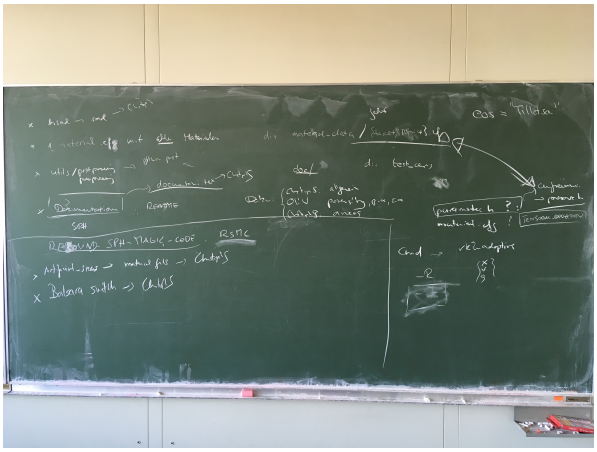
Documentation miluphCUDA



Version: November 25, 2020

written by
Christoph Burger, Christoph Schäfer, Oliver Wandel
Institut für Astronomie und Astrophysik
Eberhard-Karls-Universität Tübingen

Contents

1. Introduction

miluphCUDA was originally the CUDA port of the code *miluph* (see http://www.tat.physik.uni-tuebingen.de/~schaefer/miluph.html). It's a Smooth Particle Hydrodynamics (SPH) code for the simulation of fluids as well as solid and porous bodies. *miluphCUDA* is capable of using various different equation of state (see Sect. 4.5) and the material strength model and a damage model introduced by Benz & Asphaug for the simulation of brittle solid materials (see Sect. 4.7). Self-gravity can be included and is solved using a Barnes-Hut tree (see Sect. 5.). The code can be used for the simulation of high-velocity impacts and/or self gravitating astrophysical objects or mixed hydro-solid simulations.

The code and its numerics are well described in the publication:

Schäfer, C.; Riecker, S.; Maindl, T.; Speith, R.; Scherrer, S.; Kley, W.: A Smooth Particle Hydrodynamics Code to Model Collisions Between Solid, Self-Gravitating Objects,

published in Astronomy & Astrophysics.

Please cite this paper when using our code.

Copyright and Authors

Christoph Schäfer and Sven Riecker wrote the basic version of *miluphCUDA*.

Later additions (chronologically):

- Soil material model: Samuel Scherrer
- Johnson-Cook material model: Janka Werner
- Porosity model: Oliver Wandel
- ANEOS support: Christoph Burger
- Drucker-Prager and Mohr-Coulomb material model: Christoph Schäfer
- Navier-Stokes equation: Christoph Schäfer

Authors of some additional tools in `test_cases/` or `utils/`:

- Christoph Burger
- Thomas I. Maindl

2. Quick Start Guide

2.1 Installation steps

1. Install CUDA, e.g., from https://developer.nvidia.com/cuda-downloads
2. Check out *miluphCUDA* from the git repo, e.g.,
   `git clone https://github.com/christophmschaefer/miluphcuda.git`
3. Install *libconfig*: Download
   `http://www.hyperrealm.com/libconfig/libconfig-1.4.9.tar.gz`
   (or a newer version) and

CUDA version	Nvidia driver
6.0	331.113
6.5	
7.0	346.46
7.5	352.39, 367.27
8.0	375.26, 375.66, 384.111, 390.67
9.0	390.67
10.0	430.14

Table 1: Tested CUDA versions with Nvidia-driver versions

Unsupported cuda versions so far: $<= 5.0$.
We have one case of an user reporting that the combination of gcc- 4.4.7 and CUDA devkit 7.5 on K20 cards fails to run the SOLID version of the code. If you happen to provide access to one of these cards for us, please contact us.

- `sudo ./configure`
- `sudo make`
- `sudo make install`

or
`apt-get install libconfig9-dev`
on Ubuntu/Debian

4. If you want to use HDF5, you need to install *libhdf5* and *libhdf5-dev*
5. Make relevant changes to *parameter.h* to specify all compile-time options
6. Make relevant changes to the *Makefile*: Change the `CUDA_DIR` variable according to your CUDA version. Make sure to use the correct `GPU_ARCH` corresponding to your Nvidia-GPU model. Otherwise the code will *not* work and crash or give wrong results! Tested CUDA versions with Nvidia-driver versions are given in Tab. 2.1.

   Also change the compiler and linker flags in the Makefile according to your system (e.g., `-L` and `-I` for *libhdf5* and *libconfig*).

2.2 Usage of the code

The code and its numerics are comprehensively described in the publication:
Schäfer, C.; Riecker, S.; Maindl, T.; Speith, R.; Scherrer, S.; Kley, W.: A Smooth Particle Hydrodynamics Code to Model Collisions Between Solid, Self-Gravitating Objects,
which is published in Astronomy & Astrophysics.
Please cite this paper when using our code.

Compile time options

The number of threads for some CUDA kernels can be set as compile time options in *timeintegration.h*. You may need to change (if so then probably lower) some values depending on your hardware.

All compile time options related to the physical model have to be set in *parameter.h*! Please edit and check *parameter.h*. There are comments for each `#define`.

Command line options

A lot of important settings have to be provided at the command line (see Tab. 2, and also `./miluphcuda --help`)
One of the most important switches is `--verbose`, which provides much more helpful output.

Input file

In order to run a simulation, you need a simulation input file, that includes at least the coordinates and masses of your initial SPH particle distribution (in case of a HYDRO simulation, for SOLID simulations more columns in the input file are required). Please use
`./miluphcuda --format`
to determine the correct input file format, and to display the output file format.

