GPU accelerated fluid simulation for planet formation

Hendrik Schwanekamp and Emma Kraft

University of Koblenz, Germany
Max-Planck Institute for Astronomy, Heidelberg, Germany
hendrikschwanekamp@gmail.com
Year: 2020

Abstract. Smoothed Particle Hydrodynamics (SPH) is commonly used for fluid simulation like gases and liquids, at it's core we find a method for solving partial differential equations that can be extended to simulate solids, deformable objects and granular materials. We apply SPH to planet formation, by using it to simulate interaction in a collapsing pebble cloud. The influence of different pebble clouds on the shape and size of the resulting structures can be studied.
Such simulations can take days, running on an HPC system, and require additional processing and analysis thereafter. We aim to shorten the time from the experiment idea to the finished simulation by using GPUs. The flexible CUDA implementation allows for test runs at interactive speeds even on a laptop computer, as well as slower high accuracy simulations on workstations and HPC systems, that feature more powerful GPUs. We provide a build in visualization tool to speed up the analysis process.

Keywords: planet formation · numerical simulation · GPGPU · SPH

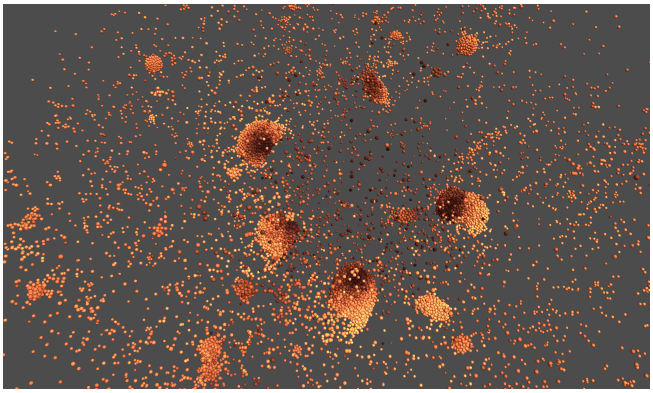

Fig. 1: Simulation of a collapsing pebble cloud with initial rotation. Rendered using our build in visualization. Brightness indicates movement speed.

1 Introduction

Computer simulation play an important role in the astrophysical studies of structure formation. Those processes can take a long time and are hard to observe. Planets and asteroids pose an even bigger challenge in observation, as they are very small and – in contrast to stars – do not emit light on there own. The details of their formation are still actively researched. Computer simulations are for example used to test new theories and to better understand observations.

Planets form in gaseous rotating disks around young stars, as shown in figure 2. Those disks also contain dust particles. Due to the complex interaction between dust and gas, regions of high dust concentration are formed. Dust grains stick together, forming pebbles of a millimeter to ten centimeters in size. Gravitational forces act between the particles, pulling them together. At the same time particles at different distances from the central star experience different forces while rotating around it (tidal forces). If the gravitational forces – pulling the dust together – are stronger then those tidal forces, the cloud collapses to form a planet or asteroid. The density, at which a clouds internal gravitational forces are stronger that the tidal forces is called the hill density.

Sometimes this process produces objects of unexpected shape and structure (see the asteroid Ultima Thule in figure 3). With our simulation we aim to study the influence of initial density (just below hill density - just above hill density - well above hill density), distance from the central star and particle size on the shape of the forming object.

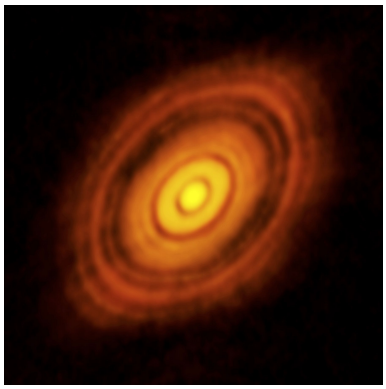

Fig. 2: Protoplanetary disk around HL Tauri. [38]

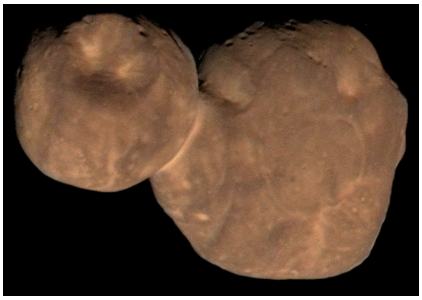

Fig. 3: Ultima Thule, asteroid made of two 'pancake'-shaped ellipsoids. Photograph 1. January 2019 by the "New Horizons" spacecraft [40].

We simulate the interaction between dust particles using Smoothed Particle Hydrodynamics (*SPH*). SPH was developed by Lucy [22] and independently Gingold and Monaghan [12] in 1977, for simulations in astrophysics. Since then it was used for various fluid simulations in different fields. Springel et al. build

the SPH-code GADET2 to simulate the evolution of galaxies [37]. Bate et al. simulated star formation (eg. [4, 5]) and protostellar discs [3]. Laibe et al. used SPH to simulate the growth of dust grains in protoplanetary discs [19]. Libersky and Petschek firstly applied SPH to continuum mechanics and simulated solid bodies [20]. With further improvements in [21, 34]. It introduced to the astrophysics community by Benz and Asphaug [7]. Schaefer et al. simulate collisions between asteroids by adding a damage model [36]. SPH is also used in computer graphics [10, 17, 30]. Mostly to simulate liquids and granular materials in movie productions but also for games and interactive software. To archive the speed needed to perform an interactive simulation, SPH can be implemented to use the graphics processing unit *GPU* [14]. The physical simulation code by Schaefer et al. also uses GPUs, as they have become a common addition to HPC systems in the past years. In this paper we can only give a quick introduction into SPH. For more complete information the reader is referred to the reviews by Monaghan [28, 29] and the more recent Eurographics tutorial by Koschier et al. [18].

Our implementation is mostly written in CUDA, one way to write code for GPUs. In addition, we provide a build-in visualization module and options to balance speed against accuracy. Test runs can be performed interactively on consumer hardware, allowing for quick experimentation. The same equations and code can then be used to run a high accuracy simulations on an HPC system. This allows for *rapid prototyping* of simulations. The code is open source and available on github [1].

