure 2.8 shows a simple tree. The blue square represents the root node, which has two children with depth 1. One of this children has again two children with depth 2. Tree data structures are usually drawn upside-down compared to real trees, this means with the root on top and depth increasing to the bottom. Nodes without children are called *leaf nodes*. A disjoint set of trees is called a *forest*.

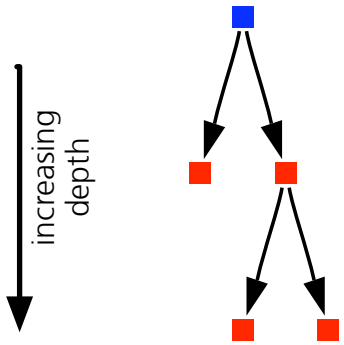

Figure 2.8: A simple tree. Squares represent nodes which are interconnected by arrows pointing from the parent node to its children. The blue square is called the root node and has depth 0.

One notices that the definition for a tree does not only match the whole set of nodes of a tree, but also subsets. Every node can be taken as a root node and forms again a tree, also called a *subtree* of the tree. Or one can say, that the children of a node form a forest of subtrees.

A tree data structure can be used to recursively subdivide space. The basic idea is to take a volume, assign this volume to a node in a tree, then subdivide this volume in disjoint sub-volumes and assign them to the children of the node. The subdivision of space can be mapped onto a tree. The easiest form of subdivision is using volumes which are aligned with the spatial dimension and have the same side length in each dimension. In 2D this corresponds to squares and in 3D to cubes along the axis. This volumes are then divided along each dimension into two equal-length intervals, which leads to 2^d sub-volumes in d dimensions. So if we have an arbitrary volume with a minimum $x_{i,min}$ and a maximum $x_{i,max}$ in each coordinate i we can enclose this volume with a

[1] often the term *daughter* is used, but because of the asexual nature of tree nodes we are using here the term *child*

volume with side length l and the center $x_{i,c}$

$$l = \max_i(x_{i,max} - x_{i,min}) \quad x_{i,center} = \frac{x_{i,min} + x_{i,max}}{2} \tag{2.74}$$

The 2^d subvolumes indexed with j have a side length of $\frac{l}{2}$ and center vector components $i = 0 \ldots d$, where d_i stands for i^{th} digit of the binary representation of the subvolume index j

$$x_{i,j\ center} = x_{i,center} + l\frac{(-1)^{d_i+1}}{2} \quad d_i = \frac{j - \sum_{k \neq i}(j \mod 2^k)}{2^i} \tag{2.75}$$

In 2D this sub-volumes are called *quadrants*, in 3D *octants*. The corresponding trees to which this subdivision can be mapped to are called *quad trees* and *octrees*, having 4 and 8 children respectively per node.

The algorithm described in Barnes and Hut (1986) subdivides the computational space containing N gravitationally interacting particles in sub-volumes, called *cells*. The particles are leaf nodes of cell nodes. The tree must be built in a way, that a sub-volume of a cell, an octant in 3D, contains no more than one particle. If it does, the sub-volume has to be subdivided into a number of cells until the requirement is fulfilled. An example of such a tree is shown in figure 2.9 for 50 randomly distributed particles in 2D. We note that no cell contains more than one particle. Shown below is the corresponding quad tree to this subdivision of space for the particles

Each cell in the tree represents now a clump of particles. For each cell a center of mass and its multipole moments can be calculated. This calculation is done *bottom-up*, so that the multipole moments of the children of node are already calculated, when the multipole moments of the node itself are to be calculated. This execution order can be assured with a pre-order tree walk, which will be discussed shortly. We finally end up with a tree of multipole moments for all the particle clumps, represented by subtrees in the tree.

For each particle the acceleration due to gravity can now be calculated according to equation 2.73. The multipole approximation only holds, when then size of the particle cluster is small compared to distance of the particle to the center of mass of the cluster $r_{part-cell}$. A good measure for the size of cluster is the cell size l in which it is contained. The cluster can not be bigger than the cell and it is usually also not smaller by magnitudes expect in pathological cases. The cell sizes get smaller by descending in the tree, so that depending on the particle-cell distance the cell size becomes small enough to use the multipole approximation. To decide whether it is necessary to descend further down in the tree, a *multipole acceptance criterion* short *MAC* is used. The one

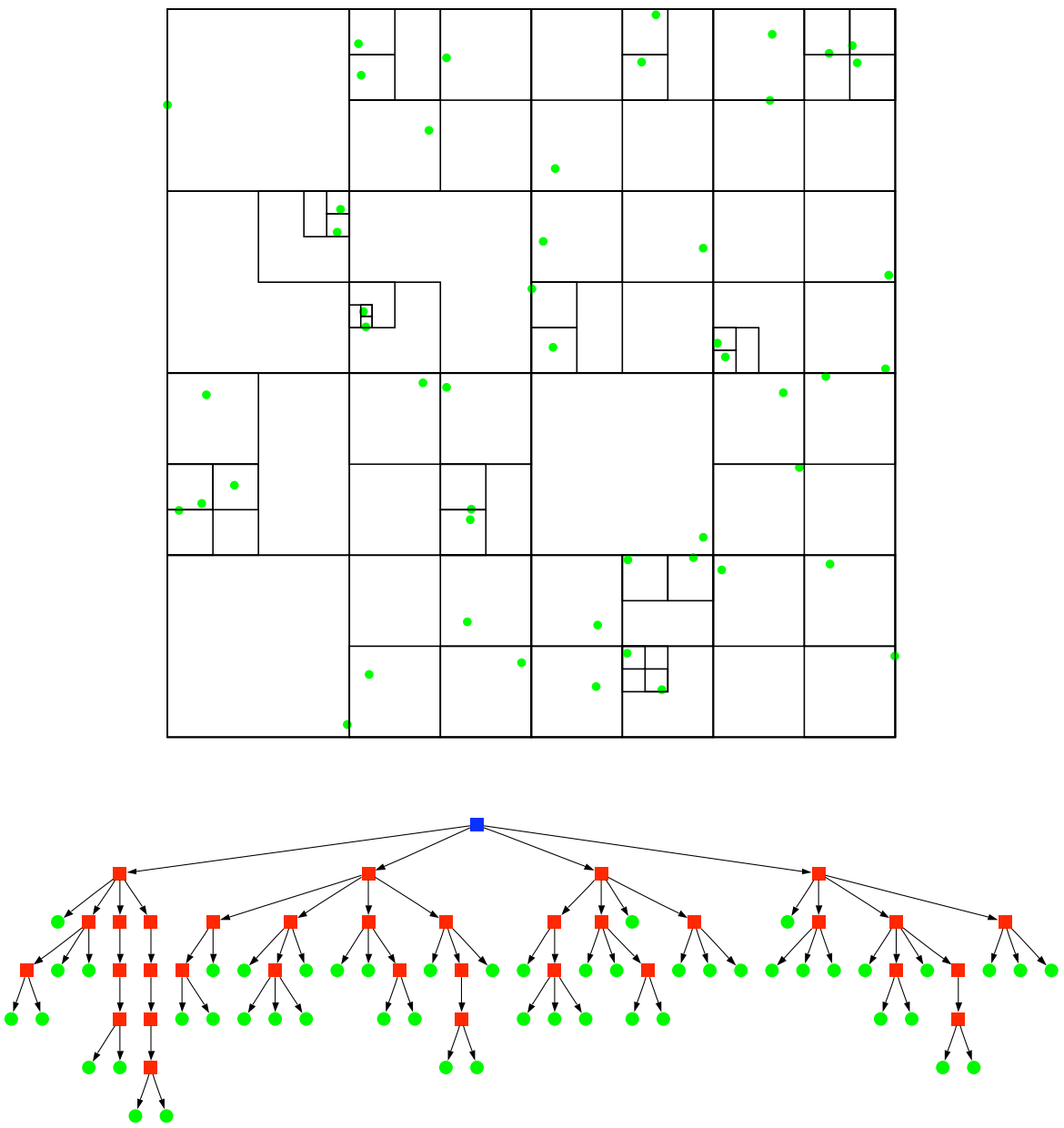

Figure 2.9: 50 particles randomly distributed in $2D$ and the subdivision into cells (upper plot). The corresponding quad-tree is shown in the figure below. Green dots the represent the particles, while red squares indicate cell nodes. The root cell node is highlighted as a blue square.

proposed by Barnes and Hut (1986) is

$$\frac{l}{r_{part-cell}} \leq \theta \quad \theta = 0.6 \ldots 1.0 \quad r_{part-cell} = |\boldsymbol{X} - \boldsymbol{r}_{part}| \tag{2.76}$$

where θ is the *opening angle* chosen depending on the desired accuracy.

The smaller the opening angle, the more accurate the calculation becomes. When $\theta = 0$ no particle cluster cell fulfills the MAC and the tree walk is completed down to each particle, leading again to the direct summation with complexity O(N^2). For $\theta \geq 1$ an additional error might be introduced, as depending on the distance between the particle and the center of mass of its own cluster, the MAC may be fulfilled, leading to a acceleration calculation based on the multipole moments which include the particle itself. This error is called *self-acceleration*. The choice of the MAC determines directly the efficiency and precision of the Barnes & Hut method. Alternative MACs are given in the literature, e.g. in Stadel (2001).

Figure 2.10 shows the relative acceleration errors and the speed-up for a self-gravitating sphere when using the Barnes & Hut tree instead of direct summation. The errors are in the range of 10^{-3} for most particles, while the speed-up is considerably even for moderate number of particles.

2.2.3 Implementing the Barnes & Hut tree

The actual implementation of the Barnes & Hut method described above consists of two main parts: The tree data structure and the algorithms which are applied onto the data structure.

All the information of the tree lies in the nodes: They contain information about their connection to the parent node and to possible children. Additionally they also contain a payload, the information about the volume in space they represent, like the center, the center of mass and the multipole moments of the particles contained in the cell.

In modern programming languages like FORTRAN or C, the most common approach for implementing a tree data structure, is to define node data structures and then dynamically allocate them in memory. Connections between the nodes can be realized with pointers pointing to other nodes in the tree. Walks in the tree can then be performed by following those pointer from node to node. Figure 2.11 shows the content of a cell node and two particle nodes in an octree. Not shown in this figure is the payload data. All nodes have a pointer to the parent node, so that also upward walks in the tree are possible. The cell node additionally has eight pointers to possible children. If there is no child for the corresponding sub-volume, the pointer has a special value (e.g. NULL) so that it becomes clear that the pointer is invalid. The *next* and *skip* pointers will be explained later.

Tree walks can now be performed in such a tree data structure by defining the *current pointer*,

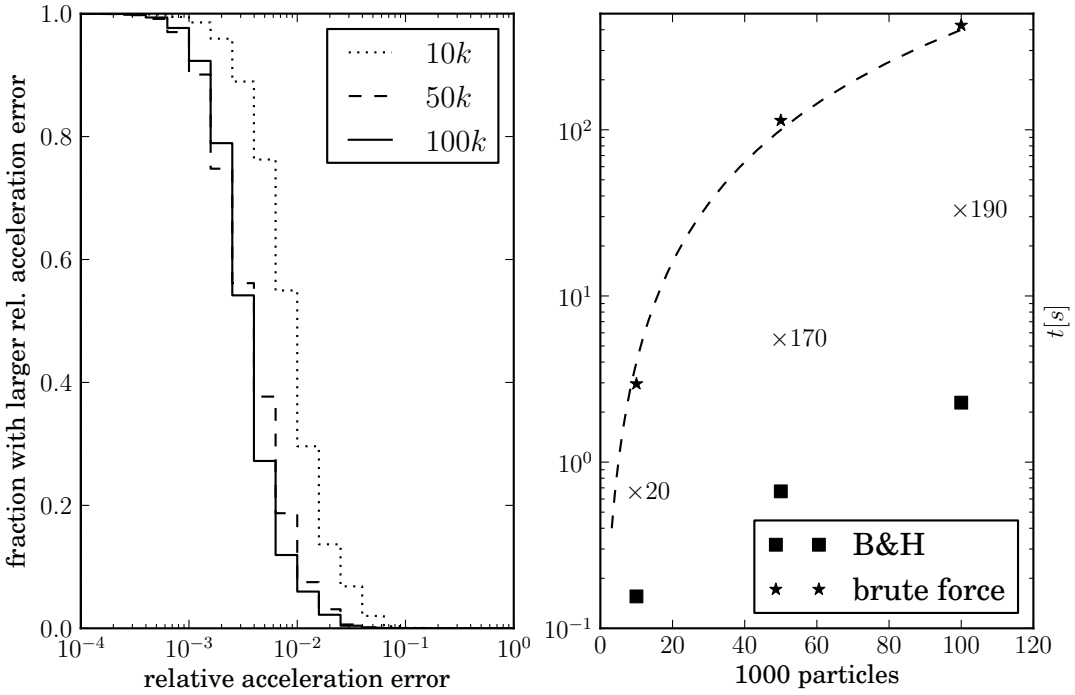

Figure 2.10: The left plot shows the relative acceleration errors, due to the multipole approximation $\Delta_{\boldsymbol{a}_{mp}} = \frac{|\boldsymbol{a}_{mp}|-|\boldsymbol{a}|}{|\boldsymbol{a}|}$ for a self-gravitating spheres using an opening angle $\theta = 0.7$. For all three resolutions, the relative errors are below 10^{-3} for more than 90% of the particles. The right plot shows the execution time for one self-gravity run for the Barnes & Hut tree and the brute force direct summation for different numbers of particles and an opening angle of $\theta = 0.7$. The code used is the SPHLATCH v2 implementation discussed below with one CPU. Direct summation execution time follows a $O(N^2)$ scaling (dashed line), while the Barnes & Hut tree shows a more complex scaling. Speed-ups ($\times 20 - 190$) are considerably even for moderate number of particles.

a variable which points to the current node being processed in a tree walk. By setting this pointer to the parent node for example, an upward movement can be performed. Similarly setting the current pointer to a child corresponds to moving down in the tree.