Material parameter file

The required material parameters are set in the file *material.cfg*, which is passed via the command line argument `-m`. Refer to the available materials in `material_data/` and the examples in `test_cases/` for the different needed material parameters.

Usage example

Example of miluphCUDA invocation (from the colliding rubber rings example in `test_cases/colliding_rings/`):

```
# set the correct LD_LIBRARY_PATH to the cuda libs
export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64
# invoke miluphCUDA: use the rk2_adaptive integrator, use HDF5
# output and write 500 output files at simulation times
1.0
2.0
3.0
(...)
500.0
# the files will be named
rings.0001
rings.0002
rings.0003
(...)
rings.0500
# the used miluphCUDA command line is
./miluphcuda -v -I rk2_adaptive -n 500 -H -t 1.0 -f rings.0000 -m material.cfg
```

Porosity usage

If you want to model one or more materials which are porous, you have to change some things–(*) marked points need to be done for simulations with at least 1 porous material, while (**) marked points need to be done if the regarded material is porous.
In *parameter.h*:

Table 2: Available command line options.

Option	Purpose
`-h, --help`	print help message
`-G, --information`	print information about detected Nvidia GPUs on this host
`-d, --device_id <int>`	try to use GPU device with id `<int>` for computation (default: 0)
`-Y, --format`	print information about input and output format of the data files
`-v, --verbose`	be talkative (stdout)
`-I, --integrator`	available Integrators are `euler` (1st order), `euler_pc` and `monaghan_pc` (2nd order), `rk2_adaptive` (2nd order with adaptive time step)
`-Q, --precision`	precision of the `rk2_adaptive` integrator (default: 1e-6)
`-n, --num`	number of simulation steps
`-H, --hdf5_output`	write HDF5 output files (default is no)
`-X, --hdf5_input`	use HDF5 file for input (default is no), file *<string>.XXXX.h5* will be opened (i.e. to open e.g., *impact.0123.h5* pass `impact.0123` to option `-f`)
`-A, --no_ascii_output`	disable ASCII output files (not set by default)
`-a, --theta`	Theta Criterion for Barnes-Hut Tree (default: 0.5)
`-t, --timeperstep`	time for one simulation step (for writing to output files for adaptive integrators)
`-M, --maxtimestep`	upper limit for the timestep (`rk2_adaptive`), or timestep size for `euler`, respectively
`-T, --starttime`	start time of the simulation
`-f, --filename`	name of input data file (default: *disk.0000*); input file name format is something like *<string>.XXXX*, where *XXXX* means runlevel and zeros
`-r, --restart`	assume that ASCII input file is old output file
`-m, --materialconfig`	name of file containing material parameters for all materials in the simulation
`-k, --kernel`	specify the kernel function, available are `wendlandc2`, `wendlandc4`, `wendlandc6`, `cubic_spline`, `quartic_spline`, default: `cubic_spline`
`-s, --selfgravity`	include self-gravity
`-D, --directselfgravity`	Calculate selfgravity using direct particle-particle force and not the tree (slower)
`-L VALUE`	Check for conservation of angular momentum. (default: off) Simulations stops once the relative difference between current angular momentum and initial angular momentum is larger than VALUE.
`-g, --decouplegravity`	decouple hydro time scale from gravitational time scale
`-b, --boundary_ratio`	ratio of additional ghost boundary particles (default: 0)

- (*) you have to switch on `PALPHA_POROSITY`

In *material.cfg*:

- (**) porosity uses `EOS_TYPE_JUTZI` which is *type = 5* for the use of the Tillotson eos (recommended), or `EOS_TYPE_JUTZI_MURNAGHAN` which is *type = 6* for the use of the Murnaghan eos
- (**) this needs to be added to the eos (it defines the crush-curve and was fitted for porous pumice):

```
porjutzi_p_elastic = 1.0e6;
porjutzi_p_transition = 6.80e7;
porjutzi_p_compacted = 2.13e8;
porjutzi_alpha_0 = 3.0;
porjutzi_alpha_e = 4.64;
porjutzi_alpha_t = 1.90;
porjutzi_n1 = 12.0;
porjutzi_n2 = 3.0;
```

  The value `porjutzi_alpha_0` needs to be set to the initial distention at the start of the simulation. In this case it is 3.0 which translates to a porosity of Phi = 1 - 1/alpha_0 = 1 - 1/3.0 = 66.6%.
- (**) the value of `till_rho_0` needs to be set to the matrix density of the regarded material

In your input file:

- (*) At the very end of your input file and before you weibull the file the last two variables are `alpha_jutzi` and `p` (the data format can also be checked using the cmd-line option `-Y`, see Tab. 2). `alpha_jutzi` is the initial distention at the start of the simulation (the same as in the *material.cfg* file) and `p` needs to be set to 0 so that there isn't any pressure at the start. `p` is needed since dP = P_new - P_old is being calculated in the code.
- (**) The initial density rho (porous density) needs to be set so that rho * alpha_jutzi = rho_s (rho_s is the matrix density which is set in *material.cfg* as till_rho_0). This has to match exactly!
- (**) The mass of the particles also needs to be adjusted to being porous. Normally if you calculate the mass using the density then the mass is already the mass for the porous particle.

Please note: for the time beeing, you will also have to set the flag `flag_alpha_quad` in `pressure.cu` directly

```
int flag_alpha_quad;   /* if this flag is set -> alpha gets
            calculated by a quadradic equation and not via the crush curve */
```

2.3 A last note

We do NOT support the use of the code for military purposes.