In this paper, we mainly present our model as well as our implementation. Only preliminary results are discussed. Section 2 starts with explaining the physical model used in our simulation. We continue with an introduction into SPH and explain how we use it to discretize our physical model in section 3. In section 4 we introduce the CUDA programming model and detail our implementation. Finally we present some first results in section 5 before drawing a conclusions in section 6.

2 Physical model

2.1 Gravity

Particles of matter are attracted towards each other by gravity. When mostly bigger chunks of matter and longer distances are considered, gravity dominates the other fundamental forces. Therefore it is an important factor in structure formation. As an approximation, objects can be treated as point masses when calculating gravitational forces [23].

The gravitational force resulting from the interaction of two objects i and j is given by

$$\boldsymbol{f_{ij}} = G\frac{m_i m_j}{||\boldsymbol{r_{ji}}||^2}\frac{\boldsymbol{r_{ji}}}{||\boldsymbol{r_{ji}}||}. \tag{1}$$

[1] Code on github: `https://github.com/hschwane/GraSPH2`.

Here m denotes the masses of both bodies and $\boldsymbol{r_{ji}} = \boldsymbol{r_j} - \boldsymbol{r_i}$ the vector from one particle to the other. G is the gravitational constant. The first fraction gives the strength of the force, while the second is a unit vector specifying its direction. In a system with multiple bodies, the total gravitational force acting on a body is calculated by taking the sum of the interaction with every other body:

$$\boldsymbol{F_i} = Gm_i \sum_{j=1;i\neq j}^{N} \frac{m_j \boldsymbol{r_{ji}}}{||\boldsymbol{r_{ji}}||^3} \tag{2}$$

As two bodies approach each other $\boldsymbol{r_{ji}}$ approaches zero and the gravitational force grows without bounds. To prevent instabilities in the simulation, it is common practice to introduce a smoothing factor $\epsilon > 0$. Since we are mostly interested in acceleration we use Newtons second law to rewrite equation 2 as:

$$\boldsymbol{a}_i^{grav} \approx G \sum_{j=1}^{N} \frac{m_j \boldsymbol{r_{ji}}}{\left(||\boldsymbol{r_{ji}}||^2 + \epsilon^2\right)^{3/2}} \tag{3}$$

Note that the introduction of ϵ also eliminates the need for the side condition $i \neq j$ [33].

2.2 Dust interaction

Dust interaction is modeled using partial differential equations from continuum mechanics. We apply a plasticity model commonly used for granular material. No additional sticking force is applied. Structures resulting from our simulation will therefore be solely gravitational bound (also known as *rubble piles*).

Governing equations The governing equations from continuum mechanics define the behavior of a body with respect to time.

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \boldsymbol{v} \tag{4}$$

$$\frac{d\boldsymbol{v}}{dt} = \frac{1}{\rho} \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{a}_{\text{other}} \tag{5}$$

Here ρ denotes density and $\boldsymbol{v}$ velocity. Density only changes as matter moves with velocity. The first equation therefore ensures conservation of mass (no mass should be created or destroyed). The second equation – the equation of momentum conservation – relates the change of velocity to a spatial derivative of the stress tensor σ. The stress tensor defines forces acting from one part of the body onto another (see next section). A part of the body accelerates when these forces vary in its vicinity. $\boldsymbol{a}_{\text{other}}$ includes all external accelerations like those resulting from gravity [36].

Stress Stress describes forces inside a body resulting from its deformation. The Cauchy stress tensor $\boldsymbol{\sigma}$ is one possible definition of stress. In three dimensions it is a 3×3 matrix defined at every point of the material. To understand the meaning of $\boldsymbol{\sigma}$, imagine a plane with normal $\boldsymbol{n}$ dividing the body. Consider a small area Δa on this plane. $\Delta \boldsymbol{p}$ is the force one part of the body exerts on the second part over this area. Then:

$$\boldsymbol{\sigma n} = \lim_{\Delta a \to 0} \frac{\Delta \boldsymbol{p}}{\Delta a} \tag{6}$$

I.e. the stress tensor relates the normal of a plane dividing a body, to the force per area exerted from one part of said body to the other (figure 4) [8].

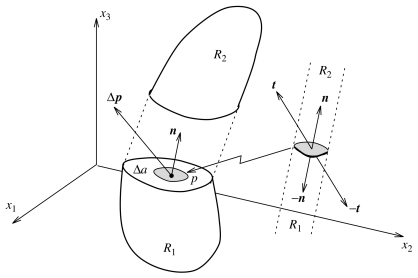

Fig. 4: Shows the relationship between values used in the above explanation of the Cauchy stress $\boldsymbol{\sigma}$. Here $\boldsymbol{t} = \boldsymbol{\sigma n}$ from equation 6 [8].

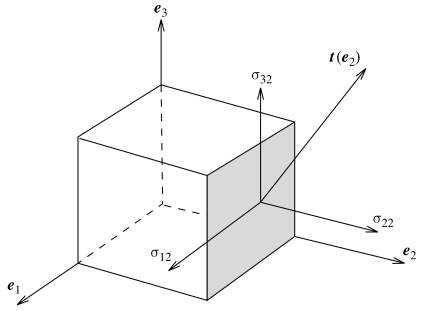

Fig. 5: Direction different components of $\boldsymbol{\sigma}$ act in. [8].

Values on the diagonal of $\boldsymbol{\sigma}$ describe stress acting in the direction of the normal $\boldsymbol{n}$ (compression / extension), while other values describe stress acting parallel to the surface (shear) (figure 5). The Cauchy stress tensor can be rewritten as the sum of the deviatoric stress tensor $\boldsymbol{S}$ and pressure p (using identity matrix $\boldsymbol{I}$) [36]:

$$\boldsymbol{\sigma} = -p\boldsymbol{I} + \boldsymbol{S} \tag{7}$$

When dealing with a liquid or gas, the deviatoric stress tensor $\boldsymbol{S}$ is zero and equation 5 becomes the Euler equation, as used in the simulation of compressible, inviscid fluids.

$$\frac{d\boldsymbol{v}}{dt} = -\frac{1}{\rho}\nabla p + \boldsymbol{a}_{\text{other}} \tag{8}$$