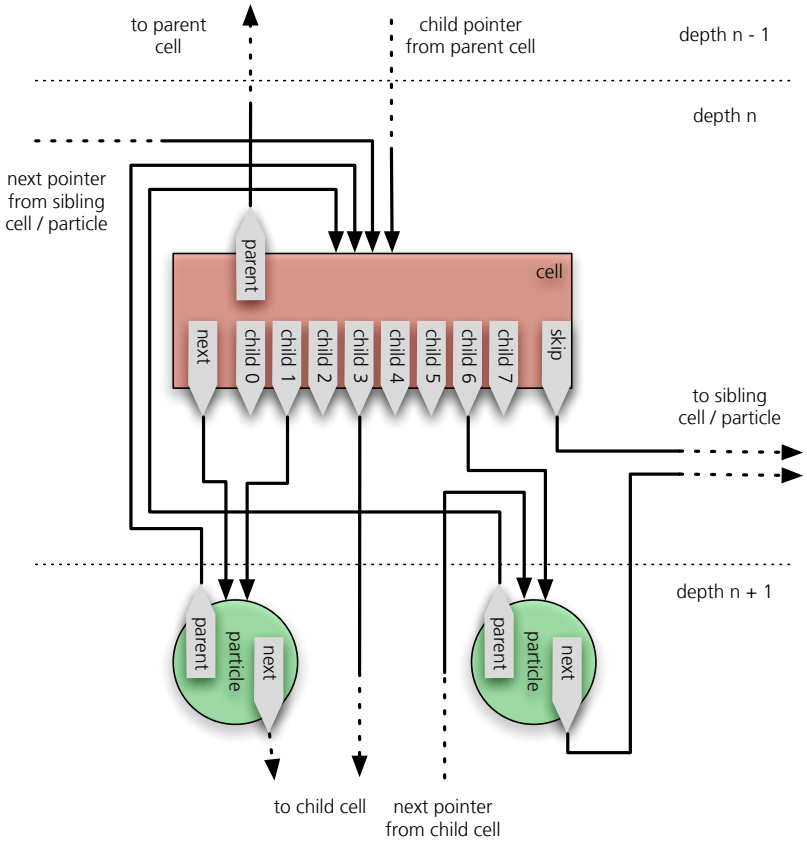

Figure 2.11: Wiring scheme of an octree cell node a depth n with particle children in sub-volume 1 and 6 and a cell node as child 3. Parent pointers allow going upwards in the tree, child pointers downwards. Following the next pointers results in a pre-order tree traversal, where taking the *skip* pointer skips a cells subtree.

With the root node as the starting point, there exist two particular ways of traversing the tree downward and applying an action to every encountered node: *Pre-order* and *post-order* traversal. Both traversals can be realized with simple, recursive algorithms. Figure 2.12 shows the two algorithms: The post-order algorithm first executes itself on the children and performs the action after that. So the action is always performed on a node, before it is performed on its parent. This algorithm is therefore suited for calculations like the multipole summation, where the multipole moments of a cell depend on the moments of its children. The pre-order algorithm works exactly the opposite way: First the action is performed on the parent node and only after that the algorithm executes itself recursively on the children.

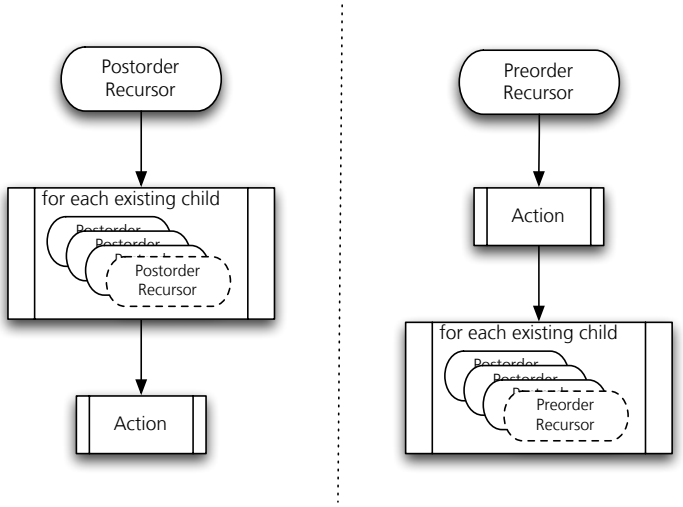

Figure 2.12: Post-order vs. pre-order recursors

The Barnes & Hut tree gravity calculation method requires the following tree algorithms: Inserting particles into the tree, calculating multipole moments, approximating the self-gravity forces and finally the tree also needs to be deleted again. All those algorithms can be derived from two walks presented above.

Figure 2.13 shows the particle insertion algorithm. For a particle which has not yet been inserted into the tree, the walk starts at the root node. The sub-volume where the particle is included is checked for an existing node. If there is no node, a new particle node is created there and the algorithm terminates. If there already exists a cell node, the algorithm simply opens the cell and executes itself at the current position. The third possibility is that there already exists a particle node for a particle B. In that case the particle node is replaced by a cell node and the particle B is placed as a child of the new cell. The particle inserter is then run again at the position of the new cell. This algorithm has complexity $O(N \log N)$ for N particles, as it has to be executed N times and the depth of the tree is in the order of $O(\log N)$.

As mentioned before, the multipole moments can simply be calculated by a recursive post-order algorithm. Figure 2.14 shows the corresponding algorithm on the left side. Tree deletion can also be accomplished by using a post-order recursive algorithm, which can be seen on the right side of figure 2.14. Deleting the children of a cell before the cell itself guarantees, that all cells are still connected to the tree until they are deleted.

Recursive algorithms are implemented in a procedural language like C or FORTRAN, as functions calling themselves recursively. While implementing tree algorithms in a recursive way results in simple algorithms, it introduces a performance penalty. Calling a procedure includes a

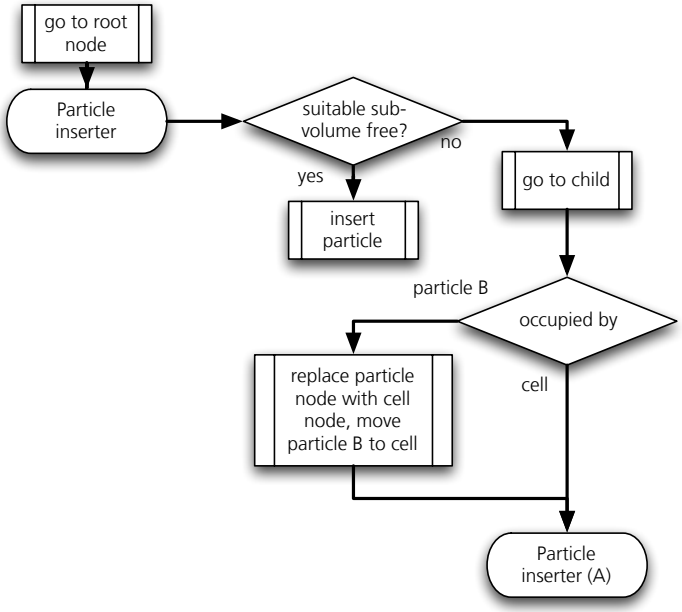

Figure 2.13: Algorithm for the insertion of particle A.

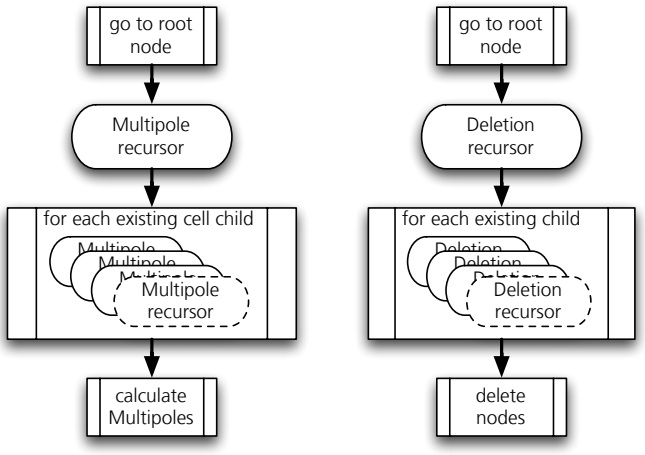

Figure 2.14: Recursive algorithms for calculating the multipole moments in the tree and for deleting all tree nodes. Note that the recursive part of the algorithm does not include going to the root node.

computational overhead (Bryant and O'Hallaron 2010). It is therefore desirable to implement the most costly tree algorithms in a non-recursive way. This can be done with the introduction of *next*- and *skip* pointers. The basic idea is to store the next node a certain tree walk would end up at for each node. Figure 2.15 shows an example of a small tree. The grey arrows depict the connections between the tree nodes. The black arrows in the middle part of the figure now show the *next*-pointers, which point to the next node in a tree walk compromising all tree nodes. Note that all tree nodes have a *next*-pointer. Another useful pointer for the Barnes & Hut algorithm is the *skip*-pointer. It points to the next node the tree walk leads too, if the cell node is accepted by the multipole acceptance criterion and its subtree is therefore skipped. The right part of figure **??** shows the skip pointers for the small example tree. A *skip*-pointer of a cell always points to a sibling node at the same or a smaller depth than the cell. It is crucial, that it only skips the subtree of its own cell and no other cells.

The two most costly tree algorithms, the gravity walk and the neighbor search for SPH, can now be realized in a non-recursive way with the help of this additional node pointers. Figure 2.16 shows the flow diagram for the gravity walk for evaluating the gravity acceleration on particle A. The current pointer is set to the root node and the walk starts. First it is checked, whether the current pointer still points to a valid node. If not the walk has terminated. If the node is valid, the type of the node is checked. If the node is a cell node, then the multipole acceptance criterion is evaluated. In case the MAC is fulfilled, the multipole moments of the cell are used to approximate the gravity force all particles represented by this cell node onto particle A. After that, following the *skip* pointer leads to the next node in the tree. In case the evaluated node is a particle, its interaction with particle A is calculated, when the particle node does not correspond to the one representing particle A itself. This algorithm has the complexity $O(\log N)$ for one particle and $O(N \log N)$, although Barnes and Hut (1986) note that the scaling in reality is more like $O(N)$

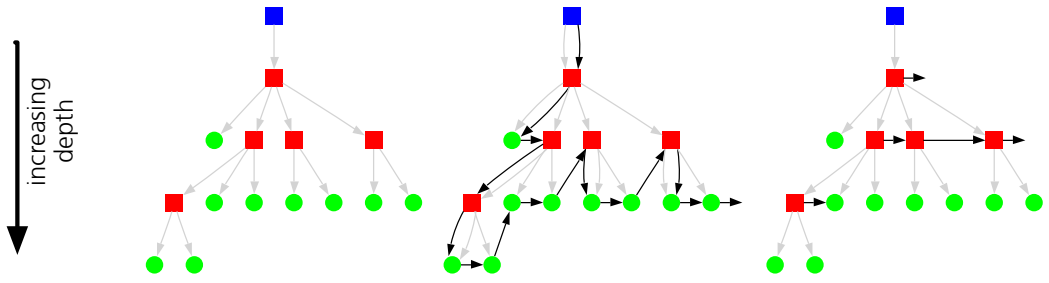

Figure 2.15: An excerpt from a tree with particles at different depths. Grey arrows depict the connections between the tree nodes. In the middle plot, the *next*-pointers are shown. Following them results in a post-order tree walk along all nodes. The very last *next*-pointer The right plot shows the *next*-pointers, lead to the next node with an equal or smaller depth. This *next*- and *skip* pointers allow tree walks to be implemented with non-recursive algorithms.

for N particles.

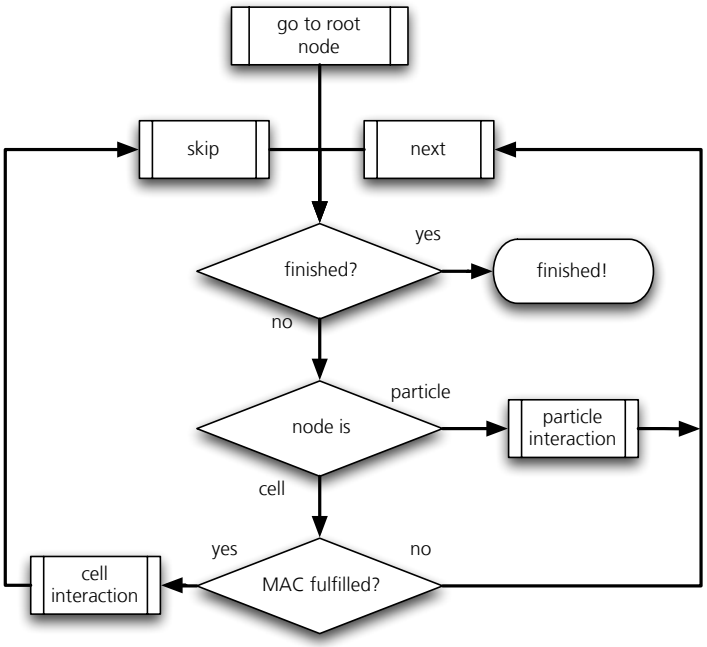

Figure 2.16: A non-recursive algorithm for the Barnes & Hut tree walk.

2.2.4 Neighbor search with the tree

Searching for particle A's neighbors can be done as shown in figure 2.17. The basic idea of the algorithm is to first find the cell which fully encloses the search sphere around the particle for which the neighbors should be found. This cell is then the root of the subtree in which all the neighbors are included. The algorithm starts by setting the current pointer to the particle node of particle A. The tree is traversed upwards until a cell node is found, which fully enclosed the search sphere given by the particles position and the search radius around it. Whether this criterion is fulfilled can be checked by comparing the edges of the cell to the most extreme coordinate of the search sphere in each dimension. From here on the subtree is searched for neighbors. If a particle node is encountered, its distance to the original particle A is checked and if it's inside the search radius, the neighbor function is executed for this particle node. In case the neighbor search is used for SPH, this function is simply a sum term executed on the neighbor. Alternatively if an actual list of neighbors is required, the function simply adds the neighbor particle to a list. If the node in the subtree search is a cell node, it is checked, whether the cell is completely outside the search sphere. If so, the cell is skipped, if not it is opened by following the *next*-pointer. The algorithm

has finished, if the search subtree has been fully traversed. This is the case, if the current pointer points to the *skip*-pointer of the search subtree root node, the one which fully enclosed the search sphere. A nice property of this search algorithm is that its complexity is constant. The size of the search subtree does only depend on the size of the search sphere and not on the size of the total tree, or in other words on the total number of particles. Neglecting performance penalties due to caching issues, the algorithm has therefore the same performance independent of the total number of particles. Note that this scaling is only possible, because the position of particle's A node is already known. If this node would have to be found first, the total size of the tree would indeed matter. Storing the position of the particle's tree node only adds the penalty of storing an extra pointer into the tree. This penalty is small compared to the gain in speed.

The neighbor search algorithm is can also be used for other tasks, than searching for the SPH neighbors. Other applications are friend-of-friend algorithms as presented later on or the initial estimate of the smoothing length, the inverse problem of finding the neighbors for a given search radius.

2.3 Implementation: SPHLATCH

In this section we will discuss the implementation of the SPH and selfgravity tree code SPHLATCH. Figure 2.18 shows the basic outline of a standard gravity and hydrodynamics particle code. In a first step, the particles are loaded. Other data like tables for the equations of state are loaded as well. After that the main loop starts, which integrates the particles forward in time. Some integrators like the Predictor-Corrector scheme from equations 2.39 - 2.43 first determine the maximum allowed time step. Then the derivatives are calculated several times, two times in case of the Predictor-Corrector. At certain points in time, special tasks like storing the particles or some post-processing is done. When the particles have been integrated far enough in time, the code finally stops.