3. Code structure

If you want to add some changes to the code, read and understand *CodingStyle.txt*.

3.1 Data structure

We use the struct `Particle` (from *miluph.h*) for storing all information about a SPH particle. The struct depends on the settings in *parameter.h*.

3.2 List of files and directory structure

Table 3 contains a list of the files that are part of the *miluphCUDA* project.
There are additional subdirectories
`utils/`
`test_cases/`
`material_data/`
`doc/`
that include pre and postprocessing utilities, test-cases, material parameters for *material.cfg*, and this documentation.

4. Physical model

In this section, we present the set of partial differential equations from continuum mechanics in Cartesian coordinates that we apply to model gas and solid bodies. Throughout this documentation, the Einstein summation rule is used, Greek indices denote spatial coordinates and run from 1 to 3.

4.1 Mass conservation

The equation of continuity is the equation of the conservation of mass. It is given in Lagrangian form by

$$\frac{\mathrm{d}\varrho}{\mathrm{d}t} + \varrho \frac{\partial v^\alpha}{\partial x^\alpha} = 0, \tag{1}$$

where ϱ denotes the density and $\mathbf{v}$ the velocity of the solid body or fluid.

4.2 Conservation of momentum

The equation of the conservation of momentum in continuum mechanics for a solid body reads in Lagrangian formulation

$$\frac{\mathrm{d}v^\alpha}{\mathrm{d}t} = \frac{1}{\varrho}\frac{\partial \sigma^{\alpha\beta}}{\partial x_\beta}, \tag{2}$$

where $\sigma^{\alpha\beta}$ is the stress tensor given by the pressure p and the deviatoric stress tensor $S^{\alpha\beta}$

$$\sigma^{\alpha\beta} = -p\delta^{\alpha\beta} + S^{\alpha\beta}, \tag{3}$$

with the Kronecker delta $\delta^{\alpha\beta}$. For a liquid or gas, the deviatoric stress tensor vanishes and the conservation of momentum is given by the Euler equation

$$\frac{\mathrm{d}v^\alpha}{\mathrm{d}t} = -\frac{1}{\varrho}\frac{\partial}{\partial x_\beta} p\delta^{\alpha\beta}. \tag{4}$$

Table 3: List of files in *miluphCUDA*.

Filename	Purpose
aneos.cu, aneos.h	functions for initializing and running ANEOS
artificial_stress.cu, artificial_stress.h	functions for artificial stress (see Gray, Monaghan, Swift 2001)
boundary.cu, boundary.h	special treatment of boundaries
checks.h	consistency checks of compile time options
cuda_utils.h	CUDA specific functions
damage.cu, damage.h	functions for the fragmentation and damage models
density.cu, density.h	calculate density or time derivative of density
device_tools.cu, device_tools.h	
euler.cu, euler.h	Euler integrator; stress tensor (regolith) and/or deviatoric stress tensor (all other solid materials)
gravity.cu, gravity.h	functions for the calculation of self gravity
internal_forces.cu, internal_forces.h	compute the change of stress tensor (regolith) and/or deviatoric stress tensor (all other solid materials)
io.cu, io.h	functions for input/output, ASCII and HDF5
kernel.cu, kernel.h	calculate SPH kernel values
linalg.cu, linalg.h	helping functions for matrix operations etcpp.
little_helpers.cu, little_helpers.h	
Makefile	Makefile for GNU Make. You'll need to make changes according to your CUDA version and your GPU architecture in this file
material.cfg	In this file, you define the material parameters for each materialID that your simulation includes.
memory_handling.cu, memory_handling.h	functions for memory allocation and copying between host and device
miluph.cu	main
miluph.h	header file with `Particle` structure definition
parameter.h	this file includes all relevant compile time options. You will need to make changes here in order to define your physical model (HYDRO, SOLID, w/o FRAGMENTATION, DIMENSION, ...)
pc_values.dat	mininum absolute values, which are needed for the predictor-corrector integration schemes
plasticity.cu, plasticity.h	material models (rheology): Drucker-Prager, Johnson-Cook, von Mises
porosity.cu, porosity.h	porosity model following Jutzi et al. (200x)
predictor_corrector.cu, predictor_corrector.h	Predictor-Corrector integrator with predictor step dt/2, following Monaghan (1983)
predictor_corrector_euler.cu, predictor_corrector_euler.h	Predictor-Corrector integrator with predictor step dt
pressure.cu, pressure.h	calculation of the pressure according to the chosen eos; currently there is, for solids: Tillotson eos, Murnaghan eos, Regolith eos (Drucker-Prager model), ANEOS; for porous solids: Jutzi eos (this is Tillotson eos with alpha), Jutzi-Murnaghan eos (this is Murnaghan eos with alpha); for gas: Ideal gas eos, Polytropic gas eos, Isothermal gas eos
rhs.cu, rhs.h	right-hand sides
rk2adaptive.cu, rk2adaptive.h	RK2 integrator with adaptive time step
soundspeed.cu, soundspeed.h	calculation of the soundspeed
stress.cu, stress.h	calculation of the stress tensor
timeintegration.cu, timeintegration.h	calculation of all internal forces of the solid or fluid; calculation of artificial viscosity and artificial stress if defined in *parameter.h*
tree.cu, tree.h	calculation of self gravity using a Barnes-Hut Tree (no quadrupoles!)
velocity.cu, velocity.h	calculate velocities

4.3 Conservation of internal energy

The equation of the conservation of the specific internal energy u for an elastic body is given in Lagrangian formulation by

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\frac{p}{\varrho}\frac{\partial v^\alpha}{\partial x^\alpha} + \frac{1}{\varrho} S^{\alpha\beta}\dot{\varepsilon}^{\alpha\beta}, \tag{5}$$

with the strain rate tensor $\dot{\varepsilon}^{\alpha\beta}$ given by eq. (8).