Constitutive equations A constitutive equation describes the material's response to deformation by relating the current state to the stress tensor. In our

case it specifies how the deviatoric stress $\boldsymbol{S}$ changes over time. We use a model based on Hooke's law:

$$\frac{d\boldsymbol{S}}{dt} = 2\mu \left(\dot{\boldsymbol{\varepsilon}} - \boldsymbol{I} \frac{1}{3} \dot{\boldsymbol{\varepsilon}}^{1,1} \dot{\boldsymbol{\varepsilon}}^{2,2} \dot{\boldsymbol{\varepsilon}}^{3,3} \right) + \text{rotation terms} \tag{9}$$

The shear modulus μ is a material parameter. The rotation terms ensure independence from the orientation of the coordinate system. Stress is only be applied for deformation not for rotation. They are derived using the Jaumann rate and read [13]

$$\text{rotation terms} = \boldsymbol{S}\boldsymbol{R} - \boldsymbol{R}\boldsymbol{S}. \tag{10}$$

The strain rate tensor $\dot{\boldsymbol{\varepsilon}}$ and rotation rate tensor $\boldsymbol{R}$ are calculated from spatial derivatives of the velocity:

$$\dot{\boldsymbol{\varepsilon}}^{\alpha\beta} = \frac{1}{2} \left(\frac{\partial \boldsymbol{v}^{\alpha}}{\partial \boldsymbol{x}^{\beta}} + \frac{\partial \boldsymbol{v}^{\beta}}{\partial \boldsymbol{x}^{\alpha}} \right) \tag{11}$$

$$\boldsymbol{R}^{\alpha\beta} = \frac{1}{2} \left(\frac{\partial \boldsymbol{v}^{\alpha}}{\partial \boldsymbol{x}^{\beta}} - \frac{\partial \boldsymbol{v}^{\beta}}{\partial \boldsymbol{x}^{\alpha}} \right) \tag{12}$$

Therefore a change of the stress results from adjacent parts of the body moving with different velocities [13, 36].

Plasticity The above constitutive equations describe a fully elastic material. It will return to its initial state no matter how big the deformation. To model realistic solids, plastic deformation – deformation resulting in an irreversible permanent change of the material – must also be modeled. To achieve this, the deviatoric stress is limited by multiplying it with the factor [26, 36]

$$f_y = \min \left(Y_0^2 / 3J_2, 1 \right) \tag{13}$$

$$J_2 = \frac{1}{2} \sum_{ij} \boldsymbol{S}^{ij} \boldsymbol{S}^{ij} \tag{14}$$

The yield stress Y_0 could be set as a material parameter. To model granular material we use the Mohr-Coloumb yield criterion instead. That means calculating the yield stress as [2]:

$$Y_0 = \tan(\Phi)\, p + c \tag{15}$$

Where Φ is the internal friction angle and c the cohesion coefficient. Imagine forming a pile of sand. Once it gets steeper than a certain angle, it becomes unstable and sand slides down the pile. Applying pressure or increasing the cohesion, e.g. by adding water, will allow a steeper pile of sand.

[2] This is taken from the implementation of Schaefer et al. [36], however, we could not find this exact formulation in literature.

Equation of state An equation of state relates pressure to density. Some equations of state are based on temperature or internal energy. Here we choose the *Murnaghan* equation of state. While it is limited to isothermal compression – where the temperature of the body remains constant over time – we do not need track internal energy at all.

$$p = \frac{K_0}{n_M}\left[\left(\frac{\rho}{\rho_0}\right)^{n_M} - 1\right] \tag{16}$$

The equation relates the current density ρ to the density of the material when at rest ρ_0, creating a pressure to counteract compression and extension. The bulk modulus K_0 and the constant n_M are material parameters and control the strength of the materials reaction. [24, 36]

2.3 Limitations

Our model solely simulates pebbles under self gravity and resolves their collisions. No additional sticking after a collision is simulated, and resulting objects are purely gravitational bound. All particles in the simulation have the same initial size and material properties. We also neglect the effects of remaining gas, as well as different temperatures.

3 Numerical model

3.1 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (*SPH*) is an interpolation technique used to approximate continuous vector and scalar fields using particles. It also allows to easily compute spatial derivatives of those fields. Initially developed for fluid simulation, it can also be extended to handle solids. In order to calculate the value of property A at position $\boldsymbol{r}$ we take a weighted sum over the value of A at all particles in the vicinity of $\boldsymbol{r}$ (also shown in figure 6).

$$A(\boldsymbol{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\boldsymbol{r} - \boldsymbol{r}_j, h) \tag{17}$$

The smoothing kernel W assigns bigger weights to values of particles closer to position $\boldsymbol{r}$. You can imagine W like a gaussian.

In practice however, a function with compact support radius h (*smoothing length*) is used as the smoothing function. Multiple kernel functions have been proposed in literature [9, 29, 30], but the cubic B-spline (figure 7) seems to be most widely used.

$$W_{spline}(\boldsymbol{x}, h) = \frac{8}{\pi h^3}\begin{cases} 1 - 6\left(\frac{||\boldsymbol{x}||}{h}\right)^2 + 6\left(\frac{||\boldsymbol{x}||}{h}\right)^3, & 0 \leq \frac{||\boldsymbol{x}||}{h} \leq \frac{1}{2} \\ 2\left(1 - \frac{||\boldsymbol{x}||}{h}\right)^3, & \frac{1}{2} < \frac{||\boldsymbol{x}||}{h} \leq 1 \\ 0, & \frac{||\boldsymbol{x}||}{h} > 1 \end{cases} \tag{18}$$

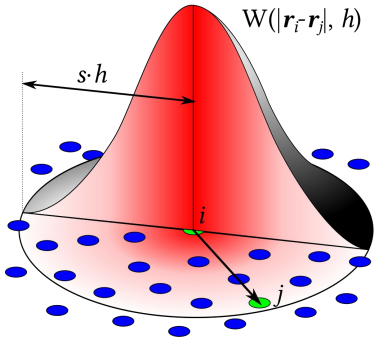

Fig. 6: Visualization of the SPH interpolation method [39].

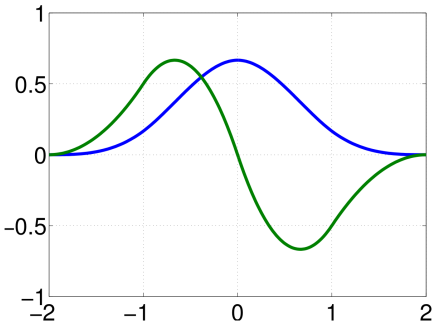

Fig. 7: The cubic spline kernel in blue with its first derivative in green [35].

