Sphere SPH user manual

Johannes Willkomm *johannes.willkomm@rwth-aachen.de*

July 26, 2010

Contents

1 Introduction

Sphere SPH is a Spmmothed particle hydrodynamics simulation suite. There is also an AD [GJU96] differentiated version.

2 Quick start guide

2.1 Compiling & installing

2.2 Running

2.3 Viewing Results

3 The SPH simulation program

3.1 Compiling & installing

In the trunk directory type make install. This installs the driver script sphere to $PREFIX/bin and the other scripts and binaries to $PREFIX/sph-rXXX/bin, where XXX is the output of svnversion src. Thus it is possible to have different versions of Sphere installed at the same time. The driver script will automatically run the current version, but this can be overridden with the -R option.

The commands shown in this manual should be run from the trunk directory, unless stated otherwise. There has been some attempt to make the scripts more general, but this requires to set the environment variable SPH_ROOT to point to the trunk directory.

3.1.1 Selecting Features at Compile-Time

The following features can be selected at compile time

- SPH_RENDER – online rendering with OpenGL
- _OPENMP – parallelization using OpenMP
- SPH_AD – automatic differentiation in forward mode

These features should be set or unset by setting some Makefile variables:

- SPH_RENDER – RENDER=yes*/no
- _OPENMP – OPENMP=yes*/no
- SPH_AD – AD=yes/no*

For example, on the Solaris system, where online rendering does not work, the feature can be removed by using make install RENDER=no. Note that the AD feature compiles additional binaries, i.e. the non-AD binaries will always be present.

Some more compile time features can be selected by editing the file src/common.hh. These include

- SPH_PERIODIC_BOUNDARIES – support for periodic boundary conditions
- SPH_USE_INTEGRATOR – use integrators, otherwise simple explicit Euler only
- SPH_PARTICLE_HAS_ID – let particles have IDs
- SPH_PARTICLE_VAR_HAS_NEIGHBOUR_SUM – compute (and save) number of neighbours of particles (only works correctly with explicit Euler time integration)

3.1.2 Requirements

Sphere uses the VTK format to load and save data, and the H5Part format, which is a subset of the HDF5 format to save data. Sphere uses the H5Part library and that in turn uses the HDF5 library. Sphere also uses Freeglut and OpenGL to render the data online. Therefor the following libraries are required to compile Sphere:

- VTK library – any recent version should do
- HDF5 library – either 1.8.X or 1.6.X since we can use the backward compatbility mode (-DH5_USE_16_API)
- H5Part library – see `http://vis.lbl.gov/Research/AcceleratorSAPP/`
- OpenGL library – only if feature SPH_RENDER is enabled
- Freeglut library – only if feature SPH_RENDER is enabled

The libraries must also be present in the library search path of the system. The script sh/set-ld-library-path.sh can be used to set the LD_LIBRARY_PATH accordingly. The H5Part library is static.

3.1.3 Building the test cases

To build the test cases (VTK files that contain initial particle configurations), cd to the vtk directory and type make. Note that some test cases require a some time to compile, although the scene compiler is parallelized. The test cases will be build in the vtk/test-cases2 subdirectory.

Building the test cases requires requires Python and Octave, since the boundary particles are mostly generated using scripts in one of these languages. As a rule of thump, 2D tests require Python while the 3D tests require Octave.

3.2 Running

3.2.1 The sphere driver script

3.2.2 Runtime parameters

The Sphere binaries take user all user parameters via environment variables, which start with the string SPH_PARAM_. This prefix will usually be ommited in the documentation here. Some parameters depend on the specific test case and should not be changed by the user. These are placed in text files with suffix .env in the form of shell export statements. The

sphere driver script will first try to source (i.e. execute) the shell script name.vtk.env if the users specifies the VTK file name.vtk as the simulation input. Then the options given to the sphere script are processed, thus it is possible to override the per-testcase settings, e.g. turn gravity on/off or run with a different value for H than recommended in the .env file.

User parameters should be given as options to the sphere driver which will export the correponding environment variables for the Sphere binaries. Some options are not modelled as command line options, these can only be set or changed using environment variables. Examples are the parameters DOWN which specifies the direction of gravity or GAMMA which is the exponent γ in the equation of state (5). To change one of these either export the corresponding variable in your shell:

```
$ export SPH_PARAM_DOWN="-0.707 -0.707 0"
$ sphere -3 sph-initial.vtk
```

or prepend the variable definition to the command:

```
$ SPH_PARAM_GAMMA=3 sphere -3 sph-initial.vtk
```

In the second case the variable applies to the single command only.

In Table 1 and Table 2 all parameters are listed with their name without the prefix, a short description, the corresponding sphere command line option and a range of possible values.