4.4 Constitutive equations

In contrast to fluid dynamics, the set (1;2;5) of partial differential equations is not sufficient to describe the dynamics of an elastic solid body, since the time evolution of the deviatoric stress tensor $S^{\alpha\beta}$ is not yet specified. For the elastic behavior of a solid body, the constitutive equation is based on Hooke's law. Extended to three dimensional deformations, it reads

$$S^{\alpha\beta} \sim 2\mu\left(\varepsilon^{\alpha\beta} - \frac{1}{3}\delta^{\alpha\beta}\varepsilon^{\gamma\gamma}\right). \tag{6}$$

Here, μ denotes the material dependent shear modulus and $\epsilon^{\alpha\beta}$ is the strain tensor. For the time evolution, it follows (for small deformations)

$$\frac{\mathrm{d}}{\mathrm{d}t}S^{\alpha\beta} = 2\mu\left(\dot{\varepsilon}^{\alpha\beta} - \frac{1}{3}\delta^{\alpha\beta}\dot{\varepsilon}^{\gamma\gamma}\right) + \text{rotation terms}, \tag{7}$$

where $\dot{\varepsilon}^{\alpha\beta}$ denotes the strain rate tensor, given by

$$\dot{\varepsilon}^{\alpha\beta} = \frac{1}{2}\left(\frac{\partial v^\alpha}{\partial x^\beta} + \frac{\partial v^\beta}{\partial x^\alpha}\right). \tag{8}$$

The rotation terms in eq. (7) are needed since the constitutive equations have to be independent from the material frame of reference. There are various possibilities to achieve this. The usual approach for particle codes is the Jaumann rate form. The rotation terms of the Jaumann rate form are

$$S^{\alpha\gamma}R^{\gamma\beta} - R^{\alpha\gamma}S^{\gamma\beta}, \tag{9}$$

with the rotation rate tensor

$$R^{\alpha\beta} = \frac{1}{2}\left(\frac{\partial v^\alpha}{\partial x^\beta} - \frac{\partial v^\beta}{\partial x^\alpha}\right). \tag{10}$$

Together, eqs. (7) and (9) determine the change of deviatoric stress due to deformation of the solid body.

4.5 Equation of state

The equation of state relates the thermodynamic variables density ϱ, pressure p, and specific internal energy u. We provide a short description about the equation of states that are implemented in the code.

4.5.1 Liquid equation of state

For small compressions and expansions of a liquid, the pressure depends linearly on the change of density. Assume ϱ_0 is the initial density of the uncompressed liquid. Then, the liquid equation of state reads

$$p = c_\mathrm{s}^2(\varrho - \varrho_0). \tag{11}$$

Here, c_s denotes the bulk sound speed of the liquid, which is related to the bulk modulus K_0 and the initial density by

$$c_\mathrm{s}^2 = \frac{K_0}{\varrho_0}. \tag{12}$$

If the density ϱ falls strongly beyond the value of the initial density ϱ_0, the liquid equation of state fails because the liquid forms droplets and attracting forces vanish. To account for this behavior, the pressure is set to zero as soon as the ratio ϱ/ϱ_0 drops below $0.8 - 0.95$ (Melosh, 1996).

4.5.2 Murnaghan equation of state

The Murnaghan equation of state is an extension of the liquid equation of state (see, e.g., Melosh 1996). In contrast to the liquid equation of state, the pressure now depends nonlinearly on the density

$$p = \frac{K_0}{n_M}\left[\left(\frac{\varrho}{\varrho_0}\right)^{n_M} - 1\right], \tag{13}$$

with the zero pressure bulk modulus K_0 and the constant n_M. The Murnaghan equation of state is limited to isothermal compression. Parameters for various materials can be found in the literature, e.g., Melosh (1996).

4.5.3 Tillotson equation of state

The Tillotson equation of state was originally derived for high-velocity impact simulations (Tillotson, 1962). There are two domains depending upon the material specific energy density u. In the case of compressed regions ($\varrho \geqslant \varrho_0$) and u lower than the energy of incipient vaporization E_iv the equation of state reads

$$p = \left[a_T + \frac{b_T}{1 + u/(E_0\eta^2)}\right]\varrho u + A_T\chi + B_T\chi^2, \tag{14}$$

with $\eta = \varrho/\varrho_0$ and $\chi = \eta - 1$. In case of expanded material (u greater than the energy of complete vaporization E_cv) the equation of state takes the form

$$\begin{aligned} p =& a_T\varrho u + \left[\frac{b_T\varrho u}{1 + u/(E_0\eta^2)}\right. \\ &\left. + A_T\chi\exp\left\{-\beta_T\left(\frac{\varrho_0}{\varrho} - 1\right)\right\}\right] \\ &\times \exp\left\{-\alpha_T\left(\frac{\varrho_0}{\varrho} - 1\right)^2\right\}. \end{aligned} \tag{15}$$

The symbols ϱ_0, A_T, B_T, E_0, a_T, b_T, α_T, and β_T are material dependent parameters.

In the partial vaporization $E_{\mathrm{iv}} < u < E_{\mathrm{cv}}$, p is linearly interpolated between the pressures obtained via (14) and (15), respectively. For a more detailed description see Melosh (1996).
The Tillotson equation of state has the advantage of being computational very simple, while sophisticated enough for the application over a wide regime of physical conditions, such as high-velocity collisions and impact cratering.