In this paper as a shorthand for $W(\boldsymbol{r}_i - \boldsymbol{r}_j, h)$ we write W_{ij}.

To calculate spatial derivatives using the SPH technique, we use derivatives of the smoothing kernel. The approximation of the gradient for example reads:

$$\nabla A(\boldsymbol{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\boldsymbol{r} - \boldsymbol{r}_j, h) \tag{19}$$

Formulations for curl and divergence can be found e.g. in [29].

3.2 Discretization of the physical model

We use SPH to discretize our physical model. Therefore our domain is filled with particles storing the relevant physical properties. They are updated every step of the simulation according to the physical model (section 2). To solve the partial differential equations numerically they need to be rewritten according to the SPH technique. For our model, the equations that need to be discretized are the time derivatives of density, velocity and stress.

Density Typically the equation of mass conservation (equation 4) is not solved directly when using SPH. Instead, the density ρ of each particle is calculated in each simulation step using the SPH method (equation 17):

$$\rho = \sum_j m_j W_{ij} \tag{20}$$

For solid bodies this estimation leads to a density error at the edges of the objects as shown in figure 8. Therefore, when simulating solid bodies, it can be advantageous to evolve density over time. The time derivatives for the density of particle i is given by [13, 36]:

$$\frac{d\rho}{dt} = \rho \sum_j \frac{m_j}{\rho_j} (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \nabla W_{ij} \tag{21}$$

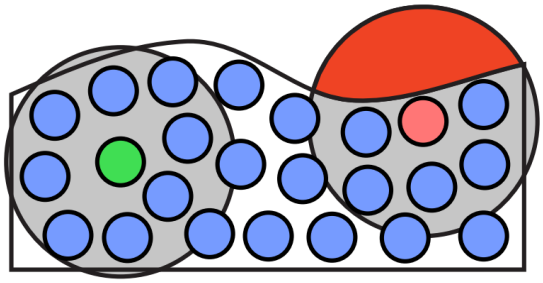

Fig. 8: The black line shows the edges of the solid body. Small circles indicate SPH-particles, big circles their smoothing radius h. The green particle in the center of the body has a full neighborhood, while the red particle at the edge of the body has less neighbors. Therefore, the red particles density is underestimated when using equation 20 [18].

Velocity Particle velocities also needs to be updated. This is done by applying the SPH-method to the equation of momentum conservation (equation 5). Directly applying the SPH gradient formulation (equation 19) unfortunately leads to asymmetric accelerations, as can be seen when considering only two particles [28]. Instead a corrected formulation is used to keep symmetry and ensure conservation of momentum [7].

$$\frac{d\boldsymbol{v}_i}{dt} = \sum_j \left[\frac{\sigma_i}{\rho_i^2} + \frac{\sigma_j}{\rho_j^2} \right] \nabla W_{ij} \tag{22}$$

To this we add gravitational acceleration (equation 3) and artificial viscosity (section 3.3).

Stress The equation of state (equation 16) can directly be used to calculate each particles pressure from its density. For the simulation we also need to evolve the deviatoric stress over time as explained in section 2.2. As the definitions of $\dot{\varepsilon}$ and $\boldsymbol{R}$ include spatial derivatives of the velocity, we calculate them using the SPH method [7]:

$$\dot{\boldsymbol{\varepsilon}}_i^{\alpha\beta} = \frac{1}{2\rho_i} \sum_j m_j \left[\left(\boldsymbol{v}_j^\alpha - \boldsymbol{v}_i^\alpha \right) \frac{\partial W_{ij}}{\partial \boldsymbol{x}_i^\beta} + \left(\boldsymbol{v}_j^\beta - \boldsymbol{v}_i^\beta \right) \frac{\partial W_{ij}}{\partial \boldsymbol{x}^\alpha} \right] \tag{23}$$

$$\boldsymbol{R}_i^{\alpha\beta} = \frac{1}{2\rho_i} \sum_j m_j \left[\left(\boldsymbol{v}_j^\alpha - \boldsymbol{v}_i^\alpha \right) \frac{\partial W_{ij}}{\partial \boldsymbol{x}_i^\beta} - \left(\boldsymbol{v}_j^\beta - \boldsymbol{v}_i^\beta \right) \frac{\partial W_{ij}}{\partial \boldsymbol{x}^\alpha} \right] \tag{24}$$

With this, equation 9 can be solved. After evolving the deviatoric stress in time, plasticity is applied by multiplication with f_y (equation 13).

3.3 Artificial viscosity

In addition to the discretized physical model, when using SPH, a non-physical viscosity is commonly added as a correction factor. The addition of such an artificial viscosity to SPH is necessary to handle shock waves in fluids and to prevent particles from penetrating each other in a non-physical way. Here, we use the improved artificial viscosity formulation by Monaghan [27]. It is computed as

$$\Pi_{ij} = -\frac{\alpha}{2}\frac{w_{ij}v_{ij}^{sig}}{\rho_i} \tag{25}$$

and can be treated as an additional pressure term. How much the viscosity will slow the movement of approaching particles is controlled by α. The component of relative velocity along the line connecting both particles is calculated as

$$w_{ij} = \begin{cases} \frac{\boldsymbol{v}_{ij}\cdot\boldsymbol{r}_{ij}}{||\boldsymbol{r}_{ij}||}, & \boldsymbol{v}_{ij}\cdot\boldsymbol{r}_{ij} > 0 \\ 0, & \boldsymbol{v}_{ij}\cdot\boldsymbol{r}_{ij} \leq 0 \end{cases} \tag{26}$$

with $\boldsymbol{v}_{ij} = \boldsymbol{v}_i - \boldsymbol{v}_j$ and $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$. It is set to zero if the particles move away from each other. This way the viscosity is only active in cases of compression. The signal velocity

$$v_{ij}^{sig} = c_i + c_j - 3w_{ij} \tag{27}$$

relates the relative velocity w_{ij} to the local speed of sound c, resulting in a quantity comparable to a mach number.