2.3.1 Tree and code parallelization: shared vs. distributed memory

Since the introduction of electronic computers based on semiconductors, the transistor count on a microchip doubled every 18 months (Moore 1965). Similarly also the performance, measured in FLOPS (floating point operations per second), increased over time. Until the late 1990ies this performance increase could be guaranteed by simply increasing the clocking frequency and complexity of the CPUs. After that physical limits upon the frequency and the size of semiconductor

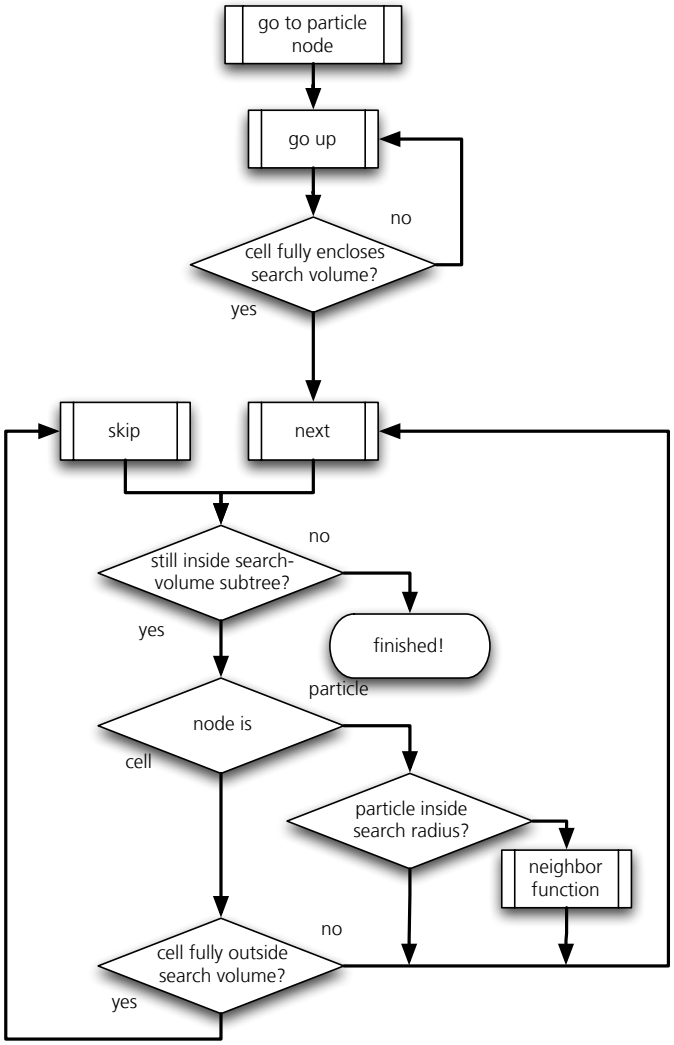

Figure 2.17: A non-recursive algorithm

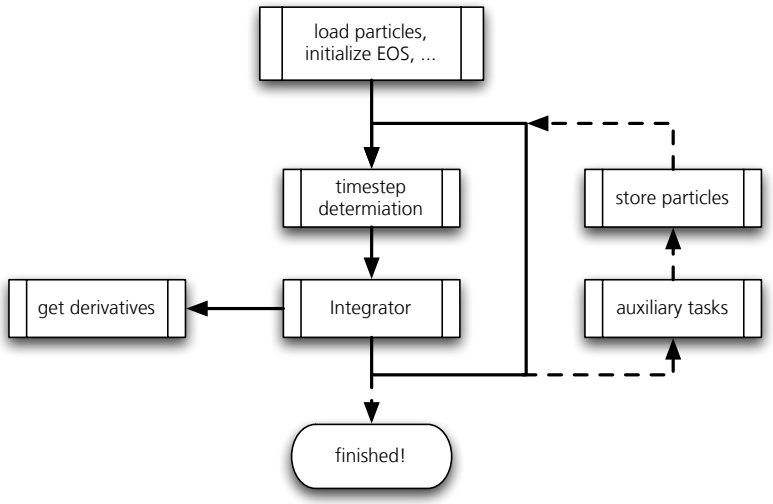

Figure 2.18: Overview of the tasks in a common particle code. The main loop consists of the integrator evolving the particles forward in time. The most time consuming part of the code is the derivation step, where derivatives of integrated values are calculated.

structures capped the performance increase of a single CPU [2]. So the computer industry started to face this problem by simply increasing the number of CPU cores per machine. The increased performance can therefore only be harvested, if a program is able to run in *parallel* on several CPU cores. There exist basically two types of parallelization: Distributed memory computing is based on several computers with independent (distributed) memory, but with a common network over which they can exchange data. A popular approach in scientific computing is MPI, a software library which provides functions to easily exchange or process numerical data. Often high speed networks like Ethernet or Infiniband are used as a transport media. Although it is non-trivial to parallelize a computer code with MPI, it provides a straightforward solution for using hundreds of CPUs at the same time simply by interconnecting a large number of cheap computers with a network. The first implementation of the code SPHLATCH presented here uses MPI for parallelization.

Shared memory computing on the other hand is based on several CPU cores sharing a common memory space. Data is simply exchanged by sharing certain parts of memory. A popularly used implementation in scientific computing is OpenMP, a simple set of compiler directives parallelizing loops and vector operations. The program runs as one process, but certain tasks are subdivided into threads, which can run on individual CPU cores. The disadvantage of this ap-

[2] A single CPU is a unit capable of running a computer program independently. In other words, two CPUs can run two programs at the same time.

proach is, that the execution of the program is limited onto one computer. Only as much CPU cores can be used, as a single computer has. While this was a limitation a decade ago, when affordable computers did not have more than two or four CPU cores, nowadays machines offer a much larger number of cores. As of 2011, server machines with 32 or 48 CPU cores are no exception and very offardable. The second implementation of SPHLATCH uses OpenMP for parallelization.

2.3.2 SPHLATCH v1

The first version of SPHLATCH uses a distributed memory parallelization by employing the MPI library. As a first order approximation, we can assume, that the computational cost to calculate the SPH-sums and gravity forces is more or less the same for all particles. Therefore a standard approach in parallelizing a set of particles is to distribute them equally onto computational domains, called *cost zones*. Each cost zone is a sub-volume of the whole simulation space and assigned to a processor. Integration of the particles is done locally in each cost-zone, so this means all forces and other derivatives for integration have to be collected in each cost-zone. Particles of course not only interact with other particles in the same computational domain. For short range forces like hydrodynamics this might be only the case for particles at the border of computational domains, for example when neighbors for a SPH sum actually come to lie in a different computational domain. For long-range forces like gravity, of all particles depend on information about other particles in other computational domains. The difficult part in a distributed memory parallelization is now to provide all the necessary information to local particles in a cost zone, without having to share all data about the remote particles.

The decomposition of the complete computational volume into individual domains can be done in any way, but geometrical calculations are simplified a lot if space is decomposed into regularly spaced and equally sized volumes. The cells in the tree for the Barnes & Hut algorithm shown before fulfill this criterion, which makes this tree a good choice also for spatial decomposition. Figure 2.19 shows the same particle set as in figure 2.9 together with the corresponding tree structure. We now choose the cell nodes at a certain depth and use those cells as buidling blocks for the computational domain. This depth is called the *cost zone depth* d_{cz}. The 2D example in figure 2.19 uses a cost zone depth of 2, resulting in 16 cost zone cells. The side length of these cells is $l/2^{d_{cz}}$, if l is the side length of the root node cell enclosing all particles. The green shaded cells now indicate an example of a cost zone. All green particles are considered to be local to this computational domain. If we assume that no particle has neighbors which are more distant to the particle than the side length of the cost zone cells, then local particles only interact with remote particles in cost zone cells directly adjacent to the local cost zone cells. In the figure such cells are shaded grey. Particles in those cells are called *ghost particles* or just *ghosts*. In case the short-range

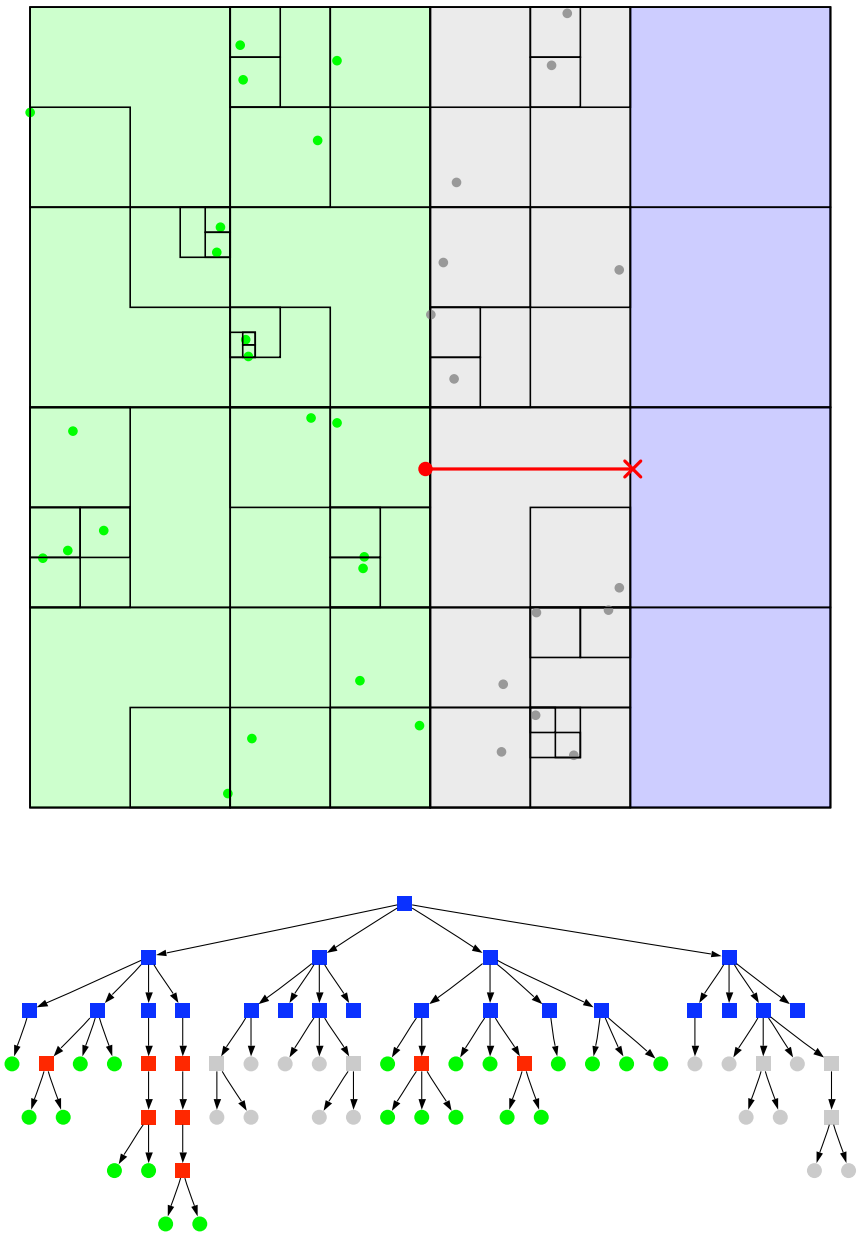

Figure 2.19: A part of the same particle distribution like in figure 2.9. The local domain is shaded green and contains 21 particles for which the acceleration has to be calculated. Bordering this zone is the ghost domain shaded grey, for which all particles are known. Beyond that, no particle information is available, only the corresponding parts of the global tree down to top-tree depth are known here. The plot below shows the corresponding tree. Note the filled top-tree nodes without any children nodes, they correspond to the domain where no particle information is available.

forces are realized as SPH sums with a kernel with compact support, this requires $2h_i < l/2^{d_{cz}}$ for all particles.

There exist basically two ways of summing up the interaction terms between local and ghost particles. One way adopted for example in the GADGET code (Springel 2005), is to send the necessary information about local particles to the remote domain. There the partial SPH sum terms caused by the ghosts are calculated and sent back to the local domain, where this partial SPH sums are added up to the local particles. This is the *distributed sums* approach. Another strategy is to store all necessary information about the ghost particles locally and use them like local particles, without exchanging sums. This is the *ghost* approach which we adopt in our code. Both approaches require the same amount of sum terms and differ just in the way they are remotely added up. Note although that in the distributed sums approach, the partial sums also need to be copied back again to the domain where the particle resides. Furthermore the neighbor search algorithm shown above requires, that the particle for which the neighbors shall be found, actually is also represented in the tree. This would not be the case in the distributed sums approach, requiring a different neighbor search algorithm.

It is desirable to have as little ghost particles as possible. The way the computational domains and subsequently are made up by combining the cost-zone volumes determine the number of ghost. Generally it is best to minimize the surface of the computational domains. Composing the domains by cutting space-filling curves like the *Peano-Hilbert-walk* along the cost-zone volumes into intervals leads usually to good results (Springel 2005).

Parallelizing the gravity calculation with a Barnes &Hit tree is not as straightforward like parallelizing a particle calculation for short-range forces, where interaction with distant non-local particles just can be omitted. The crucial point is to know, which parts of the *global tree* generated by all the particles the local particles need to know about. The tree known to the local particles is called the *local tree* and consists of the tree built by local particles and ghosts. It differs from the *global tree* as such as it does not contain sub-trees not required by the local particles. The sub-trees which can be omitted are easily identified by the MAC and a worst-case scenario for the position of a local and a non-local particle it has to interact with as shown in figure 2.19. Assume the gravity has to be calculated for a particle right at the edge of the computational domain, marked in the figure with a red dot. The shortest distance between this particle and any point in the non-local domain for which no particle information is available (marked in the figure with a red line) is the side length of the cost zone volumes or in other words the distance between the particle any point in the non-local domain r_{WC} is always bigger than this side length:

$$r_{WC} \geq \frac{l}{2^{d_{CZ}}} \tag{2.77}$$

The MAC now tells us how big a cell might be, in order to be still accepted for the gravity calculation given the worst case distance. Or in other words, it tells us how deep we have to go in the tree until the multipole approximation is acceptable. Taking the worst case with distance r_{WC} we get the deepest depth we have to go in the tree for a point in the non-local domain, we'll call that the *top-tree depth* d_{TT}

$$\frac{l_{cell}}{r_{WC}} = \frac{l}{r_{WC}2^{d_{TT}}} \geq \theta \tag{2.78}$$

The top-tree depth tells us down to which level the global tree has to be known on every computational domain. This part of the tree is the top tree. Any tree structure below that depth is either provided by the local or ghost particles, or is not needed locally. Combining the two equations we get a relation between the top-tree depth, the cost zone depth and the opening angle from the MAC:

$$2^{d_{CZ}-d_{TT}} \leq \theta \quad \text{or} \quad d_{TT} = d_{CZ} + \uparrow \left(-\frac{\log\theta}{\log 2} \right) \tag{2.79}$$

The first expression tells us down to which opening angle the local tree provides enough information for a given cost zone and top tree depth. The latter expression gives the minimal required top tree depth for a given cost zone depth and opening angle. The rounding up is because depths are always integer numbers. The example shown in figure 2.19 uses a top-tree depth of 2 and also a cost zone depth of 2, so $\theta \geq 1$. In order to use a smaller opening angle we would have to increase the top tree depth.