4.5.4 ANEOS

If you want to use the tabulated ANEOS eos for some material you need three things:

1. A file containing the ANEOS lookup-table data. These files can be generated externally with the aid of the external project *miluphANEOS*, a Fortran code that calls ANEOS subroutines directly. See there for further instructions.
2. Suitable entries in *material.cfg* for that material. See the available materials in `material_data/`.
3. Consistent initial conditions. For setting up homogeneous bodies you can use `aneos_rho_0` and `aneos_e_norm` initially (see *material.cfg*). This, and also the direct generation of equilibrated (large) bodies, can be done by the initial conditions code in the external project *Relaxation*.

4.5.5 Locally-isothermal gas

`EOS_TYPE_LOCALLY_ISOTHERMAL_GAS`

$$p = c_s^2 \varrho \tag{16}$$

4.5.6 Perfect gas equation

`EOS_TYPE_IDEAL_GAS`

$$p = (\gamma - 1)\varrho e \tag{17}$$

4.5.7 Polytropic Gas

`EOS_TYPE_POLYTROPIC_GAS`

$$p = K\varrho^{\gamma} \tag{18}$$

4.6 Plastic material equations

Together with the equation of state, the set of equations (1;2;5) can be used to describe the dynamics of a solid body in the elastic regime. We apply the von Mises yield criterion (von Mises, 1913) to model plasticity. The deviatoric stress is limited by

$$S^{\alpha\beta} \rightarrow f_Y S^{\alpha\beta}, \tag{19}$$

where the factor f_Y is computed from

$$f_Y = \min\left[Y_0^2/3J_2, 1\right], \tag{20}$$

with the second invariant of the deviatoric stress tensor J_2 given by

$$J_2 = \frac{1}{2} S^{\alpha\beta} S_{\alpha\beta}, \tag{21}$$

and the material dependent yield stress is Y_0.

4.7 Damage model for brittle material

Since one major application of the newly developed code includes the simulation of the collision between brittle planetesimals, we included a fragmentation model. Physically, fracture is related to the failure of atomic or molecular bonds. If an applied force on a solid brittle body is large enough to break the atomic bonds, the material begins to fracture and develop cracks. A successful continuum model for fragmentation was derived by Grady and Kipp (1980) and first implemented in a SPH code by Benz and Asphaug (1995). The model is based on two assumptions: brittle materials contain so-called flaws that are sources of weakness leading to activation and growth under tensile loading, and dynamic fracture depends on the rate of tensile loading. In other words, material that is slowly pulled apart develops fewer cracks than material that experiences higher strain rates, since one single crack cannot relieve the higher tensile stress, and hence more flaws get activated leading to more cracks and consequently more damage.

The scalar parameter damage d parametrizes the influence of cracks on a given volume

$$0 \leqslant d \leqslant 1, \tag{22}$$

and is defined in a way that $d = 0$ represents an undamaged, intact material and $d = 1$ a fully damaged material that cannot undergo any tension or deviatoric stress. In this manner, a material with $d = 0.5$ experiences half the stress and tension of undamaged material with $d = 0$.

Damage reduces the strength under tensile loading and the elastic stress $\sigma^{\alpha\beta}$ is decreased by the factor $(1 - d)$, or in more detail

$$\sigma_d^{\alpha\beta} = -\hat{p}\delta^{\alpha\beta} + (1 - d)S^{\alpha\beta}, \tag{23}$$

with

$$\hat{p} = \begin{cases} p & \text{for } p \geqslant 0 \\ (1 - d)p & \text{for } p < 0 \end{cases}. \tag{24}$$

The number of flaws per unit volume in a brittle body with failure strains that are lower than ε, is given by the Weibull distribution (Weibull, 1939)

$$n(\varepsilon) = k\varepsilon^m, \tag{25}$$

with the two Weibull parameters k and m. Typical values for basalt are $m = 16$, $k = 1 \times 10^{61}\,\mathrm{m}^{-3}$ (Nakamura et al., 2007).

As soon as a spherical volume $V = {}^4\!/_3\,\pi R_s^3$ undergoes a strain greater than its lowest activation threshold strain, damage starts to grow according to

$$\frac{\mathrm{d}}{\mathrm{d}t} d^{1/3} = \frac{c_\mathrm{g}}{R_s}, \tag{26}$$

with the constant crack growth velocity c_g. The crack growth velocity is chosen to be 40% of the velocity of a longitudinal elastic wave in the damaged material

$$c_\mathrm{g} = 0.4 \frac{1}{\varrho} \sqrt{K_0 + {}^4\!/_3 (1 - d)\mu}. \tag{27}$$

The fracture model requires the distribution of activation threshold strains (flaws) among the brittle body's SPH particles as a preprocessing step to the simulations.

We apply the algorithm introduced by Benz and Asphaug (1995) with the following approach: Flaws are distributed randomly among the brittle particles until each particle has been assigned with at least one activation threshold. We let N_flaw be the number of totally distributed activation flaws. For each

flaw j, $1 \leqslant j \leqslant N_{\text{flaw}}$, a particle a is chosen randomly and assigned with the activation threshold strain derived from the Weibull distribution $\varepsilon_{a,j}$,

$$\varepsilon_{a,j} = \left[\frac{j}{kV}\right]^{1/m}, \tag{28}$$

where V is the volume of the brittle material and k and m are the material dependent Weibull parameters. Using this procedure, each particle gets at least one flaw and on average a total of $N_{\text{flaws}} = N_p \ln N_p$ flaws are distributed. This number of flaws has to be stored in the memory during the simulation and is the main memory consumer in simulations with brittle materials.