This viscosity does not vanish in shear flow, when particles move parallel to each other with different velocities. To disable the viscosity in shear flow we multiply Π_{ij} by $(f_i^{visc} + f_j^{visc})/2$, where

$$f_i^{visc} = \frac{||\nabla\cdot\boldsymbol{v}_i||}{||\nabla\cdot\boldsymbol{v}_i|| + ||\nabla\times\boldsymbol{v}_i|| + 0.0001c_i/h}. \tag{28}$$

The velocity divergence and curl ($\nabla\cdot\boldsymbol{v}_i$ and $\nabla\times\boldsymbol{v}_i$) are computed using the corresponding SPH equations [1].

3.4 Time integration

After calculating the time derivatives of all quantities, they need to be advanced in time by means of numerical integration. We employ the *Leapfrog* algorithm in its kick-drift-kick form with automatic timestep adjustment. Due to its symplectic nature it gives good results with low computational effort. It is also relatively simple to implement.

Acceleration is assumed to be constant for the time Δt. For the timestep at time t, first velocity is advanced to its value at $t + \Delta t$ using the acceleration at time t:

$$\boldsymbol{v}_{t+\Delta t/2} = \boldsymbol{v}_t + \boldsymbol{a}_t\frac{\Delta t}{2}. \tag{29}$$

This is called the kick-phase. During the following drift-phase that new velocity is used to advance the particle positions

$$\boldsymbol{r}_{t+\Delta t} \;=\; \boldsymbol{r}_t + \boldsymbol{v}_{t+\Delta t/2}\Delta t. \tag{30}$$

We can then compute acceleration $\boldsymbol{a}_{t+\Delta t}$ using positions at $t+\Delta t$ and velocities at $t+\Delta t/2$. The final kick phase updates the velocity again to

$$\boldsymbol{v}_{i+\Delta t} = \boldsymbol{v}_{i+\Delta t/2} + \boldsymbol{a}_{t+\Delta t}\frac{\Delta t}{2}. \tag{31}$$

We advance density and stress during the drift phase.

The value of Δt balances execution time and accuracy. The required timestep size depends on velocity and acceleration of the particles. We calculate a new Δt at the end of each timestep by taking the minimum of the following two criteria over all particles i.

$$\Delta t_i^{acc} = \sqrt{\frac{2\eta\epsilon_i}{||\boldsymbol{a}_i||}} \tag{32}$$

$$\Delta t_i^{vel} = \frac{C_{\text{courant}} h_i}{\max_j \left(v_{ij}^{sig}\right)} \tag{33}$$

The first criterion is based on the particles acceleration, it handles timestep requirements from gravitational forces and strong reactions to collision. The second one uses the signal velocity (also used in the artificial viscosity) and therefore depends on particles relative velocities. It prevents particles from passing through each other during time integration without the chance of handling the collision. The parameters η and C_{courant} control the accuracy. Smaller values yield smaller timestep sizes [37].

Furthermore we employ a fixed and variable time step 3rd order Runge-Kutta integration scheme. For both of the algorithms a quantity q is integrated from a time step t_n to $t_n + \Delta t$ by:

$$q_{n+1} = q_n + \Delta t\left(\frac{1}{6}k_1 + \frac{4}{6}k_2 + \frac{1}{6}k_3\right) \tag{34}$$

Since the quantity is integrated by time, it is written as $q_n = q(t_n)$ and $q_{n+1} = q(t_n + \Delta t)$. For simplification the derivative is written as $f(t_n, q_n) = \frac{dq_n}{dt}$. The three substeps k_1, k_2 and k_3 are given by

$$k_1 = f(t_n, q_n) \tag{35}$$

$$k_2 = f(t_n + \frac{\Delta t}{2}, q_n + \frac{\Delta t}{2}k_1) \tag{36}$$

$$k_3 = f(t_n + \Delta t, q_n - \Delta k_1 + 2\Delta t k_2) \tag{37}$$

Using only the above equations the fixed 3rd order Runge-Kutta scheme is implemented. The disadvantage of the fixed integration scheme is the determination

of the time step. Hence we implement the variable time step 3rd order Runge-Kutta scheme as used by Schäfer et al. [36]. We calculate the difference ϵ between q_{n+1} and the Euler midpoint method q_{n+1}^{mm} by:

$$\epsilon_d = \frac{|q_{n+1}^{mm} - q_{n+q}|}{|qn + \Delta t f(t_n, q_n)|}, \text{where} \tag{38}$$

$$q_{n+1}^{mm} = q_n + \Delta t f(t_n + \frac{\Delta t}{2}, q_n + \frac{\Delta t}{2} f(t_n, y_n)) \tag{39}$$

Using the above calculated error ϵ_d the time step is increased or decreased. The error is compared to a desired relative error ϵ_r, which is given initially. The time step is computed by:

$$\Delta t_{\text{new}} = \begin{cases} \Delta t |\frac{\epsilon_r}{\epsilon_d}|^{0.3} & \epsilon_d < \epsilon_r \\ 0.9\Delta t |\frac{\epsilon_r}{\epsilon_d}|^{\frac{1}{4}} & \epsilon_d > \epsilon_r \end{cases} \tag{40}$$

4 Implementation

4.1 Algorithm overview

We now outline the algorithm for a single simulation step using equations from the previous section.

1. For all particles, calculate the velocity divergence and curl to compute the balsara factor (equation 28).
2. At the same time compute the rate of change for density (equation 21) and deviatoric stress (equation 9), as they also use spatial derivatives of the velocity.
3. For all particles calculate acceleration due to gravity (equation 3).
4. Add acceleration due to stress (equation 22) and artificial viscosity (equation 25). Stress tensor $\boldsymbol{\sigma}$ is computed from equation 7 and 16 when needed.
5. Also store $\max_j(v_{ij}^{sig})$, which is later needed to update the timestep size.
6. Update velocity $\boldsymbol{v}_t = \boldsymbol{v}_{t-\Delta t/2} + \boldsymbol{a}_t \frac{\Delta t}{2}$ for all particles. Calculate the new timestep size Δt as the minimum of equation 32 and 33 over all particles. This is omitted in the first simulation step.
7. Using the new timestep, update velocity again to $\boldsymbol{v}_{t+\Delta t_{\text{new}}/2} = \boldsymbol{v}_t + \boldsymbol{a}_t \frac{\Delta t_{\text{new}}}{2}$
8. Then advance position, density and deviatoric stress to $t + \Delta t_{\text{new}}$.