So we know now which parts of the global tree need to be in the local tree. The next question is how to build the local tree. On no process the global tree can be built, as no process knows about all the particles and also the global tree might be too big to fit into the memory available to a process. The basic idea is to build the local tree with local and ghost particles and then synchronize the top tree globally.

Synchronizing the top-tree is per se non-trivial, as a process does not know which parts of the non-local domain contain particles and therefore need nodes to represent them and which don't. The easiest solution is to build a full top tree, this means each node in the top tree contains the maximum possible number of children. In 2D this corresponds to a full quad-tree, in 3D to a full octree. The problem is now, that for non-uniform particle distributions we get many top-tree nodes not containing any particles in or below them, which will lead to unnecessary walks along them. The calculation does not get incorrect, but the adding up of vanishing terms leads to a higher computational cost. For that reason we introduce an *emptiness* flag to each cell node. A priori a cell nodes are empty. Only when a particle is inserted the cell node becomes un-empty or in other words filled. Emptiness is inherited bottom-up from the children to the parent node in a logically negative way, this means when a node has any non-empty child, it becomes non-empty.

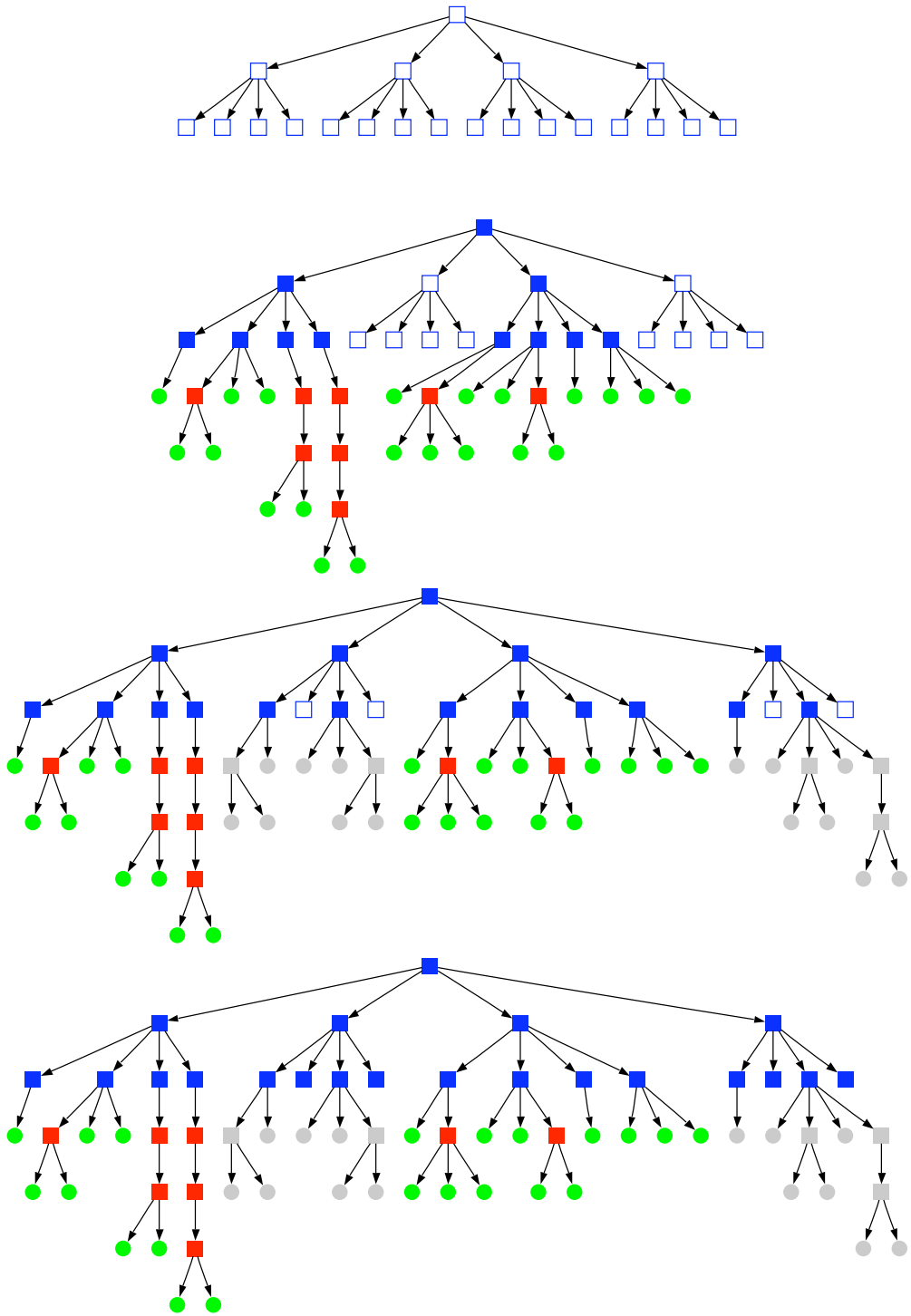

Figure 2.20: Building stages of the local tree for the computational domain in figure 2.19. The first tree shows the empty top-tree, a full quad-tree with depth 2. Then the local and the ghost particles are inserted leading to the second and third tree. As a final step the top-tree is summed up globally, filling up the 4 previously empty cell nodes. Top-tree cell nodes are coloured blue, normal cell nodes red and particles green. Ghost particles or cells are coloured grey. Non-filled shapes represent empty nodes.

This flag can now be queried during tree-walks in order to avoid empty walking along empty sub-trees. So we start building the tree by setting up a full top tree consisting of empty nodes. The first tree in figure 2.20 shows such an empty quad tree with a top tree depth of two. Every domain now has a top tree of the same structure and therefore we avoid synchronizing different tree structures between the processes.

During the next step, the particles locally known to a process, namely the local and ghost particles, are added to the local tree, inserting cell nodes where needed (second tree in figure 2.20). After that the multipole moments can be calculated bottom-up from the children to their parents. This is a little bit tricky again, as a particle may be present on different process at the same time: On the local domain as a local particles and on remote domains as a ghost. We have to make sure, that such particles contribute to the multipole moments of the tree only once. But we also have to make sure, that the ghost particles contribute to the multipole moments of a ghost cell node. An example of such a ghost cell node can be seen in the third and fourth tree in figure 2.20 represented by grey squares. This ghost cell nodes may also be used for the acceleration calculation and therefore have to contain the correct multipole moments. For this reason we introduce a *locality* flag to each node. A local particle has the locality *local*, a ghost particle the locality *remote*. Locality is inherited bottom-up from the children to their parent nodes in a positive sense. If one children is local, the parent node is also local. Only when all children are remote, the parent is remote. Top-tree nodes are excluded from this rule and have always the locality local. So when calculating the multipole moments of a node, we apply the rule that only nodes with the same locality contribute to the multipole moments. With that rule we fulfill both requirements mentioned above: Ghost cell and particle nodes do not contribute to the globally synchronized top-tree, but ghost cell nodes still get their correct multipole moments based on their remote or ghost children.

After adding up the multipoles, we can sum up the multipole moments of the top-tree. This is straightforward, as all top-trees have the same structure. The emptiness flag is summed up using an OR-relation, respecting the additive nature of multipole moments addition. After this step we end up with a valid Barnes& Hut tree containing all the necessary particles and cells required to calculate the gravitational acceleration for the local particles with a given MAC.

This parallelization method is rather simple, which is one of the advantages of it. Another advantage is, that after the particles and ghosts have been copied to the processes, only one, big additional communication step is required for summing up the multipole moments of the top-tree nodes. When summing up the multipole moments between two domains, all the data can be sent to one domain and summed up there. This is bulk communication and therefore ideal for communication networks with high bandwidth. Latency is not important. In practice, the synchronization needs only a few percent of the total time of the gravity calculation using a reasonably fast network.

The downside of the method is that for non-uniform particle distribution a lot of unused and empty top-tree nodes have to be created. Although they don't need to be accessed, they use memory and make the caching of other tree nodes worse. Enforcing cell nodes in the top tree also causes a deeper tree than without the top tree, which leads to longer and therefore slightly slower tree walks.

Figure 2.21 now shows a summary of the building blocks of the parallel implementation of the derivation step: After determining the cost zones, the particles are exchanged to their assigned domain. Note that this is not only necessary at the very first derivation step, but at each step, as particles do move and change the computational domains. Additionally, computational domains themselves are adapted after each derivation step. In the next step, information about the ghost particles is exchanged, which is required for building the local tree and for the first SPH sum. Apart from position and mass, which is required for building the tree, other variables are required. For example if the first SPH sum to be evaluated is the density sum, the ghost particles also need to have information about their smoothing length. After building the local tree by inserting local and ghost particles, the top tree is globally summed up, which requires global summation of the multipole moments of the top tree. At this stage, everything is ready for calculating the derivatives. SPH sums are calculated in the order as shown in table **??**. The order is given by the dependencies of the variables. In standard SPH for example, the density is calculated first, as all other SPH require the density of the neighbors as part of the sums. Note that each time a variable is newly calculated and required in a later expression for a derivative, this variable also needs to be copied to the remote ghost particles. The gravitational acceleration and/or potential is also calculated during the derivation step.

The other parts in the code as shown in figure 2.18 are parallelized in a straightforward way in this implementation of SPHLATCH. Loading the particles from a dump file in the parallel version differs as such, as only a part of the complete dump has to be loaded. It doesn't matter which part is loaded, as particles are exchanged into their proper cost zones anyway in the first derivation step. Determining the time step with the various criteria (equations 2.45, 2.46) is changed only as such, as the minima have to be found globally. Integrating the derivatives of the local particles can be done like in a serial code. Writing the particles into a dump file in a parallel can be a bit tricky. Writing in parallel to a single file can lead to different behavior depending on system libraries. Our code circumvents this problem by using the HDF5 library[3], which is supports parallel writes. Dump files are stored in a compatible way like in the H5PART[4] library, which allows the use of existing software for analysis and post-processing. Apart from particle data, this file format

[3] http://www.hdfgroup.org/HDF5/

[4] http://vis.lbl.gov/Research/H5Part/

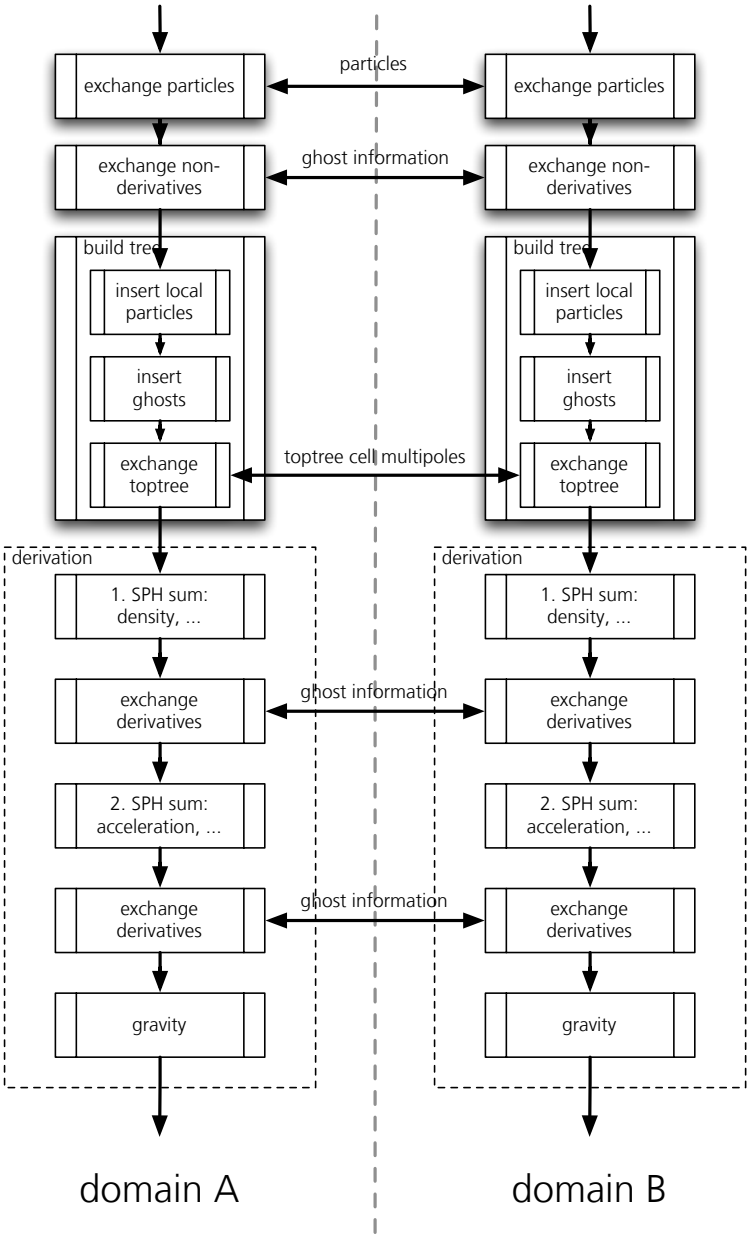

Figure 2.21: Overview of the derivation step for SPHLATCH v1

also supports attributes which can be used for example to store simulation constants, like the gravitational constant or the courant number.

This distributed memory parallelization shown here works sufficiently well for problems where particles are relatively homogeneously distributed. Other codes like GADGET or GASOLINE use a very similar parallelization approach and successfully simulated cosmological problems with billions of particles Springel et al. (2005). While cosmological simulations show large spatial dynamics by collapse, the particles are overall relatively well distributed in space, which allows this parallelization approach to scale well up to large numbers of particles. In simulations of collisions, the particles are not well distributed: In the beginning particles are in two well separated bodies. After the collisions small fragments may fly far away thus enlarging the simulation space, while most particles are still concentrated in one or two large bodies. That the approach shown before is not capable of distributing the particles well onto different processors can be seen through a quick estimate: With the multipole acceptance criterion from equation 2.76 with an opening angle between $0.6 \ldots 1.0$, we have seen that the top tree depth d_{TT} has to be one larger than the cost zone depth d_{cz}. A top tree depth of 6 leads to $8^6 = 262144$ top tree cells, which for practical reasons is an upper limit. This leaves us with a cost zone depth of 5 and 32 cost zone cells in each dimension. Now let's assume a situation in which a large, spherical central body in which most particles are located and some debris distributed around it no further than $10\times$ the central bodies diameter. Figure 2.22 shows a similar situation where the two original bodies of the collision with a mass ratio of 3:1 largely remained intact, but where large amounts of debris are generated. So in this example roughly 1.5 cost zone cell side lengths equal the central body diameter. The central body will therefore lie in no more than 8 cost zone cells. Not only is it now impossible to distribute the particles to a large number of computational domains, it is not even possible to distribute the particles evenly to a small number of domains, as the granularity of the cost zone cells is too coarse to balance the domains. This is the reason why this approach of distributing particles does not work well for problems with un-even particles distributions. Not that in practice debris can be located much further out, making the problem worse.