During the simulation, once a particle undergoes a strain greater than its lowest activation threshold strain, damage grows according to a modified version of eq. (26)

$$\frac{\mathrm{d}}{\mathrm{d}t} d^{1/3} = n_{\text{act}} \frac{c_{\text{g}}}{R_s}, \tag{29}$$

where n_{act} is the number of activated flaws of the particle. This modification accounts for the accumulation of cracks.

Additionally, a damage limiter is included in the model. Since a particle has in general more than only one activation threshold, its behavior under tensile loading would be given by the flaw with the lowest activation threshold. To prevent this, the damage of the particle is limited by the ratio between the number of activated flaws n_{act} and the number of total flaws n_{tot} of this particle

$$d_{\max} = \frac{n_{\text{act}}}{n_{\text{tot}}}. \tag{30}$$

In other words, a particle can only be totally damaged if all of its flaws are activated.

In order to determine if a particle exceeds one of its threshold strains during the simulation, we need to derive the local scalar strain from the three-dimensional stress. Benz and Asphaug (1995) suggest computing the local scalar strain ε from the maximum tensile stress $\sigma_{\max} = \max[\sigma_1, \sigma_2, \sigma_3]$, where σ_γ are the principal stresses determined by a principal axis transformation of the stress tensor $\sigma^{\alpha\beta}$, which is already reduced by damage and yield strength

$$\varepsilon = \frac{\sigma_{\max}}{(1-d)E}. \tag{31}$$

The Young's modulus E of the intact material given by the bulk and shear moduli is written as

$$E = \frac{9K\mu}{3K+\mu}. \tag{32}$$

5. Self-gravity with Barnes-Hut Tree

We have implemented the efficient routines for Barnes-Hut tree builds and calculation of the gravitational forces for GPUs introduced by Burtscher (2011). The algorithm for the tree generation is explained at the end of this section.

Barnes and Hut (1986) were the first to use a hierarchical tree to calculate the gravitational forces between N bodies. Their approximation algorithm uses a tree-structured hierarchical subdivision of space into cubic cells, each of which is divided into 2^D subcells. The algorithm has the order $\mathcal{O}(N \log N)$ compared to the $\mathcal{O}(N^2)$ algorithm of the direct sum. The basic idea is that only nearby particles have to be treated individually during the calculation of the gravitational force on a specific particle. Distant nodes of the tree can be seen as one single particle sitting at the center of mass of the specific tree node and carrying the mass of all the particles that reside in this node. Although

different algorithms for the computation of self-gravitational forces exist, the hierarchical tree method is specially suited for the use in combination with SPH. As a side product of the computation of the gravitational forces, the hierarchical tree is constructed for each individual time step and hence, a geometrical hierarchy of the SPH particles (leaves of the tree) is available at all time of the computation. It can be used to determine the interaction partners for the SPH algorithm by performing a tree walk for each individual particle (Hernquist and Katz, 1989). Although the tree walks may take longer than the search through a helping grid, the interaction partner search may be faster because, in any case the tree build needs to be carried out in simulations with self-gravity.

We calculate the center of mass and total mass of every node in the tree to calculate the gravitational force exerted on the SPH particles. We call these objects pseudo-particles in the following. Now, for each SPH particle, we perform a tree walk over all nodes. With the use of the ϑ criterion, we identify whether a pseudo-particle sitting in the node may be used for the calculation of the gravitational force acting on our particle or whether we have to descend further down in the tree. The ϑ criterion is given by

$$\frac{s_{d_t}}{r_{pp}} < \vartheta, \tag{33}$$

where s_{d_t} denotes the edge length of the tree node containing the pseudo-particle and r_{pp} is the distance between the SPH particle and the pseudo-particle. The gravitational force from the pseudo-particle acting on the particle a is then given by

$$\mathbf{F}_a = \frac{G m_a m_p}{r_{pp}^3} \left(\mathbf{x}_a - \mathbf{x}_p \right), \tag{34}$$

where $\mathbf{x}_p$ and m_p denote the location and the mass of the pseudo-particle, respectively. To avoid numerical instabilities, the distance between the pseudo-particle and the particle is smoothed.

Barnes and Hut (1989) present a detailed error analysis for their tree algorithm. They find consistent accuracies of the algorithm for $\vartheta = 0.3$ with the consideration of the monopoles of the tree compared to $\vartheta = 0.7$ and the additional calculation of the quadrupoles for each tree node. The error scales with ϑ^2 and $\sqrt{N_p}$. For the sake of memory saving, we calculate only the monopoles of the tree and use $\vartheta \leqslant 0.5$ in our simulations including self-gravity instead of the usual value of 0.8.

We have implemented the Barnes-Hut tree strictly following the efficient implementation for N-body calculations with CUDA by Burtscher (2011). The hierarchical decomposition is recorded in an octree. Since CUDA does not allow the use of pointers, which is convenient for the handling of trees, e.g., allows for recursive programming of tree walks, we use one large index array for all SPH particles and all tree nodes, which we call `childList`. This memory is allocated dynamically at the start of the simulation run. The maximum number of tree nodes is given by $\mathrm{ceil}[2.5 \times N_p]$, where N_p is the number of SPH particles in the simulation. For each tree node there are N_{children} entries in `childList`, where N_{children} is simply given by the dimension

$$N_{\mathrm{children}} = 2^D. \tag{35}$$

The tree is constructed in a way that the N_p particles reside in N_p leaves in the end. At first, the root of the tree is constructed from the geometry of the particle distribution. The computational domain (in three dimensions) is given by the cube, which encloses all SPH particles. This cube has the edge length