Table 1: User parameters controlling the numerical aspects of a Sphere SPH simulation

Name	Opt.	Sym.	Default	Val.	Description
RHO0	–	$\rho^{(0)}$	$1000\,\frac{\mathrm{kg}}{\mathrm{m^d}}$	$\mathbb{R}_+$	reference density of the medium (water) (5)
B	-B	B	$5 \cdot 10^5$	$\mathbb{R}_+$	stiffness constant (5)
GAMMA	–	γ	7	$\mathbb{N}$	exponent in (5)
QUADER_P2	–	P_2	$(1, 1, 1)\,\mathrm{m}$	$\mathbb{R}_+^d$	right, upper, back corner of Ω
H	-H	H	$0.01\,\mathrm{m}$	$\mathbb{R}_+$	radius of kernel support
KERNEL	-k	–	wendland2	$\mathcal{Z}^*$	name of kernel Section 3.3.2
SUMMATION	-s	–	symmetric	$\mathcal{Z}^*$	summation algorithm Section 3.3.4
SORTED_INDEX	–	–	csort	$\mathcal{Z}^*$	sort algorithm Section 3.3.4
M	-m	m	6	$\mathbb{N}$	hash function parameter
TMAX	-T	t_f	$3\,\mathrm{s}$	$\mathbb{R}_+$	simulation time span
DT	-d	δt	$1 \cdot 10^{-4}\,\mathrm{s}$	$\mathbb{R}_+$	time step size
NT	-n	n_t	$\lceil t_f/\delta t \rceil$	$\mathbb{N}$	number of time steps
G	-g	g	$9.81\,\frac{\mathrm{m}}{\mathrm{s^2}}$	$\mathbb{R}_+$	gravity acceleration
DOWN	–	–	$(0, -1, 0)$	$\mathbb{R}_+^d$	unit direction of gravity acceleration
SPEED_OF_SOUND	–	c_s	$1484\,\frac{\mathrm{m}}{\mathrm{s}}$	$\mathbb{R}_+$	viscosity (1)
MOVING_PARTICLE	-p	–	ferrari	$\mathcal{Z}^*$	fluid particle type Section 3.3.1
BOUNDARY_PARTICLE	-b	–	point_symmetric	$\mathcal{Z}^*$	boundary particle type Section 3.3.1
MU	-v	μ	0	$\mathbb{R}_+$	dynamic bulk viscosity (6)
ALPHA	-v	α	0.01	$\mathbb{R}_+$	Monaghan viscosity (1)
BETA	-w	β	0	$\mathbb{R}_+$	Monaghan viscosity (1)
ETA	-e	η	0	$\mathbb{R}_+$	Monaghan viscosity (1)
EPSILON	–	ε	0	$\mathbb{R}_+$	Monaghan XSPH (4)
LJ_R0	–	r_0	$H\,\mathrm{m}$	$\mathbb{R}_+$	Lennard–Jones radius (10)
LJ_D	–	D	100	$\mathbb{R}_+$	Lennard–Jones strength (10)
LJ_P2	–	δ_2	2	$\mathbb{N}$	second exponent (10)
SENSORS	–	–	0	$\mathbb{N}$	number of sensors
SENSOR_TYPE	–	–	velocity	$\mathcal{Z}^*$	default sensor type
INTEGRATOR	-i	–	pk1_1	$\mathcal{Z}^*$	time integration algorithm Section 3.3.3
SPH_AD	-a, -C	–	no	–	start -ad or -cv binary
SPH_NDIM	-1, -2, -3	d	-2	–	number of dimensions

Table 2: User parameters controlling the I/O and interface aspects of Sphere

Name	Opt.	Sym.	Default	Val.	Description
THREADS	-t	n_p	4	$\mathbb{N}$	number of OpenMP threads
CSORT_THREADS	–	–	n_p	$\mathbb{N}$	number of OpenMP threads used in the sorting operations **??**
MAX_THREADS	–	–	256	$\mathbb{N}$	max. OpenMP threads in interactive mode
MIN_LEFT	–	–	1	$\mathbb{N}$	particles left before abort
RENDER_EVERY	–	n_r	10	$\mathbb{N}$	render every n_r-th timestep
SLEEP_EVERY	–	n_S	0	$\mathbb{N}$	sleep every n_S-th timestep
SLEEP_TIME	–	t_S	1 s	$\mathbb{R}_+$	sleep for t_S
TIMINGS	-z	–	0	$\{0,1\}$	print more timing information
ABORT_ON_FE	–	–	1	$\{0,1\}$	abort after a FP exception
TRAP_FE	–	–	0	$\{0,1\}$	on FP exception raise SIGFPE
INITIAL_FILE	arg	`infile`	sph-initial.vtk	$\mathcal{Z}^*$	input file name
RESULT_FILE	-o	`outfile`	sph-result-*dd*	$\mathcal{Z}^*$	prefix for output file name
LOG_FILE	–	–	`outfile`.sphrun.xml	$\mathcal{Z}^*$	input file name
SAVE_FIELDS	–	–	–	$\mathcal{Z}^*$	names of fields to save
LOAD_FIELDS	–	–	–	$\mathcal{Z}^*$	names of fields to load from `infile`
SAVE_RATE	-S	f_s	25 Hz	$\mathbb{R}_+$	rate to save results
SAVE_EVERY	–	n_s	$\frac{1}{f_\mathrm{s}\delta t}$	$\mathbb{N}$	save every n_s-th timestep
SAVE_OUT	–	–	1	$\{0,1\}$	save particles which left Ω
SAVE_BOUNDARY	–	–	1	$\{0,1\}$	save boundary particles in first step

3.2.3 Test cases

3.2.4 Save file format

The Sphere simulation programs can load the VTK file format and use both the H5Part and the VTK file format to save results. VTK files can be viewed with Paraview. Simulation sequences are save as series of files with numbered names. Paraview detects such file series and displays them as trees similar to a directory. One can both load the entire series and individual files. The advantage of the H5Part file format is that results from all time steps are saved in a single file.