Another disadvantage of this approach is the computational overhead of ghost particles. In practice there are usually twice the number of ghost particles on a domain than local particles. Not only uses this a lot of memory, but communicating ghost information introduces a large overhead.

2.3.3 SPHLATCH v2

The disadvantages of SPHLATCH v1 and the demand for a fast code suited for simulations of medium resolution collisions led to the development of SPHLATCH v2, a shared memory code. The big difference to the first version, is that not the number of particles is distributed amongst

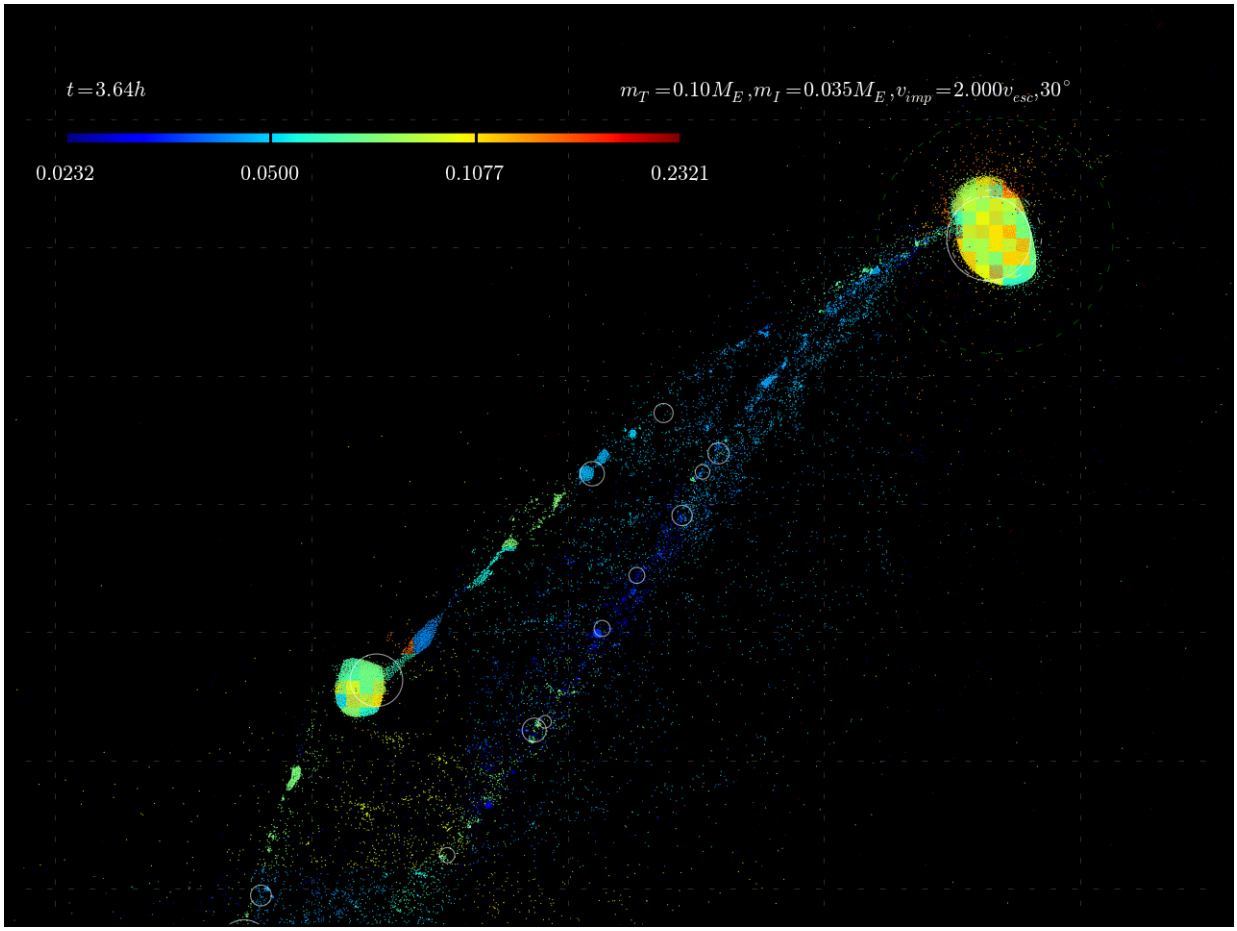

Figure 2.22: Relative computation cost per particle in arbitrary units. Particles inside large clumps require a larger number of interactions during the gravity tree walk, as they are surrounded by a larger number of particles. Note that the cost is only resolved down to individual cost zone cells.

individual processors, but the actual *computational cost*. Computational cost is simply the time spent per particle to calculate for example gravity or the SPH sums and the unit is $[s/\text{particle}]$. If we now distribute the time being spent on computation equally onto processors, we get an efficient and scalable parallelization. Measuring cost for every particle individually would introduce a large overhead and would not be very accurate. For that reason the cost is measured for each cost zone cell. Computations are always performed for all particles, which are in the sub tree of a cost zone level. The elapsed time is measured and the cost is estimated by dividing the time through the number of particles a cost zone has. If particles move from one cost zone to another, the particles' individual cost is removed from one cost zone cell and added up to another cost zone cell.

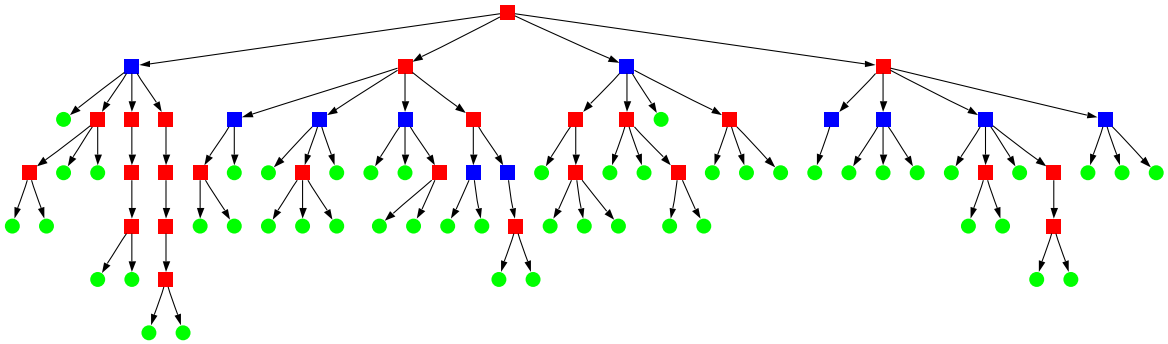

Figure 2.23: Cost zone cells in SPHLATCH v2.

Figure 2.23 shows the same tree structure as in figure 2.19. The difference is now, that cost zone cells are not created down to a fixed level, but are created on demand whenever the total cost of all particles which are in a sub tree of a cell node lies within certain boundaries. Note again that cost is only measured on a cost zone cell level. For a normal cell node which is a child of a cost zone cell, the cost can only be estimated by dividing the total cost of the parent cost zone cell through the number of particles in the given cell. Also note in figure 2.23 that the set of all cost zone cells presents a cut through the tree and includes all particles in the tree. All cost zone cells are then stored in a list.

Cost zone cells are moved to a higher or lower level if their cost goes below or above a certain level. Relative cost is introduced for that reason and is simply a normalized value for cost. The total relative cost of all cost zones in a tree is 1. Now we can introduce the requirement, that cost should be equally distributed amongst a number of cost zone cells, for example 1000 cost zone cells. This means each cost zone cell should have a relative cost of roughly 0.001. Relative cost of cost zone cells vary during the simulation. If now the cost of a cost zone cell rises above a certain threshold, for example 120% of the 0.001 target value, we *refine* the cell by making its cell children cost zone cells and the cell itself a normal cell again. Now the cost zone cell children will each have

a relative cost which is again below the target value. The inverse process is applied, if the common parent cell of several cost zone cells has a cost below a set threshold, for example 80% of the target value. In this case, the cost zone cell children of this cell will be *gathered* and converted to normal cells again. The parent becomes now a new cost zone cell. In the beginning of a simulation usually no cost information is available. In this situation relative cost is simply set to the inverse of the total number of particles for every particle. This is a rough estimate which allows a very first set of cost zone cells.

The actual parallelization of SPHLATCH v2 uses the OpenMP[5]. Compared to MPI, where multiple instances of the code are run as separate system processes, OpenMP runs the code in one single process. For the parallel parts in the code, so called *threads* are started, which can run independently on multiple process core. The threads still run under the control of the process which started them and therefore all threads can access the process' memory space. OpenMP now provides simple compiler directives[6] in order to parallelize certain constructs like for-loops oder vector operations. The user does not have to care on how exactly a loop for example is parallelized, in particular in which order and on how much threads it is executed. The library takes care of that. Variables are normally shared between threads, but can also be marked as private, so that each thread gets its own instance the variable.

Figure 2.24 shows how the derivation procedure is parallelized in SPHLATCH v2. Tree building is not parallelized, as it requires dynamic memory allocation which is not allowed inside an OpenMP thread. As the cost of tree building is only 1-2% of total cost of gravity and SPH sums, executing this part of the code serially is no major drawback. Instead of building the tree every derivation step from scratch, it can also be built by moving the particle nodes in the tree according to their new position. This speeds up tree building a bit, but also introduces new problems like unnecessary tree nodes.

The remaining parts of the derivation parts are now run in parallel. The tree data structure is now only accessed read-only, which allows multiple threads to perform tree walks on it. Work is distributed by running a for-loop over *workers* which run on the elements of the cost zone cell list. This for-loop is parallelized with OpenMP. A worker is a small independent procedure, which can for example run a neighbor search algorithm or calculate gravity for each particle inside a cost zone cell or

[5]http://www.openmp.org/

[6]Directives are not part of the programming language and are only used by the compiler.

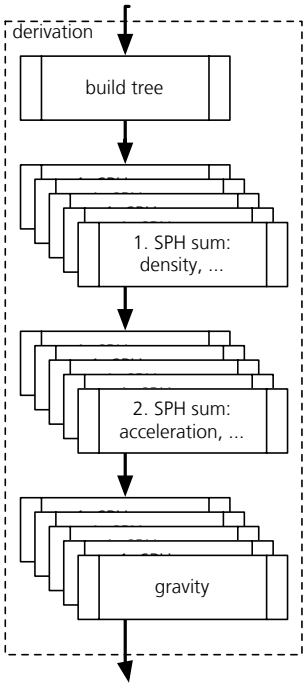

Figure 2.24

2.3.4 Optimizing the code

Although the parallelization of SPHLATCH v2 itself is simple, running it efficiently on multiple processor cores is not easy. The problem is that all workers have to access the nodes of the tree and share its data. The performance of nowadays computers is limited by the *von Neumann bottleneck* (Bryant and O'Hallaron 2010): While even most complex floating point operations like divisions can be completed in only a few clock cycles, accessing the main memory takes between 50-300 cycles (Fry 2008). Modern CPUs only achieve a high performance because of caching the main memory. The basic idea is to have small cache memories which are very fast and store the most recently used pieces of data from the main memory. While of course the data still has to be fetched from memory the first time it is used, chances are very high that it is used again very shortly and can be retrieved from the much faster cache.

Figure 2.25 shows a schematic of the cache architecture of an AMD Opteron 61xx, a modern mutli core CPU. The orange boxes depict the CPU cores. For every memory access, it is checked whether the caches already contain the requested data. In case of the AMD Opteron, memory is divided into pieces of 64 bytes, so called *cache lines*, and always cached as a whole. This means even if only a single byte is fetched from memory, the whole cache line this byte resides in is stored

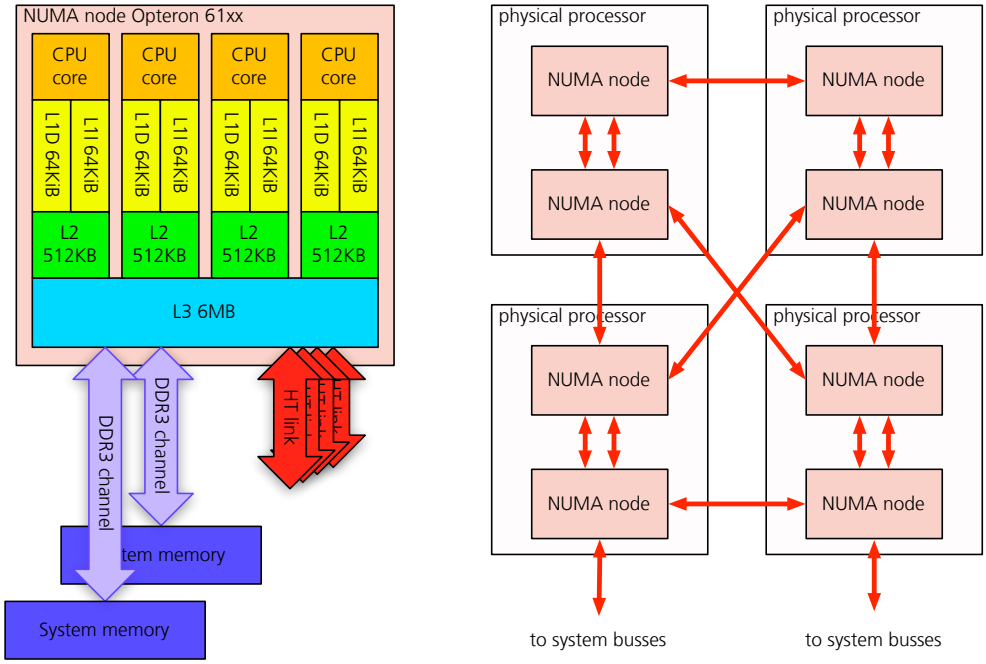

Figure 2.25: Architecture of a nowadays multi-core machine: The example shows a four processor configuration with AMD Opteron 61xx series processors. The left plot shows the layout of a single NUMA (non-unified memory architecture) node. There are four processor cores per node, each with individual L1 and L2 caches. The L1 caches are divided into data and instruction caches, each with a capacity of 64 kilobytes. The slower L2 cache has a size of 512 kilobytes. The L3 cache is shared between all processor cores of the node and has a size of 6 megabytes. It serves also as an for accessing system memory, which is accessible through two busses. Communication with other nodes or the system busses is achieved through 4 *HyperTransport* point-to-point connections. The term NUMA comes from the fact, that system memory is not accessible in an uniform fashion: Memory might be connected locally to the nodes or to a different node in the network. In the latter case, memory access happens indirectly by communicating to a remote node either directly or via multiple other nodes. The right plot shows how those NUMA nodes are interconnected. Two nodes are are located on each physical processor package and interconnected through two HT links. The processors are interconnected via the remaining 4 HT links per package in a topology, which guarantees that the maximum path length between any two nodes is no longer than 3 hops. Especially in case the memory is located far from the local node, access times are considerable.

in the caches.