$$s_0 = \max_{\alpha \in D} \left[\max_{a,b \in N_p} \left[x_a^\alpha - x_b^\alpha \right] \right] \tag{36}$$

and defines the edge length of the root node. The edge length of a tree node of depth d_t is given by

$$s_{d_t} = \left[\frac{1}{2} \right]^{d_t} s_0. \tag{37}$$

Starting from the root node, the particles are inserted into the tree in the following way. In the beginning, all but one (the root node) entry in `childList` are -1, corresponding to the status "no kid". All particles are distributed among the threads on the GPU. Now N_{threads} threads on the GPU start to compare the locations of their SPH particles with the position of the root node to determine the childindex. Since there are many more threads than children (especially in the beginning of the algorithm), we need to avoid race conditions actively. To accomplish this, we add the status -2 or "locked" to the `childList` and use the CUDAfunction "atomic compare and save" `atomicCAS` to lock an entry in the `childList`. This functions checks first if the specific index is already locked. If the index in `childList` is already locked, the thread waits for the other threads to finish and restarts its attempt. If the index is not locked, the threads read the index value, which may be -1 in the beginning or a different node index, and saves it. Now, the second round starts and the children of the root node are already filled with SPH particles that were sorted there. Since leaves or particles cannot have children by definition, new inner tree nodes have to be generated. A new inner tree node has half of the edge length from its parent. New inner tree nodes are added until both particles, the new particle and the particle in the parent node, can be sorted into two different leaves. As soon as both particles reside in different leaves, the thread writes and cancels the lock.

When we want to calculate the gravitational forces between the SPH particles, we have to calculate the positions and locations of the pseudo-particles in the tree. We use the algorithm presented by Burtscher (2011) in the following manner to determine the pseudo-particle in a specific node: For each inner node, we start in parallel for N_{threads} a CUDA kernel that picks the last N_{threads} added inner nodes, for which we calculate the center of mass and total mass at first. If the inner nodes are not leaves, the thread looking at these nodes and has to wait for all other threads to end beforce continuing because it needs the location and mass of the pseudo-particles in these nodes. This is again done by the use of -1 as the initial mass of a pseudo-particle. When the thread finishes the calculation for its node, it moves up one tree depth and continues with the next node. The implementation of Burtscher is especially designed for CUDA, regarding memory access and parallelization.

6. SPH representation of the equations

6.1 Settings in `parameter.h`

The code allows to choose between two different SPH representations of the conservation of momentum and internal energy equation. In the following for `SOLID==1`.

6.1.1 `SPHEQUATIONS == SPH_VERSION1`

The acceleration for particle a is given by

$$\frac{\mathrm{d}v_a^\alpha}{\mathrm{d}t} = \sum_b m_b \left(\frac{\sigma_a^{\alpha\beta}}{\varrho_a^2} + \frac{\sigma_b^{\alpha\beta}}{\varrho_b^2} \right) \frac{\partial W_{ab}}{\partial x_a^\beta}. \tag{38}$$

6.1.2 `SPHEQUATIONS == SPH_VERSION2`

The acceleration for particle a is given by

$$\frac{\mathrm{d}v_a^\alpha}{\mathrm{d}t} = \sum_b m_b \left(\frac{\sigma_a^{\alpha\beta} + \sigma_b^{\alpha\beta}}{\varrho_a \varrho_b} \right) \frac{\partial W_{ab}}{\partial x_a^\beta}. \tag{39}$$

7. Kernel Function

Several kernel functions are available via command line parameter `-k/--kernel`. Information about which kernel is implemented is provided by the standard `miluphCUDA` help option `-h/--help`. In the following sections, h denotes the smoothing length, q is the ratio between distance and smoothing length, $q = \frac{x_{ab}}{h}$, where $x_{ab} = |\mathbf{x}_a - \mathbf{x}_b|$ is the distance between particle a and b. Please note, that in contrast to the original definition of the smoothing length, the interaction radius in `miluphCUDA` is given by h, not $2h$.

7.1 variable and fixed smoothing length

The code supports variable and fixed smoothing lengths. Please set `VARIABLE_SML` to 1 in `parameter.h` if you want a variable smoothing length. You can additionally choose between a fixed number of interaction partners by setting `FIXED_NOI` to 1 or the code integrates the time evolution of the smoothing length via

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \frac{h}{d} \nabla \cdot \mathbf{v}, \tag{40}$$

where d denotes the dimension if you set `INTEGRATE_SML`. Additionally, please set two factors for the minium and maximum allowed smoothing length in `material.cfg`, namely `factor_sml_min` and `factor_sml_max`. The minimum (maximum) allowed smoothing length is then given by `factor_sml_min * sml` (`factor_sml_max * sml`).

7.2 spline kernels

7.2.1 cubic spline

Standard cubic spline kernel by Monaghan and Lattanzio (1985), chosen by command line option `-k cubic_spline`.

$$W(q) = \frac{\sigma}{h^d} \begin{cases} (6q^3 - 6q^2 + 1) & \text{for} \quad 0 \leqslant q < \frac{1}{2} \\ 2(1-q)^3 & \text{for} \quad \frac{1}{2} \leqslant q \leqslant 1 \\ 0 & \text{for} \quad q > 1. \end{cases} \tag{41}$$

where d denotes the dimension, and σ is the normalization constant $\frac{4}{3}$, $\frac{40}{7\pi}$ and $\frac{8}{\pi}$ for 1D, 2D, and 3D, respectively.