H5Part file format The H5Part file format can be viewed directly with the Paraview modification Paraview-Meshless. Since H5Part is a subset of HDF5, the HDF5 utilities can also be used. Especially interesting are the commands h5diff, h5ls and h5dump. In Octave HDF5 files can be imported by simply using load(). In the octave subdirectory there is also a script hdf5load which implements the same functionality for use in MATLAB. However this script will probably only work with the H5Part format and not with arbitrary HDF5 files. The script sh/h52vtk.sh can batch convert H5Part files into series of VTK files. While in the .hdf file the first time step contains both boundary and fluid particles, there is a series of VTK files containing only the fluid particles, one for each time step and the boundary particles are saved once into the file boundary.vtk.

VTK file format The VTK file format can be only viewed with the original Paraview. Sphere can also save simulation results in the VTK format directly, using the parameter RESULT_FORMAT.

3.2.5 Utility programs and scripts

h52vtk.sh Convert H5Part files to VTK file series for viewing with paraview. Several H5Part files can be converted at once. The option -o specifies a directory for output. A subdirectory with the stem of the filename will be created for each input file. On the cluster it may be advantaguous to place the output on the temporary file system. For example, sh/h52vtk.sh −o $TMP/vtkout ∗.h5 converts all H5Part files to VTK series in the $TMP/vtkout directory.

The main conversion work is done by a MATLAB/Octave script. By default Octave is used and with option -m MATLAB is used.

vtkdump Dump VTK files, one line per time step. The first line starts with a # and lists the names of the fields.

buildall.sh A script to build sphere on several machines with different compilers. This script is custom written for the RZ cluster. It can serve as a reference how to build sphere on a given OS with a given comiler.

scale-test.sh Perform a scale test. Simply prepend this to the call to sphere. Make sure to set a small number of time steps. For example,

```
sh/scale-test.sh -o test -T "1 2 4 8 16" sphere -n 10 -2 -d 1e-5 \
  vtk/test-cases2/2D/falling-film/gross/test.vtk
```

runs a scale test with 1, 2, 4, 8 and 16 threads and produces the files

- test-raw.txt – contains the raw times in seconds
- test-speedup.txt – contains the speedup results
- test-efficiency.txt – contains the efficiency results

The files are cumulative, i.e. test-efficiency.txt contains all the results. When column n hold the efficiency result for certain timing, then column $n+1$ contains the corresponding speedup and column $n+2$ has the underlying time result in seconds. The first column is always the number of threads.

vtk2hdf5 Convert a VTK file or a series of VTK files into a H5Part file. This is mainly usefull to convert results from other source to H5Part for comparison with h5diff. The input file name can contain a printf pattern that will be used to determine the filenames from a range of integers.

run-sph.sh Synonym for the sphere driver script.

3.3 Special features

3.3.1 Particle types

Fluid particles Sphere implements two kinds of fluid particles, one are the the classical SPH fluid particles as given by Monaghan, the other implement the new SPH equations given by Ferrari et.al.

Monaghan particles These particles are selected with the option -p monaghan of the driver script, or the parameter MOVING_PARTICLE. The Monaghan particles implement the SPH equation as given in [Mon94]. The change in velocity (force, acceleration) is given by

$$\frac{\mathrm{d}\,\mathbf{v}_i}{\mathrm{d}\,t} = \sum_{j\in\mathcal{N}(i)} -m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ij}\right) \nabla_i W_{ij} + \mathbf{f}_i, \tag{1}$$

where $\mathbf{f}_i$ is the body force (gravity), $\nabla_i W_{ij}$ is the gradient of the kernel function wrt. the position of particle i. The so-called artificial Monaghan-type viscosity Π_{ij} is given by

$$\Pi_{ij} = \begin{cases} 2\,\dfrac{\beta\mu_{ij}^2 - \alpha c_\mathrm{s}\mu_{ij}}{\rho_i + \rho_j}, & (\mathbf{v}_i - \mathbf{v}_j)^T(\mathbf{x}_i - \mathbf{x}_j) < 0, \\ 0, & (\mathbf{v}_i - \mathbf{v}_j)^T(\mathbf{x}_i - \mathbf{x}_j) > 0, \end{cases}$$

where α and β are user parameters and c_s is the speed of sound. Finally,

$$\mu_{ij} := \frac{h(\mathbf{v}_i - \mathbf{v}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) + \eta^2},$$

with η again a user parameter. Usually we set $\alpha = 0.01$, $\beta = 0$ and $\eta = 0$.

The differential equation for the density is given by

$$\frac{\mathrm{d}\,\rho_i}{\mathrm{d}\,t} = \sum_{j \in \mathcal{N}(i)} m_j \left(\mathbf{v}_i - \mathbf{v}_j\right)^T \nabla_i W_{ij}. \tag{3}$$

The heat energy is given by

$$\frac{\mathrm{d}\,u_i}{\mathrm{d}\,t} = \frac{1}{2} \sum_{j \in \mathcal{N}(i)} m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ij}\right) \left(\mathbf{v}_i - \mathbf{v}_j\right)^T \nabla_i W_{ij}.$$

The position is given by

$$\frac{\mathrm{d}\,\mathbf{x}_i}{\mathrm{d}\,t} = \mathbf{v}_i + 2\varepsilon \sum_{j \in \mathcal{N}(i)} m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_i + \rho_j} W_{ij}, \tag{4}$$

where ε is a user parameter. Setting $\varepsilon \neq 0$ yields the so called XSPH method.