The cache architecture is hierarchic. Caches are checked for stored data in decreasing level order. If data is found in a cache, a *cache hit* has occured, otherwise a *cache miss*. The sizes of the caches increase with the level, but so do the access times. Data is stored in caches, whenever it has been used by the processor and kept as long as it has not been changed in memory. Note that data in memory can be changes by the other processor cores independently running instructions. Space in the caches is heavily limited. If new data has to be cached, the data which was not accessed for the longest time is expelled and evicted to the next level. So an evicted cache line from the L1 cache is stored in L2 and so on. This means the chances for a cache hit are the highest for more recently accessed data or in other words for data with a strong *temporal locality*. Another way of increasing chances of a cache hit is assuring *spatial locality*, by accessing data in a cache line which already resides in a cache.

Code performance can now be improved considerably by trying to increase the chances of cache hits. In SPHLATCH v2 this is achieved with different techniques. Most importantly the tree node data structures are aligned to the cache lines. A particle node uses exactly one cache line of 64 bytes, as shown in figure 2.26. These guarantees, that only one cache line is used per particle node. In case the data structure would not be aligned, even if it were smaller than a cache line, a particle node could use two adjacent cache lines, which would lead to wasted cache memory (compare figure 2.27 and less nodes to be cached.

Temporal locality in the memory accesses is also assured by the order the derivatives of the particles are calculated in SPHLATCH v2. As described above, workers calculate the derivatives of the particles on a per-cost zone cell basis. If the number of particles in a cost zone cell is relatively small compared to the total number of particles in a simulation, chances are high that those particles are spatially close to each other. For the SPH sums, interactions will take place mostly between the particles of a cost zone cell itself. So chances are high, that neighbors have already been used before for another particles or were actually calculated themselves and reside already in a cache. The same argument applies for the particle to particles interactions in the Barnes & Hut algorithm. If the cost zone cell is small compared to the simulation space, similar cell nodes will be used for the multipole approximations and again chances are high that those cells will already reside in caches.

The rest of the code is equal to SPHLATCH v1 as described above.

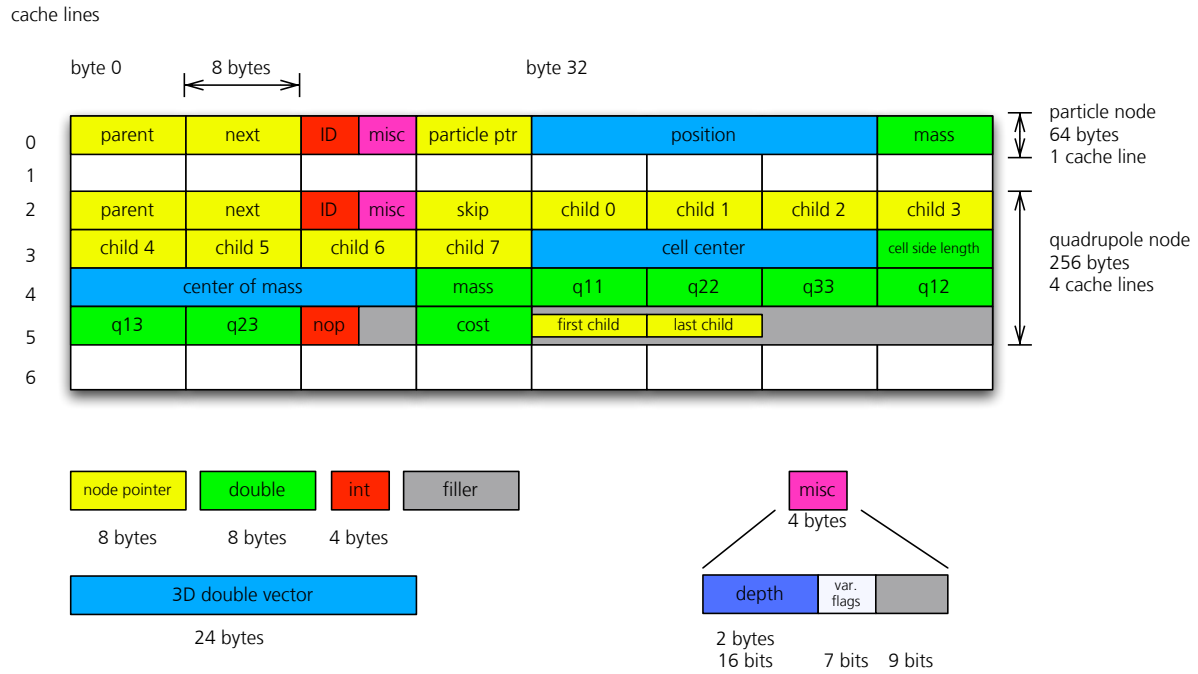

Figure 2.26: Memory layout of the tree nodes in SPHLATCH v2 on the AMD 64 bit architecture with double precision for floating point numbers. The rows indicate cache lines with a width of 64 bytes. The top row shows a particle node, the block below a quadrupole node. Yellow boxes depict node pointers, which are 64 bits or 8 bytes long and can point to any tree node. Green boxes are double precision floating point numbers (8 bytes), red boxes integer number (4 bytes). The purple box depicts a small 4 bytes data structure containing status information about a tree node like depth (short integer, 2 bytes) and various flags, indicating the type of node, whether it is a local or a remote node or in case it is a cell node whether it is a normal or a cost zone cell. Light blue boxes are double precision 3D vectors occupying 24 bytes. Both node data structures share a common first 24 bytes including the pointer to the parent node and the next node, an unique node ID and the node status information. The particle node then also has a pointer to the actual particle, a vector containing the particles position and a floating point number containing its mass. With exactly 64 bytes required for one particle node, it fits into one cache line. The cell node is more complex: Besides the common node data it contains the skip and eight child pointers. Then a floating point vector and number describe the cells position and side-length. Note the data up to here fits into two cache lines. So for tree operations only requiring geometric operations like in case of tree neighbor search, only this data need to be accessed. Each cell node also contains the multipole moments for the Barnes & Hut algorithm: A center of mass 3D vector and a mass floating point number for the monopole moment. The quadrupole moments require six floating point numbers. Additionally, the number of particles and the total cost of all particles in the subtree of the cell node are stored. For cost zone cells, pointers to the first and last child nodes are stored.

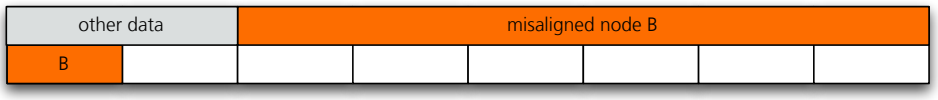

Figure 2.27: Cache line waste due to misaligned data structure: Although the node B data structure is smaller than a cache line, the misalignment causes two cache lines to be used. Because of that, fewer nodes can be stored in a cache.

2.4 Performing collision simulations

The motivation for running collision simulations is to deduce simple quantities like the mass of the largest remnant or deflection angle of the impactor for a given set of impact parameters like impact angle and velocity. A hydrodynamical particle code on the other hand requires initial conditions in the form of an initial distribution of particles with their physical quantities set according to the physical situation to be modeled. The particles should also be in equilibrium initially, in order to start with a well defined condition. The output of the code consists of dumps of particles which need to be post processed. The following few sections will cover techniques required for setting up initial conditions and for post processing particle dumps.

2.4.1 Particles distributions for SPH spheres

For the collisions considered in this thesis, we use three dimensional, spherical impactor and target bodies. The aim is to start with a particle distribution as close to equilibrium as possible. In case of fluids, SPH particles in a relaxed state tend to arrange themselves in a closed-pack configuration. Starting with an HCP lattice and cutting a sphere out of it is a good approximation to such a closed-pack configuration. For a given sphere radius the lattice length yields

$$l = 2r\big(KN\big)^{\frac{1}{3}} \tag{2.80}$$

K denotes the filling factor of the HCP lattice and N the desired number of particles of the sphere. The smoothing length set to $h = 0.85l$ leads to roughly 50 neighbors, except at the surface of the sphere.

2.4.2 Calculating 1D structures

In case of small spheres which are not self-gravitating, the structure of the body is trivial. Pressure has to vanish in equilibrium, so density is simply set to the zero-pressure value ρ_0 for the given specific energy or entropy. All quantities are radius independent.

Self-gravitating spheres have non-trivial structures. For simple equations of state like ideal gas, exact solutions can be calculated analytically in some special cases. In all other cases a solution has to be found numerically. The basic idea is to start with a 1D structure with vanishing pressure and then letting this structure collapse under the influence of self-gravity with a 1D code. A dampening term assures, that the structure does not oscillate forever, but relaxes sufficiently fast:

$$a_{fric} = -\frac{1}{\tau_{fric}} v \qquad \tau_{fric} \gg \tau_{ff} \qquad \tau_{ff} = \frac{1}{\sqrt{G\rho}} \tag{2.81}$$

The dampening time scale τ_{fric} has to be larger than the free-fall time scale τ_{ff} (Shore 2007), so that the oscillations do not get overdamped. We use a 1D Lagrangian code as described in (Benz 1991) and keep the specific entropy constant for all cells. This avoids heating up of the structure during the relaxation process due to artificial viscosity. The 1D Lagrangian code uses the same equations of state as SPHLATCH, like ANEOS or ideal gas. We define the structure to be sufficiently relaxed, of the remnant velocities of the mass elements are below 1% of the escape velocity.

2.4.3 flavoring & relaxing SPH spheres

Flavoring the "vanilla" spheres made out of particles on a HCP lattice with a one dimensional structure is straightforward for standard or miscibile SPH: Dividing the required density ρ_i with the particle density δ_i obtained with SPH sum 2.48 leads to the particle mass m_i. The inverse of the particle density is the particle volume. In case density is an integrated variable, both the particle mass and the density have to be set. The thermodynamic state variables can be set directly, as they are intensive. For convenience reasons, the particles are also numbered with a unique ID, so that each particle can later on be identified uniquely.

In the next step, the 3D SPH sphere also needs to be relaxed to equilibrium. Even if the 1D structure was in equilibrium, the 3D SPH won't be, because all the physical quantities are now interpolated between the particles. This adds in particular edge effects like mentioned before in figure 2.1. We apply the same damping term as mentioned before, but now in 3D:

$$\boldsymbol{a}_{fric} = -\frac{1}{\tau_{fric}}\boldsymbol{v} \qquad \tau_{fric} \gg \tau_{ff} \qquad \tau_{ff} = \frac{1}{\sqrt{G\rho}} \tag{2.82}$$

Note that the distance vector uses the center of mass for the origin. The SPH sphere is now integrated again for a few free fall times, until random particle velocities reduce below 1% of the escape velocity of the body. Velocities of the particles near the surface will be on the order of the escape velocity in a collision simulation. So remnant velocities below $1\% v_{esc}$ will fall in the same order of magnitude as SPH noise. Instead of integrating specific energy, the specific entropy is simply kept constant, in order to avoid artificial heating by the artificial viscosity terms. Due to summation errors in the self-gravity calculation, a small random momentum might be introduce. At the end of the relaxation, the remnant momentum is divided by the bodies' mass and the resulting velocity is subtracted from all particles. Now also the final radius of the body can be determined, by taking the particles at the very surface of the body and adding roughly a smoothing length to this radius. We now have a relaxed, 3D SPH body with the desired structure.

2.4.4 Setting up a collision

Again without gravity, setting up a collision is simple. To all particles of both the impactor and the target, positions and velocities are added, so that the desired initial separation r_0, impact angle θ_{imp} and velocity $\boldsymbol{v}_{imp}$ result:

$$\boldsymbol{r}_0 = \boldsymbol{r}_{0,imp} - \boldsymbol{r}_{0,tar} \tag{2.83}$$

$$\boldsymbol{v}_0 = \boldsymbol{v}_{0,imp} - \boldsymbol{v}_{0,tar} \tag{2.84}$$

$$\theta_{imp} = \arccos \frac{\boldsymbol{r}_0 \boldsymbol{v}_0}{v_0 v_0} \tag{2.85}$$

$$t_0 = -\frac{r_0}{v_0} \tag{2.86}$$

The impact angle θ_{imp} determines the geometry of the impact and is defined as the angle between the relative velocity $\boldsymbol{v}$ and position vector $\boldsymbol{r}$ at the time of impact. Instead of the angle, often the impact parameter $b = \sin \theta_{imp}$ is used.

t_0 denotes the time of the initial conditions particle dump, so that the $t = 0$ yields the time of impact. r_0 is the initial separation of the bodies and has to fulfill $r_0 > R_{imp} + R_{tar}$, so that they don't touch each other initially. R_{imp} and R_{tar} are the radii of the impactor and target body.