In total, we perform four loops over all particles, where the first and second are actually over all pairs of particles. The first loop contains steps 1 and 2, however, the interaction is only computed for pairs of particles which are closer than h. In the second loop, step 3 is executed for all pairs of particles, while steps 4 and 5 are again only executed for pairs with a separation of less than h. The third loop executes step 6, before the fourth loop finishes the simulation step with 7 and 8.

To get a fast running simulation despite the $O(N^2)$ complexity, we use *CUDA* to exploit the massively parallel architecture of GPUs. Together with special optimization techniques that allows us to even reach interactive frame-rates for low resolution runs, as detailed in the following sections.

Furthermore we save the simulation results in an HDF5 file. This has the advantage of saving large simulation data efficiently in a binary-format. Since the original HDF5 library is written in C, we use the C++ HighFive[3] library. The simulation is saved for certain time steps given a delta save time in the settings of our simulation. We save each particle and its corresponding attributes in an HDF5 data set. Which attributes to store is a further setting of our simulation. The time step of the simulation is saved in the file name.

4.2 CUDA

GPUs have evolved from specialized hardware for 3D-rendering into programmable, parallel many-core processors. They surpass CPUs of traditional architecture in memory bandwidth and computational power. More transistors are used for actual calculation instead of cache and control functions. To hide latency and yield good performance, the problem needs to expose a lot of parallelism. Using tens of thousands of threads ensures, that there are always some threads not waiting on a memory operation and ready to be executed.

CUDA was introduced by *NVIDIA* in 2006 to allow general computations to be performed on GPUs. Other options to program GPUs are for example OpenCL, the directive based OpenACC as well as functionality bundled with graphics framework, like OpenGL and DirectX. While CUDA does only work with *NVIDIA* GPUs, it does allow code to be written in C or C++ and therefore integrates well with the rest of the code. It also comes with many useful tools like a debugger and profilers. Most of our simulation code is written in CUDA using C++. We use OpenGL for the optional visualization module.

When programming in CUDA, C++ functions can be marked as a *Kernel function* which allows them to run on the GPU. When executing a Kernel function on the GPU, the amount of threads to launch is specified. Every thread executes the Kernel once, providing a unique id. I.e. all threads execute the same code, but can use different data. This can be compared to a parallel region in openMP.

In addition threads are grouped into blocks. Threads in the same block are guaranteed to be executed by the same core. This allows the use of special synchronization and communication instructions as well as fast on chip memory, called *Shared Memory*. Threads of one block are further grouped into subgroups of 32 threads each, called *warps*. A CUDA core executes threads of one warp at the same time (similar to *SIMD* instructions). Therefore block size should be set to a multiple of 32. Long branches should also be avoided. All threads of a warp that do not take a particular branch are paused and wait for the other threads to execute the branch.

[3] `https://github.com/BlueBrain/HighFive`

For more information on CUDA and performance tuning, the reader is referred to the programming guide and tuning guide part of the *CUDA Toolkit Documentation* [32].

4.3 Direct gravitational interaction

Our implementation closely follows the algorithm presented in Section 4.1. We implement a CUDA Kernel for each of the four loops and launch one thread per particle. Even though the GPU does not rely on cache as heavily as the CPU, it is still advantageous for threads in the same block to access adjacent memory addresses. Therefore, we store our particle attributes in a structure of arrays style (one continuous array storing a value per particle for a particular attribute). The *template metaprogramming* features of C++14 allow flexible adjustment of this particle structure.

To further increase speed, we reduce the number of memory operations during the parts of the code that loop over all pairs of particles, as they make up for the majority of the overall runtime. To do that, we follow the techniques from [33]. Particles are partitioned into computational tiles matching the CUDA block-size. Each thread of a block loads particle-data corresponding to one particle of the current tile into shared memory. Then each thread calculates interactions of its own particle with all the particles of the current tile, using the data from shared memory. The process is repeated with the next tile until every particle has interacted with every other particle.

4.4 Barnes and Hutt Tree

While the above implementation is fast for relatively small particle numbers, it still has $O(N^2)$ complexity, making it unfeasible for larger simulations. One solution is a tree method proposed by Barnes and Hutt [2]. The domain is hierarchically divided using a tree structure. Actual particles are leaves of the tree. For every node the total mass as well as the center of mass of all children is computed. To calculate gravitational forces, the tree is traversed for every particle. If a particular node is far away and small, we use the data stored in it to approximate the gravitational influence of all its children. If the node is close and big, we continue the traverse on the next level, performing the same test on its children. The decision, whether or not a node is accepted for approximation, is made using the *Multipole Acceptance Criterion* (MAC). Several such criteria have been proposed in Literature. We implement the MAC found in [6]. It features a parameter θ, which is used to balance accuracy against computational time.

Hernquist and Katz combined the tree method with SPH and called the resulting algorithm TREESPH [16]. It is used in most SPH codes that include self gravity (e.g. [37]).

While such an algorithm might not seem to be well suited for the GPU at first glance, a lot of research has been done in this field [6, 11, 15, 25, 31]. We mostly follow *BONSAI* by Bedorf et al. [6] in our implementation. While not finished at the time of writing, test runs indicate that it increases the number

of particles that can be simulated at interactive framerates on consumer grade hardware by a factor of four.

4.5 Interactive features

The code includes a basic visualization tool based on OpenGL. It can be enabled or disabled at compile-time depending on the capabilities of the machine used. If enabled, the simulation can be viewed in 3D space at the same time it is computed. If disabled, long term simulations can be run on remote servers without graphical output and supercomputing systems. In any case, the results after each simulation step are stored as *hdf5* or tabulator separated text files to allow further analysis.

While the number of particles can be varied to achieve different resolution at the cost of higher computational effort, multiple other settings exist to balance accuracy against speed. The timestep size can be adjusted indirectly, by tuning the accuracy parameter η and C_{courant}. When using the gravitational tree algorithm, θ can be adjusted. At compile time, the user can select between single and double precision floating point arithmetic, which has a big impact on performance especially with consumer grade GPUs. Finally, there is an option to enable the use of faster intrinsic math functions by the CUDA compiler.