The pressure is given by the equation of state, using the so called Taits equation here:

$$P_i = B \left[\left(\frac{\rho_i}{\rho^{(0)}} \right)^{\gamma} - 1 \right], \tag{5}$$

where B and γ are constants and $\rho^{(0)}$ is the reference density of the medium. The parameter B controls the stiffness of the medium, while $\gamma = 7$ and $\rho^{(0)} = 1000$ for the medium water.

Ferrari particles In the work by Ferrari et. al. [FDTA09] slightly different equations are given for the SPH methods. The artificial viscosity term Π_{ij} is removed from (1) and instead (3) is modified by an additional term. No equation is given for the heat energy. Viscosity is then reintroduced into the velocity equation, but it is entirely optional in this case while in the Monaghan equations the artificial viscosity is required for the stability.

$$\frac{\mathrm{d}\,\mathbf{v}_i}{\mathrm{d}\,t} = \sum_{j \in \mathcal{N}(i)} -m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right) \nabla_i W_{ij} + \mathbf{f}_i. \tag{6}$$

The differential equation for the density is given by

$$\frac{\mathrm{d}\,\rho_i}{\mathrm{d}\,t} = - \sum_{j \in \mathcal{N}(i)} m_j \left[\left(\mathbf{v}_j - \mathbf{v}_i\right)^T \nabla_i W_{ij} - \mathbf{n}_{ij}^T \nabla_i W_{ij} \left(\frac{c_{ij}}{\rho_j} (\rho_j - \rho_i) \right) \right], \tag{7}$$

where c_{ij} is the celerity

$$c_{ij} = \max(c_i, c_j), \quad c_i = \sqrt{\gamma \frac{B}{\rho^{(0)}} \left(\frac{\rho_i}{\rho^{(0)}} \right)^{\gamma - 1}},$$

and the vector $\mathbf{n}_{ij} = \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|}$ is the unitary vector from particle i to j. Note that this new formula is not symmetric in i and j. Also note that this new formula leads to an egalization of densities even if particles do not move. The position is given by

$$\frac{\mathrm{d}\,\mathbf{x}_i}{\mathrm{d}\,t} = \mathbf{v}_i.$$

These equations already give a SPH formulation which is stable enough for water simulation. However a viscosity term is provided which can be turned on optionally:

$$\frac{\mathrm{d}\,\mathbf{v}_i}{\mathrm{d}\,t} = -\sum_{j\in\mathcal{N}(i)} F_{ij}^{\mathrm{I}} - \sum_{j\in\mathcal{N}(i)} F_{ij}^{\mathrm{V}} + \mathbf{f}_i, \tag{8}$$

where F_{ij}^{I} and F_{ij}^{V} are the inviscid and viscous components:

$$\begin{aligned}
F_{ij}^{\mathrm{I}} &= \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right)\nabla_i W_{ij},\\
F_{ij}^{\mathrm{I}} &= \frac{m_j\mu}{\rho_i\rho_j}\left(\frac{7}{3}\left(\mathbf{v}_j - \mathbf{v}_i\right) + \frac{5}{3}\mathbf{n}_{ij}\mathbf{n}_{ij}^T\left(\mathbf{v}_j - \mathbf{v}_i\right)\right)\Theta_{ij}, \text{ and}\\
\Theta_{ij} &= -\frac{\mathbf{n}_{ij}^T}{|\mathbf{x}_j - \mathbf{x}_i|}\nabla_i W_{ij}.
\end{aligned}$$

Now μ is the user parameter that controls viscosity, usually $\mu = 0$.

Boundary particles

Lennard–Jones particles The so called Lennard–Jones boundary particles as given in [Mon94] simply exert a repellant force on a fluid particle i whose distance from a boundary particle j $x_{ij} = |\mathbf{x}_{ij}| = |\mathbf{x}_j - \mathbf{x}_i|$ is smaller than r_0.

$$LJ(x_{ij}) = \begin{cases} -D\left(\left(\frac{r_0}{x_{ij}}\right)^{\delta_1} - \left(\frac{r_0}{x_{ij}}\right)^{\delta_2}\right)\cdot\frac{\mathbf{x}_{ij}}{x_{ij}^2}, & 0 \le x_{ij} < r_0,\\ 0, & \text{else.}\end{cases} \tag{10a}$$

Here r_0 and D are user parameters where we use $r_0 = H$ and $D = 100$. For the exponents δ_1 and δ_2 we fix $\delta_1 = 2\delta_2$ and use $\delta_2 = 2$. One can use either Monaghan or Ferrari particles with these boundary particles.

Point symmetric ghost particles In [FDTA09] the boundaries are treated by point symmetrically creating ghost particles. Let the i-th fluid particle approach the boundary particle p_{B}, then the fictuous partikle p_k is created with the following properties:

$$\mathbf{x}_k = 2\mathbf{x}_{\mathrm{B}} - \mathbf{x}_i, \quad \mathbf{v}_k = 2\mathbf{v}_{\mathrm{B}} - \mathbf{v}_i, \quad \rho_k = \rho_i, \quad m_k = m_i,$$

where $\mathbf{x}_{\mathrm{B}}$ and $\mathbf{v}_{\mathrm{B}}$ are the position and velocity of the boundary particle. Then particle p_i is related with the fictous particle p_k as usual. One can use either Monaghan or Ferrari particles with these boundary particles.