In case gravity is no longer negligible, things become slightly more complicated. The motion of the impactor relative to the target no longer follow a straight, un-accelerated trajectory. Instead the impactor with mass M_{imp} and target with M_{targ} present a two-body problem, where both bodies orbit around the mutual bary-center and ultimately hit each other. The relative motion between these two bodies can be replaced by a one-body problem with the reduced mass

$$M_{red} = \frac{M_{imp} M_{targ}}{M_{imp} + M_{targ}} \tag{2.87}$$

Like in the case without gravity, the actual impact is defined as the point of time, where the surfaces of the two bodies first come in contact with each other. Neglecting tidal deformation of the bodies, this is the case when $r = r_{imp} = R_{imp} + R_{tar}$. Figure 2.28 shows the orbit of the impactor in frame of reference where the target is at rest. The impact angle is still defined the same way as without gravity, but we newly introduce the heading angle β which is defined as the deviation of the heading relative to a motion perpendicular to the centre of mass of the larger mass:

$$\theta_{imp} = \frac{\pi}{2} - \beta_{imp} = \arccos \frac{\boldsymbol{r}_{imp} \boldsymbol{v}_{imp}}{v_{imp} v_{imp}} \tag{2.88}$$

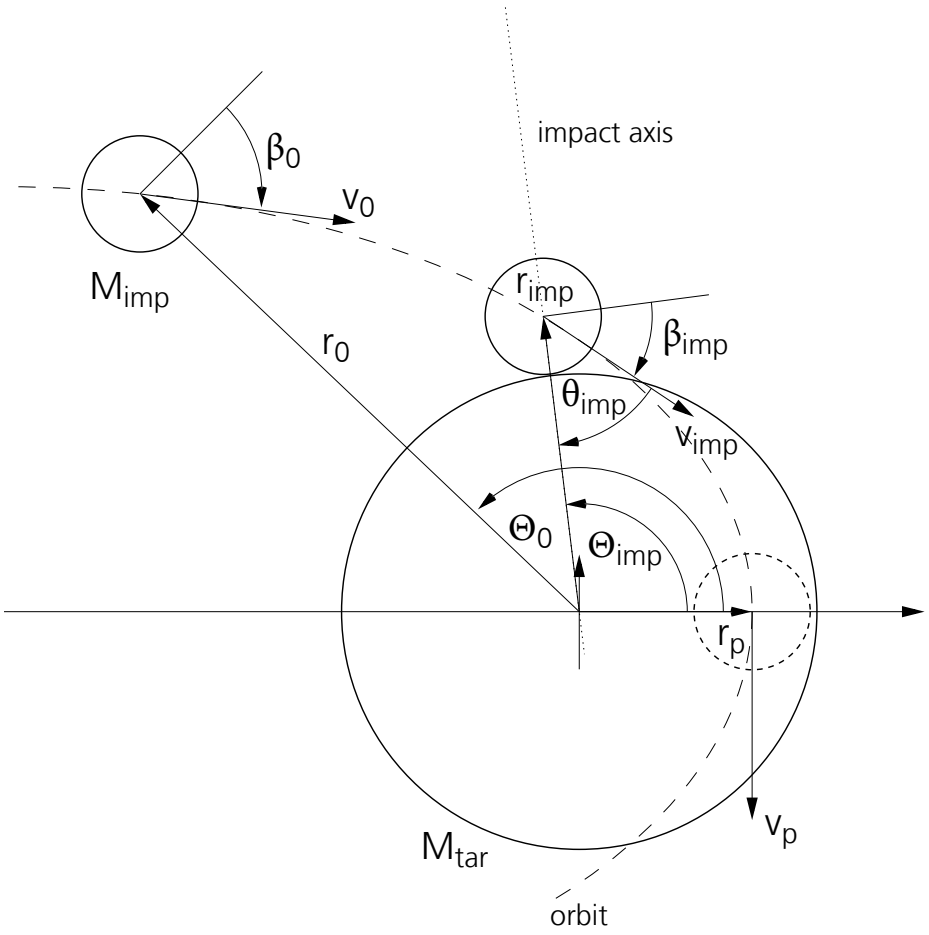

Figure 2.28: Sidecut view of the impact plane. In a target centric view, the impactor follows an orbit around a body with the reduced mass.

At perihel, the heading angle is zero. Energy and angular momentum conservation let us determine the eccentricity of the orbit (Thomson 1986):

$$e = \left(\frac{r_{imp}\boldsymbol{v}_{imp}^2}{\mu} - 1\right)^2 \cos\beta_{imp}{}^2 + \sin\beta_{imp}{}^2 \qquad \mu = G(M_{imp} + M_{targ}) \tag{2.89}$$

which stays constant. It is $e = 1$ for a parabolic orbit, when $v_\infty = 0$ or $e > 1$ for a hyperbolic orbit where $v_\infty > 0$. The true anomaly of the impact can now be determined as

$$\Theta_{imp} = \arctan\frac{k_1 \sin\beta_{imp}\cos\beta_{imp}}{k_1\cos\beta_{imp}{}^2 - 1} \qquad k_1 = \left(\frac{r_{imp}\boldsymbol{v}_{imp}^2}{\mu} - 1\right)^2 \tag{2.90}$$

Note that for the arctangent in this formula, a special numerical function has to be used in order to get the correct true anomaly. The function is called *atan2* in most programming languages and returns the correct angle in each quadrant of the coordinate system. The distance at perihel is then

$$r_p = r_{imp}\frac{1 + e\cos\Theta_{imp}}{1 + e} \tag{2.91}$$

The initial true anomaly then follows from the initial separation r_0 between the two bodies which is also a free parameter

$$\Theta_0 = \arccos\frac{r_p(1+e) - r_0}{er_0} \tag{2.92}$$

The initial relative velocity follows from energy conservation

$$v_0 = \sqrt{v_\infty^2 + \frac{2\mu}{r_0}} \tag{2.93}$$

Now the orbital state vectors can be determined. We choose a reference frame, where total linear momentum vanishes and the overall center of mass is at the origin. For convenience reasons, the spatial orientation is chosen as such that the impact axis coincides with the y-axis of the coordinate

system. The relative position and velocity vectors therefore yield

$$\boldsymbol{r}_0 = r_0 \begin{pmatrix} -\cos\alpha \\ \sin\alpha \end{pmatrix} \qquad \alpha = \frac{\pi}{2} - \Theta_{imp} + \Theta_0 \tag{2.94}$$

$$\boldsymbol{v}_0 = v_0 \begin{pmatrix} -\cos\gamma \\ \sin\gamma \end{pmatrix} \qquad \gamma = -\beta_0 - \Theta_{imp} + \Theta_0 \tag{2.95}$$

$$\boldsymbol{r}_{0,tar} = -\boldsymbol{r}_0 \frac{M_{imp}}{M_{tar} + M_{imp}} \qquad \boldsymbol{v}_{0,tar} = -\boldsymbol{v}_0 \frac{M_{imp}}{M_{tar} + M_{imp}} \tag{2.96}$$

$$\boldsymbol{r}_{0,imp} = \boldsymbol{r}_0 \frac{M_{tar}}{M_{tar} + M_{imp}} \qquad \boldsymbol{v}_{0,imp} = \boldsymbol{v}_0 \frac{M_{tar}}{M_{tar} + M_{imp}} \tag{2.97}$$

$$\tag{2.98}$$

The initial time can be calculated via the mean anomaly Ψ, which links true anomaly with time. Together with the semi-major axis a, time is given by

$$\sqrt{\frac{\mu}{a^3}} t(\Theta) = \frac{e\sqrt{e^2-1}\sin\theta}{1 + e\cos\theta} - \log \frac{\sqrt{e^2-1} + (e-1)\tan\frac{\Theta}{2}}{\sqrt{e^2-1} - (e-1)\tan\frac{\Theta}{2}} \qquad \text{for } e > 1 \tag{2.99}$$

$$\sqrt{\frac{\mu}{a^3}} t(\Theta) = \frac{1}{2}\Big(\tan\frac{\Theta}{2} + \frac{1}{3}\tan^3\frac{\Theta}{2} \Big) \qquad \text{for } e = 1 \tag{2.100}$$

$$a = \frac{r_p}{e-1} \tag{2.101}$$

so that the initial time yields

$$t_0 = t(\Theta_{imp}) - t(\Theta_0) \tag{2.102}$$

For perfectly head-on collisions ($\theta_{imp} = 0$) this calculation breaks down. In this case, the second components of both the relative position and velocity vectors are then simply set to the scalar values r_0 and v_0. The initial time is given by the free fall time from the initial separation to impact:

$$t_0 = -\sqrt{\frac{r_0^3}{2\mu}} \Big(\sqrt{r'(1-r')} + \arccos r' \Big) \qquad r' = \frac{r_{imp}}{r_0} \tag{2.103}$$

2.4.5 Gravitationally bound clump search

In collisions where self-gravity is non-negligible, an important question is which particles re-accrete and form gravitationally bounded clumps. Re-accretion often happens on longer timescales than are possible to simulate. Therefore a method is required to estimate which particles re-accrete

again. A good estimate is to look at the total energy of the particles:

$$\frac{E_{tot,i}}{m_i} = \frac{1}{2}\boldsymbol{v}_i^2 + \Phi \tag{2.104}$$

The problem is that both terms depend on the choice of reference. A method which works well is to start with a particle and look what other particles are bound to this initial particle. Those particle are then taken as the members of the new clump and again it is checked whether other particles are bound relative to this clump. This procedure is repeated until no more particles can be added. Figure 2.29 shows the algorithm in detail.

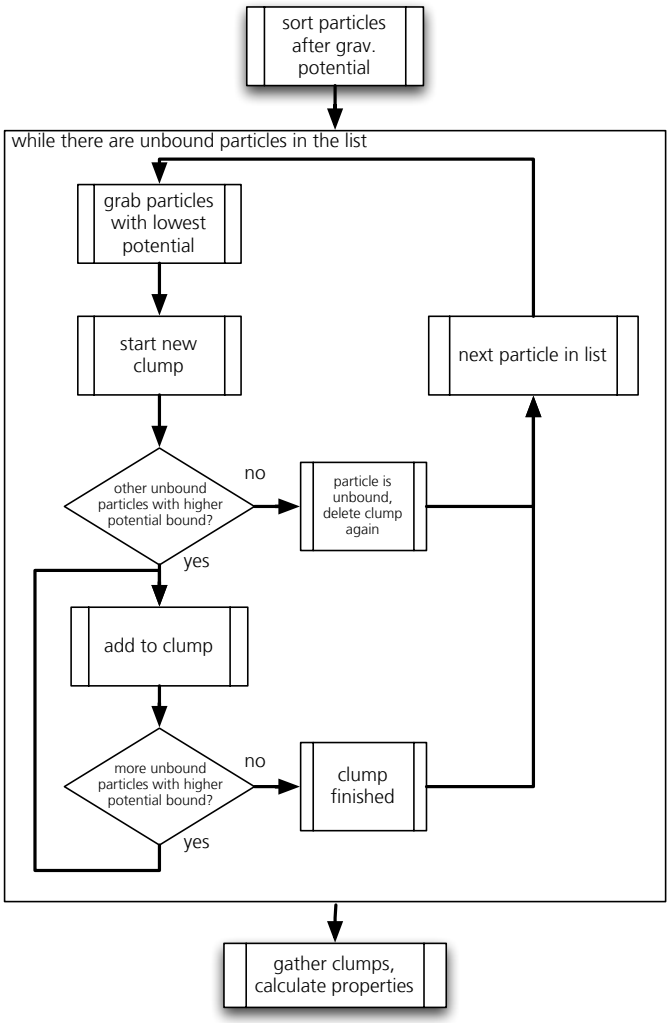

Figure 2.29: Algorithm to find gravitationally bound clumps.

A list is made of all particles, which is sorted after the gravitational potential. The algorithm is then started for the particle lowest in the potential. It is checked whether there are bound particles until no more particles are added to the current clump. If there are no bounded particles at all for a particle, it is marked as unbound and the next particle in the sorted list is checked. In situations where most particles form a few well bound clumps, the algorithm has roughly a computational complexity $O(N)$. The bounded clumps are then each found after a few iterations. The worst case is, when the particles are only marginally bound and in each iteration only very few new bound particles are add to the clump. In that case the complexity of the algorithm approaches $O(N^2)$.

This clump search algorithm is normally run directly during code execution as an auxiliary task whenever a particle dump is stored (see figure 2.18). As this algorithm is run serially and in case of slow convergence would lead to a considerable slow down of a simulation, it is only executed if the system appears to be gravitationally bound enough and fast convergence can be expected. This is estimated by comparing total kinetic E_{kin} and potential energy E_{pot} of all particles. The clump search is only performed if the virial ratio k_{virial} [7] is smaller than a certain value:

$$E_{kin} = \sum_i \frac{1}{2} m_i \boldsymbol{v}_i^2 \qquad E_{pot} = \sum_i \frac{1}{2} m_i \phi_i \qquad k_{virial} = -\frac{E_{kin}}{2E_{pot}} \tag{2.105}$$

A value of $k_{virial} < 3$ assures in practice, that the clump search algorithm converges reasonably fast. Otherwise the algorithm is executed serially after the simulation during post-processing. After the clumps have been found, several properties of this gravitationally bound objects can be determined, like mass, center of mass, linear and angular momentum, mean density, composition and so on.

2.4.6 Friend-of-friend (FOF) clump search

Another type of clump search is based on a friend of friend algorithm. The goal is to find clumps of particles with a certain minimal density ρ_{FOF} adjacent to other particles with a least that minimal density. Adjacent means, that they are no further apart than their mean smoothing length h_{ij} times a cohesion factor k and are *friends*. In case $k = 2$, friends corresponds to the definition of SPH neighbor. Figure 2.30 shows the algorithm:

For all particles it is checked, whether they have a least the minimal density ρ_{FOF}. If not, they are marked as a non-friend. Otherwise it is checked, whether they already have a clump ID set and are therefore part of a clump. If not, a neighbor search is started within a radius kh_i in order to find the friends. If some friends already have a set clump ID, the current particle is

[7] the virial ratio of a system fulfilling the virial theorem is 1

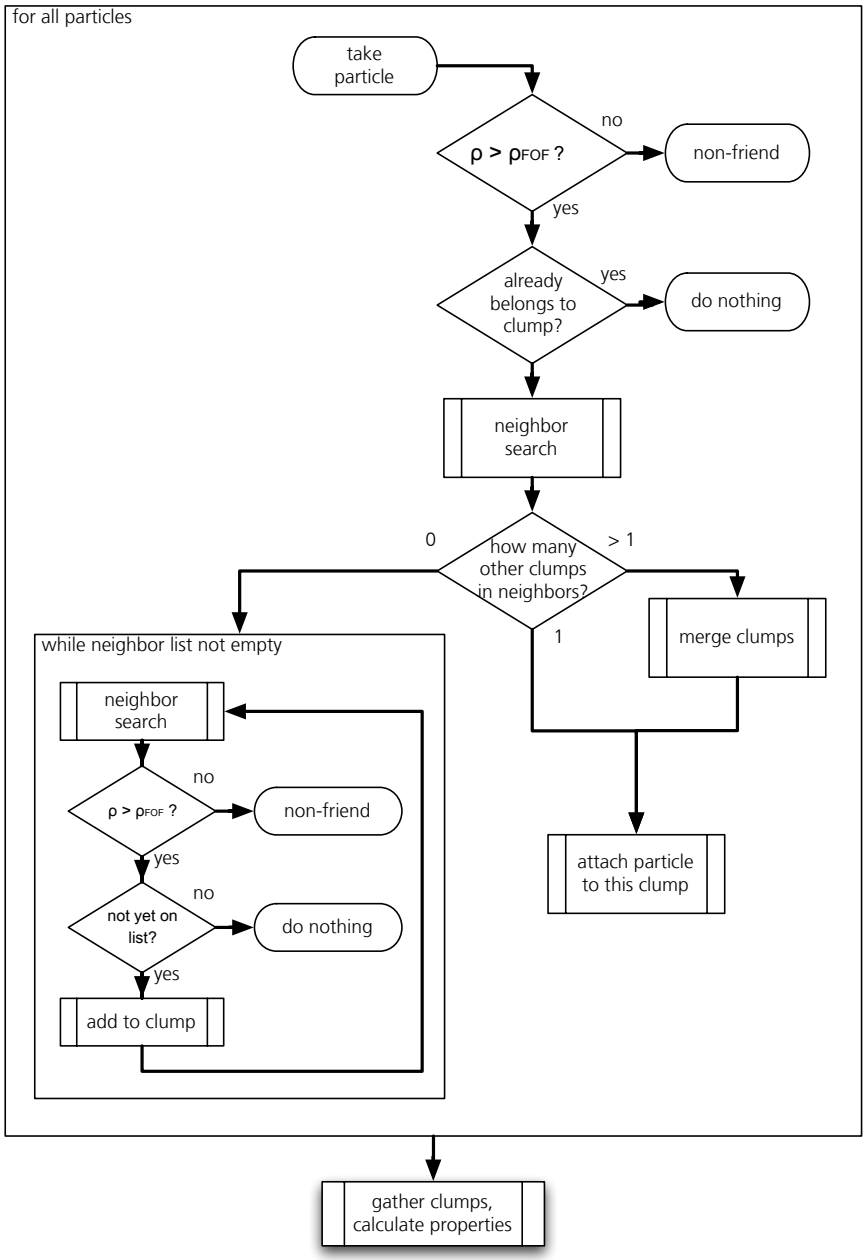

Figure 2.30: Algorithm to find clumps of particles, which are interconnected as friends of friends.

simply attached to this clump. In case those friends have different IDs, the corresponding clumps are additionally merged. In case friends are found, but none of them have a set clump ID, a new clump is started with this particles by putting them on a list of friend neighbors. Each of the found friend neighbors is marked as a friend and also searched for friends. It is important only to put particles onto the list, which have not yet been marked as friends of the current clump. Otherwise pairs of friends could find each other again as new neighbors infinitely and the algorithm would never terminate. At the end of the algorithm the properties of the FOF clumps can be determined as in the previous clump searching algorithm.