Therefore, very fast interactive test runs as well as high accuracy production runs can be performed, using the same physical, as well as numerical model. Errors resulting from the reduced accuracy in the different settings are investigated in the next section.

5 First results

To test the influence of the different accuracy settings on performance and accuracy of the simulation, we performed several test runs with different parameters[4]. The initial conditions consist of 16384 particles uniformly random distributed in a sphere of radius 1. The velocity is initialized with a solid body rotation. Simulation where performed on NVIDIA Tesla V100 GPUs using the direct gravity algorithm (section 4.3). We simulated 6 time units for every setup, which corresponds to about half an orbit of the pebble cloud around the central star.

Using the HDF5 save files, we analyzed the results with python scripts. The first script calculates linear and angular momentum for each particle and writes the results in the corresponding row of the save file. This file is then used in our analysis script. For better visualization we calculate the linear and angular momentum for the system and calculate its length. This is done for each HDF5 file in the folder, thus for each time step. The vector lengths are stored in a python list and used for plotting the results. To generalize this script a .csv file can be passed to the script which contains information about the different

[4] These tests have been performed and analyzed by Emma Kraft. All diagrams in this section were compiled by Emma Kraft.

scenarios used as label in figures 9 and 10. The plots are saved as .svg, so that the data can be viewed more clearly.

The simulation tuned for precision took 0.49 hours, while the simulation tuned for speed took only 0.25 hours. We compare linear and angular momentum conservation between simulations in figure 9 and 10. Both diagrams show the simulations with more accurate settings to perform better, especially after around 3 time units. That is when more and more particles get close to each other and the SPH interactions become more important than the gravitational interactions. SPH seems to have a stronger dependency on numerical precision than the simulation of gravitational interactions.

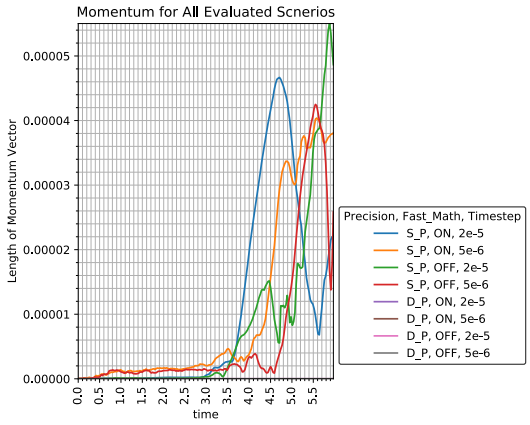

Fig. 9: Linear momentum conservation for all test simulations.

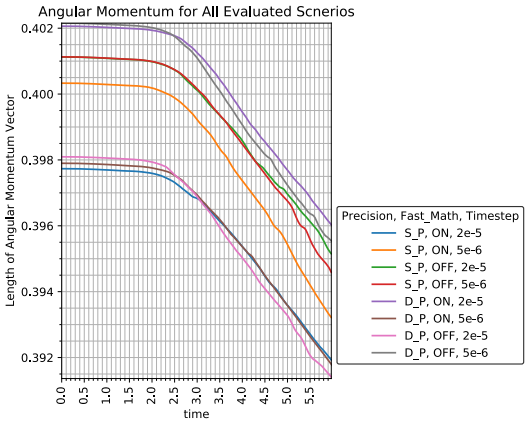

Fig. 10: Angular momentum conservation for all test simulations.

The difference in execution time is even bigger when using more particles, or on consumer grade GPUs, as double precision calculations are slower. Users have to decide for themselves what errors are acceptable for their particular use-case. Our code gives them the ability to compare simulations on different accuracy levels and the flexibility to save computational time if larger errors can be tolerated.

6 Conclusion and future work

We have presented a physical model for the simulation of a collapsing dust cloud. The model was then discretized using Smoothed Particle Hydrodynamics and implemented on the Graphics Processing Unit using CUDA. We also showed some preliminary results in the previous section. Our code features a build in interactive visualization. While the accuracy is lower in interactive mode, it allows for rapid testing and experimentation. Findings can then be verified by performing a high accuracy simulation on an HPC system.

Typically it takes days or even weeks to perform physical simulations, not to mention visualize the results. We think that providing build-in visualization features and the ability to perform interactive test runs can decrease the time needed to obtain results in numerical experiments.

This project is still a work in progress and we hope to increase speed and particle numbers by including the a tree-algorithm as detailed in section 4.4. We are also working on improving our model to allow dust particles to stick together in a physical way and exploring different techniques for time integration.

References

1. Balsara, D.S.: von Neumann stability analysis of smoothed particle hydrodynamics-suggestions for optimal algorithms. Journal of Computational Physics **121**(2), 357–372 (oct 1995). https://doi.org/10.1016/S0021-9991(95)90221-X
2. Barnes, J., Hut, P.: A hierarchical o (n log n) force-calculation algorithm. nature **324**(6096), 446–449 (1986)
3. Bate, M.R.: On the diversity and statistical properties of protostellar discs. Monthly Notices of the Royal Astronomical Society **475**(4), 5618–5658 (2018)
4. Bate, M.R., Bonnell, I.A., Bromm, V.: The formation mechanism of brown dwarfs. Monthly Notices of the Royal Astronomical Society **332**(3), L65–L68 (2002)
5. Bate, M.R., Bonnell, I.A., Bromm, V.: The formation of a star cluster: predicting the properties of stars and brown dwarfs. Monthly Notices of the Royal Astronomical Society **339**(3), 577–599 (2003)
6. Bédorf, J., Gaburov, E., Portegies Zwart, S.: A sparse octree gravitational N-body code that runs entirely on the GPU processor. Journal of Computational Physics **231**(7), 2825–2839 (2012). https://doi.org/10.1016/j.jcp.2011.12.024
7. Benz, W., Asphaug, E.: Simulations of brittle solids using smooth particle hydrodynamics. Computer Physics Communications **87**(1-2), 253–265 (may 1995). https://doi.org/10.1016/0010-4655(94)00176-3