3.3.2 Kernels

Sphere implements several kernel functions, cf. Table 3 for a list. The desired kernel is selected by the option -k or the parameter KERNEL. All of them are radial functions. Different from the SPH literature we always use the radius H when talking about the kernel "from the outside". Thus for all kernels W it holds that $W(r) = 0$, $r \ge H$. In the smoothing function

formulas in the Table 3 we give the mapping from the kernel radius H to the kernel parameter h as known from the SPH literature.

The value of H is fixed globally and set by the sphere option -H or parameter H. Note however that this parameter has to be inside a very limited range for good results. Following [FDTA09], considering a spline kernel with $H = 2h$ we assume the optimal smoothing length H_i for a given particle to be

$$h_i = 2\sqrt[d]{\frac{m_i}{\rho_i}}, \tag{11}$$

that is the side length of a cube in d dimensions. However, in Sphere variable smoothing length have not yet been implemented. Thus the scene compiler sets the global value $H = 2\delta x$, where δx is the fluid particle spacing, and saves this value in the .env file of the test case.

Note that the Gaussian kernel is simply cut off at $H = 3h$. The table also lists the proportionality constants α_{1D}, α_{2D} and α_{3D} such that the integral over the whole domain $\int_\Omega W(|\vec{x}|)\mathrm{d}\vec{x} = 1$ in 1D, 2D and 3D, resp. Information on the Wendland kernels can be found in [Wen95], the other kernels are described in [LL03]. The proportionality constants can also be found in the literature, except those of the Wendland functions and those of the morris4 kernel for 2D and 3D. Note that I have different results from [LL03] for the morris5 kernel in 1D and 3D.

3.3.3 Time integration algorithms

Sphere features several different time integration algorithms, all which are explicit. The sphere option -i selects the desired time integration, cf. Table 4 for a list. Note that the Adams-Moulton implicit multistep method is implemented and referred to in the table as amm1, amm2, ... amm7 but it can not be used directly since it requires an implicit solution. Thus it is only used indirectly by the predictor-corrector schemes which feed the solution of the Adams-Bashforth methods into the Adams-Moulton method. Note that a breakdown of the runtime spend in each function evaluation can be obtained using the option -z when running sphere.

3.3.4 Summation algorithms

Sphere implements two parallel algorithms to sum the differential updates at a current state:

- naive – take each particle in turn, find neighbours and sum contributions
- symmetric – walk over the world and update both particles update value for each evaluation of particle-particle forces
- sym-mb – same as symmetric but uses 2^d times more memory for particle updates instead of 2^d barriers in the algorithm

The summation algorithm is selected by the option -s of the sphere driver script. Symmetric is usually about twice as fast as naive, but the parameter -m must be set correctly. For small (2D) test cases -m 4 or -m 5 is apropriate, while for the larger test cases -m 6 (the default) works best.

Table 3: Smoothing functions (kernels) available in Sphere.

Name	d	$W(R) := W(r/h)$	H	$\alpha_{1\mathrm{D}}$	$\alpha_{2\mathrm{D}}$	$\alpha_{3\mathrm{D}}$
gauss		$\alpha e^{-R^2}, \quad 0 \le R \le 3$	$3h$	$\frac{1}{h\sqrt{\pi}}$	$\frac{1}{h^2\pi}$	$\frac{1}{h^3\pi\sqrt{\pi}}$
lucy		$\alpha\,(1.0-R)_+^3\,(3R+1)$	h	$\frac{5}{4h}$	$\frac{5}{\pi h^2}$	$\frac{105}{16\pi h^3}$
spline		$\alpha \begin{cases} \left(\frac{2}{3} - R^2 + \frac{1}{2}R^3\right), & 0 \le R < 1, \\ \frac{1}{6}\,(2-R)^3, & 1 \le R \le 2, \\ 0, & \text{else} \end{cases}$	$2h$	$\frac{1}{h}$	$\frac{15}{7\pi h^2}$	$\frac{3}{2\pi h^3}$
morris4		$\alpha \begin{cases} (R+2.5)^4 - 5(R+1.5)^4 + 10(R+0.5)^4, \\ \quad \text{when } 0 \le R < 0.5, \\ (2.5-R)^4 - 5(1.5-R)^4, \\ \quad \text{when } 0.5 \le R < 1.5, \\ (2.5-R)^4, \quad 1.5 \le R < 2.5, \\ 0, \quad \text{else} \end{cases}$	$2.5h$	$\frac{1}{24h}$	$\frac{96}{1199\pi h^2}$	$\frac{1}{20\pi h^3}$
morris5		$\alpha \begin{cases} (3-R)^5 - 6(2-R)^5 + 15(1-R)^5, \\ \quad \text{when } 0 \le R < 1, \\ (3-R)^5 - 6(2-R)^5, & 1 \le R < 2, \\ (3-R)^5, & 2 \le R < 3, \\ 0, & \text{else} \end{cases}$	$3h$	$\frac{1}{120h}$	$\frac{7}{478\pi h^2}$	$\frac{1}{120\pi h^3}$
johnson		$\alpha\left(\frac{3}{16}R^2 - \frac{3}{4}R + \frac{3}{4}\right), \quad 0 \le R \le 2$	$2h$	$\frac{1}{h}$	$\frac{2}{\pi h^2}$	$\frac{5}{4\pi h^3}$
wendland0	1	$\alpha\,(1.0-R)_+$	h	$\frac{1}{h}$	–	–
	2,3	$\alpha\,(1.0-R)_+^2$	h	–	$\frac{6}{\pi h^2}$	$\frac{15}{2\pi h^3}$
wendland1	1	$\alpha\,(1.0-R)_+^3\,(3R+1)$	h	$\frac{5}{4h}$	–	–
	2,3	$\alpha\,(1.0-R)_+^4\,(4R+1)$	h	–	$\frac{7}{\pi h^2}$	$\frac{21}{2\pi h^3}$
wendland2	1	$\alpha\,(1.0-R)_+^5\,(8R^2+5R+1)$	h	$\frac{3}{2h}$	–	–
	2,3	$\alpha\,(1.0-R)_+^6\,(35R^2+18R+1)$	h	–	$\frac{3}{\pi h^2}$	$\frac{165}{32\pi h^3}$
wendland3	2,3	$\alpha\,(1.0-R)_+^8\,(32R^3+25R^2+8R+1)$	h	–	$\frac{78}{7\pi h^2}$	$\frac{1365}{64\pi h^3}$