The algorithm has two free parameters: the minimal clump density ρ_{FOF} and the cohesion factor k. In case fluid or solid bodies have to be found ρ_{FOF} should be set slightly below the reference density ρ_0 of the least dense material in the simulation. With this choice solid and liquid material counts as clump interconnecting material, but atmospheres of material in vapor phase do not.

This FOF algorithm is useful for example to find satellites of bodies. While the previous algorithm will treat a parent body and its satellite as one bound object, the FOF successfully finds two individual bodies.

2.4.7 Determining disks

Determining the disk of a body is important for simulations of satellite formation. Disk detection involves several steps: First of all the gravitationally bound clump has to be found for which the disk particles should be determined. This is done with algorithm previously described. The basic idea is now to determine the surface of the central body and then find the particles which have orbits with a perihel above that surface. To find the surface, a method which works well in practice is to sort all particles according to the gravitational potential. This sorted list is then traversed toward higher potential. Particles with a density below a certain value ρ_{surf} are then considered surface particles. This density ρ_{surf} should be slightly below the zero-pressure density ρ_0 for which pressure vanished for a given thermodynamic state. In standard SPH, the particles of a body which is surrounded by vacuum will have a density slightly below this zero-pressure density, because of edge effects demonstrated before (see figure 2.1). After a fraction of $\approx$ 1% of the total particles in the gravitationally bound clump have been found as surface particles, the list traversal is stopped. Taking this fraction guarantees, that in case the central body is surrounded by an atmosphere, not too many low-density atmosphere particles are considered to belong to the surface layer. Note that it is still not known at this point, which particles belong to the central body. For that reason, the center of mass of all surface particles is now taken as a first guess for the center of mass of the central body. The mean distance of the surface particles to this first guess of the central body's

center of mass is determined and all particles within this radius are now defined as being part of the central body. The center of mass and the velocity of the central body can now be determined. The radius of the central body is now R_{bod}. All other particles are now assumed to be on an orbit around the central body. Determining the orbit is basically the inverse problem what we have solved for setting up a collision in the gravity regime. For each particle we have the orbital state vectors $\boldsymbol{r}_0$ and $\boldsymbol{v}_0$ relative to the central body and would like to determine the orbit parameters, notably the radius of perihel r_p and the eccentricity in order to know whether the particle is bound. In a first step the heading angle β_0 and the eccentricity vector $\boldsymbol{e}$ determined:

$$\beta_0 = \arcsin \frac{\boldsymbol{r}_0 \boldsymbol{v}_0}{r_0 v_0} \tag{2.106}$$

$$\boldsymbol{e} = \frac{\boldsymbol{v}_0 \times (\boldsymbol{r}_0 \times \boldsymbol{v}_0)}{\mu} - \frac{\boldsymbol{r}_0}{r_0} \qquad \mu = G(M_{bod} + m_i) \tag{2.107}$$

M_{bod} is the mass of the central body and m_i the particle's mass. From the heading angle and the eccentricity vector follow directly the true anomaly Θ_0, the eccentricity and from that the radius of perihel:

$$\Theta_0 = \arctan \frac{k_1 \sin\beta_0 \cos\beta_0}{k_1 \cos\beta_0 \cos\beta_0 - 1} \qquad k_1 = \frac{r_0 v_0^2}{\mu} \tag{2.108}$$

$$e = \|\boldsymbol{e}\| \tag{2.109}$$

$$r_p = r_0 \frac{1 + e\ \cos\Theta_0}{1 + e} \tag{2.110}$$

Now the particles not belonging to the central body can be categorized into three different types. Figure 2.31 shows a snapshot of a collision simulation, with particles color coded according to the categorization from the disk determination algorithm. Now if the eccentricity $e \geq 1$ and the true anomaly $\Theta_0 > 0$, the particle will escape the gravitational clump (red particles). Not that not too many such particles are usually detected, as they should not have been detected as part of the gravitationally bound clump in the first place (grey particles are not part of the clump). If $\Theta_0 < 0$, it has to be checked, whether $r_p < R_{bod}$, in which case the particle is probably going to re-impact. Particles with $e < 1$ have bound orbits. If the perihel now comes to lies inside the central body so that $r_p < R_{bod}$, the particle is classified as re-impacting (orange), otherwise it is classified as a disk particles (green).

Note that this algorithm assumes, that each particle simply behaves as a single particle orbiting the center of mass of the central body. Pressure forces are completely neglected, as are interactions

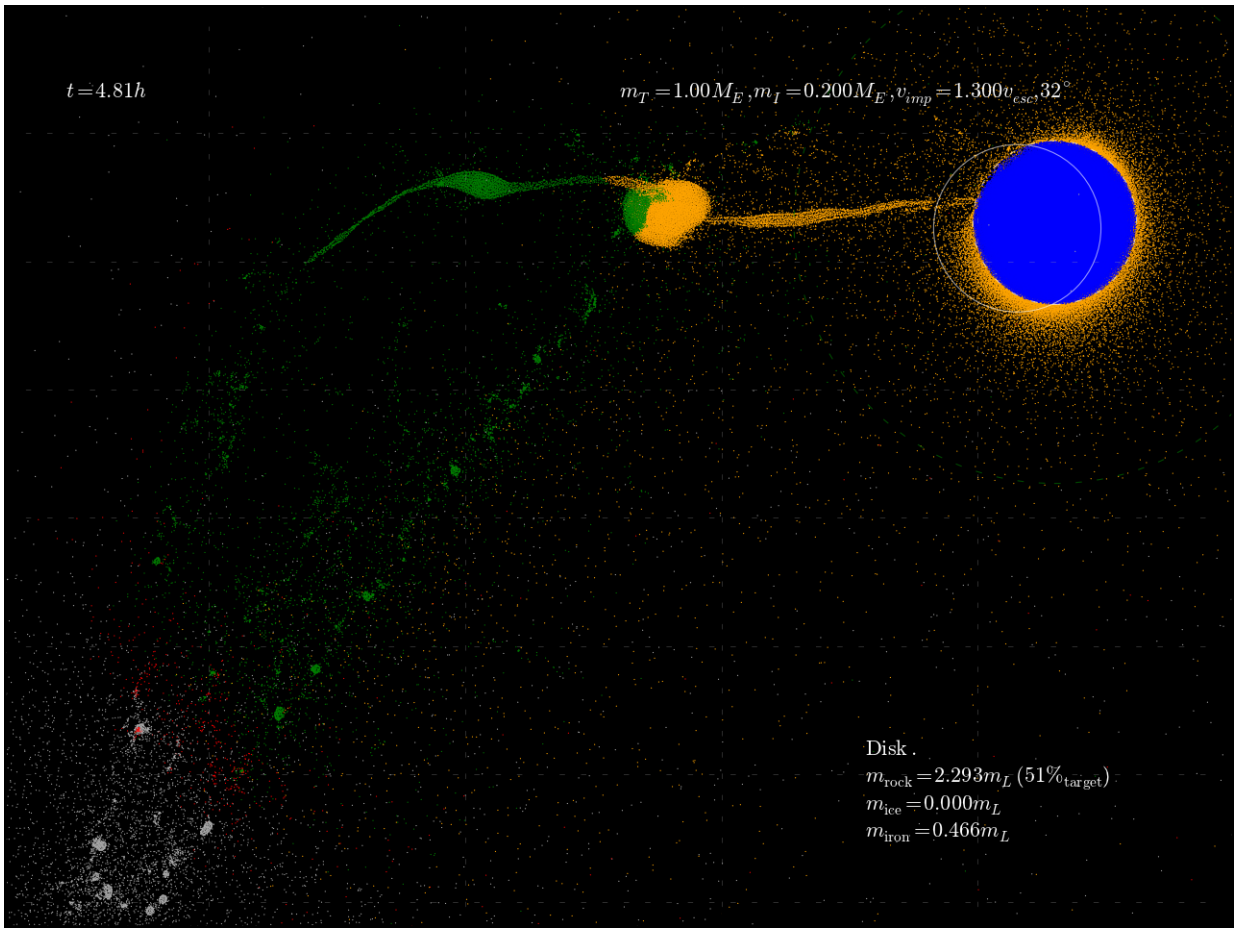

Figure 2.31: A snapshot of a potentially Moon-forming collision. Particles are color coded according to their classification from the disk determination algorithm: Blue indicates particles of the central body. Green particles have orbits with a perihel above the central body's orbit, neglecting hydrodynamical forces. Orange particles will re-impact again the central body. Red particles are on escaping orbits and grey particles are not part of the gravitationally bound clump. Note that only very few particles are marked as escaping, as most of them are not found as members of the gravitationally bound clump in the first place.

with other disk particles or particles from other clumps. Nevertheless this method determines disks surprisingly well, if the whole system has evolved sufficiently enough.

Bibliography

Oscar Agertz, Ben Moore, Joachim Gerhard Stadel, Doug Potter, Francesco Miniati, Justin Read, Lucio Mayer, Artur Gawryszczak, Andrey Kravtsov, J. J Monaghan, Ake Nordlund, Frazer Pearce, Vincent Quilis, Douglas Rudd, Volker Springel, James Stone, Elizabeth Tasker, Romain Teyssier, James Wadsley, and Rolf Walder. Fundamental differences between SPH and grid methods. *arXiv*, astro-ph, Oct 2006.

D. S. Balsara. von Neumann stability analysis of Smooth Particle Hydrodynamics -- Suggestions for optimal algorithms. *Journal of Computational Physics*, 121:357--372, 1995.

Josh Barnes and Piet Hut. A Hierarchical O(N log N) Force-Calculation Algorithm. *Nature*, 324: 446, Dec 1986. doi: 10.1038/324446a0.

W Benz. An introduction to computational methods in hydrodynamics. *Late Stages of Stellar Evolution. Computational Methods in Astrophysical Hydrodynamics. Proceedings of the Astrophysics School II*, 373:258, Jan 1991. doi: 10.1007/BFb0032277. ISBN: 0387536205.

W. Benz and E. Asphaug. Impact simulations with fracture. I - Method and tests. *Icarus*, 107: 98--+, January 1994. doi: 10.1006/icar.1994.1009.

R. Bryant and D.R. O'Hallaron. *Computer systems: A Programmer's Perspective*. Pearson Education Canada, 2010. ISBN 9780136108047.

Greg Fry. Implementing AMD cache-optimal coding techniques. *Whitepaper*, Sep 2008.

D.E. Grady and M.E. Kipp. Continuum modelling of explosive fracture in oil shale. *International Journal of Rock Mechanics and Mining Sciences Geomechanics Abstracts*, 17(3):147 -- 157, 1980. ISSN 0148-9062. doi: DOI:10.1016/0148-9062(80)91361-3.

Eiichiro Kokubo, Shigeru Ida, and Junichiro Makino. Evolution of a circumterrestrial disk and formation of a single moon. *Icarus*, 148:419, Dec 2000. doi: 10.1006/icar.2000.6496. (c) 2000: Academic Press.

Stephen L. W McMillan and Sverre J Aarseth. An O(N log N) integration scheme for collisional stellar systems. *ApJ*, 414:200, Sep 1993. doi: 10.1086/173068.

H. J Melosh. A hydrocode equation of state for SiO_2. *Meteoritics & Planetary Science*, 42:2079, Jan 2007.

J. J. Monaghan. Smoothed particle hydrodynamics. ARA&A, 30:543--574, 1992. doi: 10.1146/annurev.aa.30.090192.002551.

J. J Monaghan. Smoothed particle hydrodynamics. *In: Annual review of astronomy and astrophysics. Vol. 30 (A93-25826 09-90)*, 30:543, Jan 1992. doi: 10.1146/annurev.aa.30.090192.002551.

J. J Monaghan. SPH and Riemann Solvers. *Journal of Computational Physics*, 136:298, Sep 1997. doi: 10.1006/jcph.1997.5732.

G.E Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114--117, Apr 1965.

J. P. Morris and J. J. Monaghan. A switch to reduce SPH viscosity. *J. Comput. Phys.*, 136(1):41--50, 1997. ISSN 0021-9991.

Andrew F Nelson, Willy Benz, and Tamara V Ruzmaikina. Dynamics of circumstellar disks. ii. heating and cooling. *ApJ*, 529:357, Jan 2000. doi: 10.1086/308238.

Bruno Nyffeler. Modelling of impacts in the solar system on a beowulf cluster. *Dissertation*, page 145, Mar 2006.

Frank Ott and Erik Schnetter. A modified SPH approach for fluids with large density differences. *arXiv*, physics.comp-ph, Jan 2003.

W. H. Press. *Numerical recipes in C++ : The Art of Scientific Computing*. ISBN : 0521750334, 2002.

Daniel J Price. Magnetic fields in astrophysics. *Dissertation*, Oct 2004.

S.N. Shore. *Astrophysical hydrodynamics: An Introduction*. Physics textbook. Wiley-VCH, 2007. ISBN 9783527406692.

B Solenthaler and R Pajarola. Density contrast SPH interfaces. *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Jul 2008.

V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce. Simulations of the formation, evolution and clustering of galaxies and quasars. Nature, 435:629--636, June 2005. doi: 10.1038/nature03597.

Volker Springel. The cosmological simulation code GADGET-2. *MNRAS*, 364:1105, Dec 2005. doi: 10.1111/j.1365-2966.2005.09655.x.

Joachim Gerhard Stadel. Cosmological N--body Simulations and their Analysis. *Dissertation*, Jul 2001.

S Thompson. ANEOS Analytic Equations of State for Shock Physics Codes Input Manual. *Sandia Report*, 1990.

W.T. Thomson. *Introduction to Space Dynamics*. Dover books on engineering. Dover, 1986. ISBN 9780486651132.