7.2.2 quartic spline

Quartic spline kernel (Dehnen and Aly, 2012), chosen by command line option `-k quartic_spline`.

$$W(q) = \frac{\sigma}{h^d} \left((1-q)_+^4 - 5\left(\frac{3}{5} - q\right)_+^4 + 10\left(\frac{1}{5} - q\right)_+^4 \right) \qquad \text{for} \quad 0 \leqslant q < 1 \tag{42}$$

where d denotes the dimension, and σ is the normalization constant $\frac{5^5}{768}$, $\frac{3 \cdot 5^6}{2398\pi}$ and $\frac{5^6}{512\pi}$ for 1D, 2D, and 3D, respectively.

7.3 Wendland kernels

7.3.1 Wendland C2

Wendland C2 kernel (Dehnen and Aly, 2012), chosen by command line option `-k wendlandc2`.

$$W(q) = \frac{\sigma}{h^d} \begin{cases} (1-q)_+^4(1+4q) & \text{for 2D and 3D and} \quad 0 \leqslant q < 1 \\ (1-q)_+^3(1+3q) & \text{for 1D and} \quad 0 \leqslant q < 1 \\ 0 & \text{for} \quad q >= 1. \end{cases} \tag{43}$$

where d denotes the dimension, and σ is the normalization constant $\frac{5}{4}$, $\frac{7}{\pi}$ and $\frac{21}{2\pi}$ for 1D, 2D, and 3D, respectively[1].

7.3.2 Wendland C4

Wendland C4 kernel (Dehnen and Aly, 2012), chosen by command line option `-k wendlandc4`.

$$W(q) = \frac{\sigma}{h^d} \begin{cases} (1-q)_+^6(1+6q+\frac{35}{3}q^2) & \text{for 2D and 3D and} \quad 0 \leqslant q < 1 \\ (1-q)_+^5(1+5q+8q^2) & \text{for 1D and} \quad 0 \leqslant q < 1 \\ 0 & \text{for} \quad q >= 1. \end{cases} \tag{44}$$

where d denotes the dimension, and σ is the normalization constant $\frac{3}{2}$, $\frac{9}{\pi}$ and $\frac{495}{32\pi}$ for 1D, 2D, and 3D, respectively.

7.3.3 Wendland C6

Wendland C6 kernel (Dehnen and Aly, 2012), chosen by command line option `-k wendlandc6`.

$$W(q) = \frac{\sigma}{h^d} \begin{cases} (1-q)_+^8(1+8q+25q^2+32q^3) & \text{for 2D and 3D and} \quad 0 \leqslant q < 1 \\ (1-q)_+^7(1+7q+19q^2+21q^3) & \text{for 1D and} \quad 0 \leqslant q < 1 \\ 0 & \text{for} \quad q >= 1. \end{cases} \tag{45}$$

where d denotes the dimension, and σ is the normalization constant $\frac{55}{32}$, $\frac{78}{7\pi}$ and $\frac{1365}{64\pi}$ for 1D, 2D, and 3D, respectively.

7.4 spiky kernels

7.4.1 Desbrun & Cani spiky kernel

$$W(q) = \frac{\sigma}{h^d} \begin{cases} (1-q)^3 & \text{for} \quad 0 \leqslant q < 1 \\ 0 & \text{for} \quad q >= 1. \end{cases} \tag{46}$$

where d denotes the dimension, and σ is the normalization constant $\frac{10}{\pi}$ and $\frac{15}{\pi}$ for 2D and 3D, respectively.

[1] $(\cdot)_+ \equiv \max\{0, \cdot\}$

References

J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, Dec. 1986. doi: 10.1038/324446a0.

J. E. Barnes and P. Hut. Error analysis of a tree code. *ApJS*, 70:389–417, June 1989. doi: 10.1086/191343.

W. Benz and E. Asphaug. Simulations of brittle solids using smooth particle hydrodynamics. *Computer Physics Communications*, 87:253, 1995.

M. Burtscher. *GPU Computing Gems Emerald Edition*. 2011.

W. Dehnen and H. Aly. Improving convergence in smoothed particle hydrodynamics simulations without pairing instability. *MNRAS*, 425:1068–1082, Sept. 2012. doi: 10.1111/j.1365-2966.2012.21439.x.

D. E. Grady and M. E. Kipp. Continuum modelling of explosive fracture in oil shale. *Int. J. Rock Mech. Min. Sci. Geomech. Abstr.*, 17:147, 1980.

L. Hernquist and N. Katz. Treesph: A unification of sph with the hierarchical tree method. *ApJS*, 70:419, 1989.

H. Melosh. *Impact Cratering: A Geologic Process*. Oxford monographs on geology and geophysics. Oxford University Press, 1996. ISBN 9780195104639.

J. J. Monaghan and J. C. Lattanzio. A refined particle method for astrophysical problems. *A&A*, 149:135, Aug. 1985.

A. M. Nakamura, P. Michel, and M. Setoh. Weibull parameters of Yakuno basalt targets used in documented high-velocity impact experiments. *Journal of Geophysical Research (Planets)*, 112:E02001, Feb. 2007. doi: 10.1029/2006JE002757.

J. H. Tillotson. Metallic equations of state for hypervelocity impact. Technical Report General Atomic Report GA-3216, General Dynamics, San Diego, CA, 1962.

R. von Mises. Mechanik der festen Körper im plastisch deformablen Zustand. *Göttin. Nachr. Math. Phys.*, 1: 582–592, 1913.

W. A. Weibull. A statistical theory of the strength of materials [translated]. *Ingvetensk. Akad. Handl.*, 151:5, 1939.