Table 4: Possible values for the option -i with a description and the number of function evaluations per time step.

name	description	evals/step
explicit	explicit Euler scheme	1
abm1	Adams-Bashforth multistep method with a history of length 1	1
abm2	Adams-Bashforth multistep method with a history of length 2	1
...	...	1
abm7	Adams-Bashforth multistep method with a history of length 7	1
pk1_1	Predictor-Corrector algorithm using abm1 as predictor and amm1 as corrector	1
pk2_1	Predictor-Corrector algorithm using abm2 as predictor and amm2 as corrector	1
...	...	1
pk7_1	Predictor-Corrector algorithm using abm7 as predictor and amm7 as corrector	1
rk2	Runge Kutta method of 2nd order	2
rk3	Runge Kutta method of 3nd order (Simpson's rule)	3
rk4	Runge Kutta method of 4nd order (Classic Runge-Kutta)	4
rk4_3_8	Runge Kutta 3/8 method of 4nd order	4
heun2	Heun method of 2nd order	2
heun3	Heun method of 3nd order	3
bs2	Bogacki-Shampine method of 2nd order	2
bs3	Bogacki-Shampine method of 3nd order	3
rkf4	Runge-Kutta-Fehlberg method of 4nd order	4
rkf5	Runge-Kutta-Fehlberg method of 5nd order	5
ck4	Cash-Karp method of 4nd order	4
ck5	Cash-Karp method of 5nd order	5
dp4	Dormant-Prince method of 4nd order	4
dp5	Dormant-Prince method of 5nd order	5

3.3.5 Sensors

Sphere implements several kinds of sensors to obtain information about the particles in a certain area. Each sensor has particle property, which it measures, e.g. density of the norm of the velocity. In addition, each sensor has a type, which specifies when a particle triggers a sensor. A sensor also has a position and sometimes an orientation or a size. Finally each sensor has a double variable v, which holds the accumulated sensor value, and a constant A, which holds the sensor area.

The current values of all sensors are printed for each time step in the *s* status element, as attributes *s0*, *s1*, *s2*, ... In addition elements *sensor-value* are printed after the end of simulation, which contain the the sensor values in the attribute *value*.

Configuring sensors Sensors can be configured for any test case by using the parameters. The parameter SENSORS determines the number of sensors. There is currently a maximum of 20 sensors that can be chosen.

For each sensor the sensors properties are configured using a set of indexed parameters **??** with the prefix SENSOR_S0 for the first sensor. The available parameters are

SHAPE – Defines the kind of area inclusion test used for the sensor
TYPE – Defines the particle property to be measured
POINTS_P1 – Defines the sensor position
POINTS_P2 – Defines the sensor size
VECTORS_P1 – Defines the sensor orientation. Optional
VECTORS_P2 – Defines the sensor orientation. Optional
An example of a valid configuration for one sensor is given here:

```
export SPH_PARAM_SENSORS=1
export SENSOR_S0_SHAPE=hyperplane_oriented
export SENSOR_S0_TYPE=pressure
export SENSOR_S0_POINTS_P1="0.04 0.5 0"       # point on plane
export SENSOR_S0_POINTS_P2="0 0 0"            # ignored
export SENSOR_S0_VECTORS_P1="1 0 0"           # normal vector
export SENSOR_S0_VECTORS_P2="0 1 0"           # ignored
```

Sensor specification In pseudo-code, the algorithm is:
if areaTest(p) addToSensorValue(p, particleProperty(p)) end

The areaTest function is chosen on the sensor area type, the particleProperty function is chosen according to the particle property that is to be measured. The function addToSensorValue is the same in all case. When v is the accumulated sensor value, and x is the property value read from the current particle, then addToSensorValue resets v as follows:

$$v = v + x * volume(p)/A * \delta t * T_{\max},$$

where $volume(p)$ returns the particle volume, A is the sensor area, δt is the time step, and $T_{\max}$ is the length of the simulation time span.

In addition, when the areaTest function returns true, i.e. the particle p has triggered the sensor, then a particular bit is set in the particles sensors field. Otherwise the bit is cleared. The first sensor configured sets or clears the first bit, the next the second and so forth. The

sensors field is not written by default. It can be written by adding "sensors" to the parameter SAVE_FIELDS. Note that as this flags is changed in each step, in order to get reliable results with this method, each time step should be saved, setting SAVE_EVERY to one.