8. Bonet, J., Wood, R.: J. Bonet, R. D. Wood, Nonlinear Continuum Mechanics for Finite Element Analysis, Cambridge University Press, Cambridge, UK, vol. 24 (03 2008). https://doi.org/10.1017/CBO9780511755446
9. Dehnen, W., Aly, H.: Improving convergence in smoothed particle hydrodynamics simulations without pairing instability. Monthly Notices of the Royal Astronomical Society **425**(2), 1068–1082 (2012)
10. Desbrun, M., Gascuel, M.P.: Smoothed Particles: A new paradigm for animating highly deformable bodies pp. 61–76 (1996)
11. Gaburov, E., Bédorf, J., Zwart, S.P.: Gravitational tree-code on graphics processing units: Implementation in CUDA. In: Procedia Computer Science. vol. 1, pp. 1119–1127 (may 2010). https://doi.org/10.1016/j.procs.2010.04.124
12. Gingold, R.A., Monaghan, J.J.: Smoothed particle hydrodynamics: theory and application to non-spherical stars. Monthly Notices of the Royal Astronomical Society **181**(3), 375–389 (1977). https://doi.org/10.1093/mnras/181.3.375
13. Gray, J.P., Monaghan, J.J., Swift, R.P.: SPH elastic dynamics. Computer Methods in Applied Mechanics and Engineering **190**(49-50), 6641–6662 (oct 2001). https://doi.org/10.1016/S0045-7825(01)00254-7
14. Harada, T., Koshizuka, S., Kawaguchi, Y.: Smoothed Particle Hydrodynamics on GPUs. Computer Graphics International pp. 63–70 (2007)
15. Hegeman, K., Carr, N.A., Miller, G.S.P.: Particle-based fluid simulation on the GPU. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **3994 LNCS**, 228–235 (2006)
16. Hernquist, L., Katz, N.: TREESPH - A unification of SPH with the hierarchical tree method. The Astrophysical Journal Supplement Series **70**, 419 (1989). https://doi.org/10.1086/191344
17. Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A., Teschner, M.: SPH Fluids in Computer Graphics. Eurographics (2), 21–42 (2014)
18. Koschier, D., Bender, J., Solenthaler, B., Teschner, M.: Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids Dan (2019)
19. Laibe, G., Gonzalez, J.F., Fouchet, L., Maddison, S.T.: SPH simulations of grain growth in protoplanetary disks. Astronomy & Astrophysics **487**(1), 265–270 (2008)
20. Libersky, L.D., Petschek, A.G.: Smooth particle hydrodynamics with strength of materials. In: Advances in the free-Lagrange method including contributions on adaptive gridding and the smooth particle hydrodynamics method, pp. 248–257. Springer (1991)
21. Libersky, L.D., Randles, P.W., Carney, T.C., Dickinson, D.L.: Recent improvements in SPH modeling of hypervelocity impact. International Journal of Impact Engineering **20**(6-10), 525–532 (1997)
22. Lucy, L.B.: A numerical approach to the testing of the fission hypothesis. The astronomical journal **82**, 1013–1024 (1977)
23. Makino, J., Hut, P.: Moving Stars Around. The Art of Computational Science **1** (2007)
24. Melosh, H.J.: Impact cratering : a geologic process. Oxford University Press ; Oxford : Clarendon Press, 1989 (1989)
25. Miki, Y., Umemura, M.: GOTHIC: Gravitational oct-tree code accelerated by hierarchical time step controlling. New Astronomy **52**, 65–81 (oct 2017). https://doi.org/10.1016/j.newast.2016.10.007

26. von Mises, R.: Mechanik der festen Körper im plastisch-deformablen Zustand. Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse **4**, 582–592 (1913)
27. Monaghan, J.J.: SPH and Riemann Solvers. Journal of Computational Physics **136**(2), 298–307 (1997). https://doi.org/10.1006/jcph.1997.5732
28. Monaghan, J.J.: Smoothed particle hydrodynamics. Reports on Progress in Physics **68**(8), 1703–1759 (2005). https://doi.org/10.1088/0034-4885/68/8/R01
29. Monaghan, J., J.: Smoothed Particle Hydrodynamics. Annual Review of Astronomy and Astrophysics **30**, 543–574 (1992). https://doi.org/10.1146/annurev.aa.30.090192.002551
30. Müller, M., Charypar, D., Gross, M.: Particle-Based Fluid Simulation for Interactive Applications. Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation (5), 154–159 (2003)
31. Nakasato, N.: Implementation of a parallel tree method on a GPU. Journal of Computational Science **3**(3), 132–141 (may 2012). https://doi.org/10.1016/j.jocs.2011.01.006
32. NVIDIA: Cuda toolkit documentation. `https://docs.nvidia.com/cuda/` (2020), [Online; accessed 29-February-2020]
33. Nyland, L., Harris, M., Prins, J.: Fast N-Body Simulation with CUDA. Simulation **3**(1), 677–696 (2007)
34. Randles, P., Libersky, L.D.: Smoothed particle hydrodynamics: some recent improvements and applications. Computer methods in applied mechanics and engineering **139**(1-4), 375–408 (1996)
35. Röthlin, M., Klippel, H., Wegener, K.: Meshless Methods for Large Deformation Elastodynamics (jul 2018)
36. Schäfer, C., Riecker, S., Maindl, T.I., Speith, R., Scherrer, S., Kley, W.: A smooth particle hydrodynamics code to model collisions between solid, self-gravitating objects. Astronomy &Astrophysics **19** (2016)
37. Springel, V.: The cosmological simulation code GADGET-2. Monthly Notices of the Royal Astronomical Society **364**(4), 1105–1134 (2005). https://doi.org/10.1111/j.1365-2966.2005.09655.x
38. Wikipedia: Hl tauri image. `https://en.wikipedia.org/wiki/Protoplanetary_disk#/media/File:HL_Tau_protoplanetary_disk.jpg` (2020), [Online; accessed 29-February-2020]
39. Wikipedia: Sph interpolation image. `https://en.wikipedia.org/wiki/Smoothed-particle_hydrodynamics#/media/File:SPHInterpolationColorsVerbose.svgVerbose.svg.png` (2020), [Online; accessed 29-February-2020]
40. Wikipedia: Ultima thule image. `https://de.wikipedia.org/wiki/(486958)_2014_MU69#/media/Datei:UltimaThule_CA06_color_20190516.png` (2020), [Online; accessed 29-February-2020]