The third measure is that each sensor records a hit in a log file. This avoids the need to save all particles in each time step. By default the sensor log file is the global log file, written to *out*.sphrun.xml. This setting can be overwritten using the parameter SENSOR_LOGFILE. This string is passed through sprintf(name, index), where index is a sensor's index. An individual log file can be specified for each sensor by using the per-sensor indexed parameter **??** LOGFILE.

The XML data recorded of sensor hits in the form of sensor-hit elements can be converted into a text, one line per hit, list using the XSLT sensor-hits.xsl.

The list of sensor shapes is:

ORTHOGONAL_RECTANGLE - A d-dimensional cuboid, spanned by the two points given by the POINTS area parameter. A particle triggers the sensor if it is inside the cuboid, the lower bounds being inclusive, the upper ones (given by _P2) exclusive.

HYPERPLANE - A d-dimensional hyperplane, given by one point c and a normal vector n. A particle triggers the sensor if it will cross the plane in the next time step, i.e. $(p.x - c) \cdot n$ is negative and $(p.x + p.v * \delta t - c) \cdot n$ is positive, or vice versa. The point c is given by the POINTS_P1 parameter and n by the VECTORS_P1 parameter. The sensor area is set to $A = 1$ in this case. Note that although POINTS_P2 and VECTORS_P2 are ignored, they must still be set.

HYPERPLANE_ORIENTED - A d-dimensional hyperplane, given by one point c and a normal vector n. The behaviour is the same as for type HYPERPLANE, except that the sensor is only triggered when the particle moves from the negative to the positive side of the plane, i.e. in direction of the normal vector. The sensor area is set to $A = 1$ in this case. Note that although POINTS_P2 and VECTORS_P2 are ignored, they must still be set.

TWO_HYPERPLANES - Two parallel d-dimensional hyperplanes, given by two points and one normal vector. A particle triggers the sensor if it is on the positive side of the first plane but on the negative side of the second plane. The points are given by the POINTS_P1 and POINTS_P2 area parameters and n by the VECTORS_P1 parameter. The sensor area is set to $A = 1$ in this case. Note that although VECTORS_P2 is ignored, it must still be set.

SPHERE - A d-dimensional sphere, given by one point and a radius. The sensor is triggered if a particles distance from the point given by the POINTS_P1 parameter is less than the distance between POINTS_P1 and POINTS_P2.

The list of measurable properties is given in the following. The values can be used as word arguments **??** in the parameter _TYPE of the i-th sensor parameter.

COUNT - The contribution is 1 for each partice

DENSITY - The contribution is the particle's density

ENERGY_HEIGHT - The particle's energy height is computed

IMPULSE - The norm of the particle velocity times the particle's mass

KINETIC_ENERGY - The square of the particle velocity times 0.5, times the particle's mass

MASS - The particle's mass

PRESSURE - The particle's pressure

VELOCITY - The norm of the particle's velocity

3.3.6 AD

The AD mode uses the operator overloading tool yafad. The two AD enables binaries, sph-2d-ad and sph-3d-ad are compiled from the source files sph-2d-ad.cc and sph-3d-ad.cc. These simply include yafad's header, set some defines and then include the sph.cc. Thus the AD programs are mostly from the same source as the non-AD programs, easing development.

Yafad is used to provide first order AD of the code in vector mode, the number of directional derivaties SPH_AD_NDIR is fixed at compile time in sph-2d-ad.cc and sph-3d-ad.cc, currently it is 3. In AD mode the code automatically saves the SPH_AD_NDIR derivative components of the particle properties density ρ (dichte), pressure P (druck), velocity $\vec{v}$ (vel) and location $\vec{x}$ (coord). Saving can be turned off by overriding the parameter SAVE_FIELDS. The vectors of derivatives of property prop are called ad_prop_d0, ad_prop_d1, ..., ad_prop_dn where n is the number of directions SPH_AD_NDIR. For the vector valued properties vel and coord the derivatives of each direction will be saved as vector properties as well.

The input derivatives (seeding) are read from the input VTK file. If one of the properties ad_coord_d0, ad_coord_d1 or ad_coord_d2 is encountered the data in it is loaded into the corresponding derivative component of the particle location $\vec{x}$.

Generating AD input cases with AD enabled SPHYSICS 2D

- sphysics2vtk_2D.sh – convert SPHYSICS 2D test cases into a VTK input file
- sphysics-make-ad-testcase.sh – generate a new SPHYSICS 2D test case directory based on an existing one, changing some parameters, and optionally run the SPHYSICS_2D program
- sphysics-ad-testcase.sh – combination of the above, generates temporary new test case directory and then generates a VTk file from that

sphysics2vtk_2D.sh The script takes as input a case directory from the 2D SPHYSICS SPH suite (one of the directories inside the run_directory). For example to convert the test case directory CaseTreppe1 to a VTK file called treppe1.vtk:

$ sh/sphysics2vtk_2D.sh −o treppe−test1.vtk \
path/to/adspyhsics/run_directory/CaseTreppeTest1/output

The conversion is done by a MATLAB/Octave script, by default Octave is used, MATLAB can be used by specifying the option -m. The option -o specifies the name of the output file. Using the option -h the help is printed:

```
$ sh/sphysics2vtk_2D.sh -h
sh/sphysics2vtk_2D.sh: usage: sphysics2vtk_2D.sh [ -o output ] <SPHYSICS-Case-Dir>
 -h               show help and exit
 -v               show version and exit
 -V               be verbose
 -o <name>        write output to file <name>
 -m               use MATLAB instead of octave
 -s <name>        set MATLAB/octave conversion script name
 -q               tell octave to be quiet (no banner)
```

sphysics-make-ad-testcase.sh This script takes an existing SPHYSICS case directory, given by option -C, possibly changes some parameters given by the options -s, -l, -x and -d and generates a new SPHYSICS test case directory whose name is given by option -o. Note that the names are given without the prefix "Case". For example generate a new case "TT23" based on existing case "TreppeTest1", setting the parameter stairLengh to 0.01 and the parameter wallSlope to 23, run the command:

sh/sphysics−make−ad−testcase.sh −VC TreppeTest1 −s 23 −l 0.01 −o TT23.

Using the option -r the newly generated test case will also be run. The option -h prints the help information:

```
sphysics-make-ad-testcase.sh - generate SPHYSICS 2D SPH scenerio with AD derivatives
usage: sphysics-make-ad-testcase.sh {option}
  -C <case>                    - name of case to generate and import
  -S <SPHYSICS2D-dir>          - path to where SPHYSICS2D is installed
  -s <number>                  - the variable wallSlope
  -l <number>                  - the variable stairLength
  -d <number>                  - the time step size dt
  -x <number>                  - the spatial resolution dx
  -o <file>                    - the output case name
  -r                           - run the new test case generated
  -h                           - show this help
  -V                           - be verbose
```

sphysics-ad-testcase.sh This script accepts the same options as sphysics-make-ad-testcase.sh, but it creates only a temporary SPHYSICS case directory, then runs the SPHYSICS to VTK conversion script sphysics2vtk_2D.sh to produce a VTK output file.

For example to generate a VTK input file called treppe1.vtk based on the test case "TreppeTest1", setting the parameter stairLengh to 0.01, the parameter wallSlope to 23 and the spatial resolution to 0.02, run the command:

$ sh/sphysics−ad−testcase.sh −o treppe1.vtk −x 0.02 −s 23 −l 0.01 −C TreppeTest1

The option -h prints the help information:

```
sphysics-ad-testcase.sh - generate and import SPHYSICS 2D SPH scenerio with AD derivatives
usage: sphysics-ad-testcase.sh {option}
  -C <case>                    - name of case to generate and import
  -S <SPHYSICS2D-dir>          - path to where SPHYSICS2D is installed
  -s <number>                  - the variable wallSlope
  -l <number>                  - the variable stairLength
  -d <number>                  - the time step size dt
  -x <number>                  - the spatial resolution dx
  -o <file>                    - the output file name
  -m                           - use MATLAB (instead of octave)
  -X <option>                  - pass option to sphysics2vtk.sh
  -h                           - show this help
  -q                           - be quiet
  -V                           - be verbose
```

3.3.7 Periodic boundaries

Periodic boundaries have been implemented to facilitate some test cases like the Poiseuille flow and falling film test cases. Enabling periodic boundary conditions in one dimension yields the surface of a cylinder as the computing domain in 2D and a torus in 3D. The boundaries in any dimension can be flagged as being periodic. Particles that cross the such a boundary are reinserted at the other side. Also, particles close to such a boundary may interact with particles close to the boundary on the other side.

Warning: Periodic boundaries only work correctly if the naive summation algorithm is selected.

For periodic boundary conditions to be available the compile time feature SPH_PERIODIC_BOUNDARIES has to be enabled. To switch on periodic boundary conditions use the environment variable SPH_PARAM_PERIODIC_BOUNDARIES. For example, for the boundaries in the Y-dimension to become periodic, type the following into your shell before starting the simulation program

$ **export** SPH_PARAM_PERIODIC_BOUNDARIES="0 1 0"

Then run the driver script as usual, but make sure you use the naive summation algorithm:

$ sphere −s naive −d 1e−5 −2 −m 5 vtk/**test**−cases2/2D/poiseuille−flow/mittel/**test**.vtk

3.3.8 Online OpenGL rendering

Usage The interactive online render window opens automatically when a simulation starts and the feature SPH_RENDER has been compiled in. The sphere option -r sets the rate of rendering, the date is rendered every r-th step. If -r 0 is specified the rendered window does not open.

The render is mostly controlled by keystrokes and the mouse wheel. Pressing h makes the program print a help listing detailing the usage. There is also a menu which opens by right-clicking on the window.

Particle representation Particles are represented either as spheres or as points, which can be chosen in the menu. If spheres are selected, then holding shift and turning the mouse wheel changes the scale factor for the spheres by powers of 2. The base size is given either by the particle volume as determined by (11), which is the default or by the globally fixed value H. Thus with the first possibility particles with smaller density are rendered as bigger spheres.

The particles are colored according to one of several properties, both numeric and other. This can be choosen via the menu. If a vector property is chosen (velocity $\mathbf{v_i}$ or the derivative of velocity or position), the magnitude is used.

4 The SPH scene compiler

5 Tools and utilities

5.1 Paraview

Download

Open VTK series

Periodic boundries: open twice, use transform to place to copies side by side

Save animation

Movie: ffmpeg -y -i 'test.%04d.jpg' -r 100 -vcodec libxvid -b 4000k output.avi. Other codecs: ffmpeg -codecs — grep EV

References

[FDTA09] Angela Ferrari, Michael Dumbser, Eleuterio F. Toro, and Aronne Armanini. A new 3D parallel SPH scheme for free surface flows. *Computers & Fluids*, 38(6):1203–1217, 2009.

[GJU96] Andreas Griewank, David Juedes, and Jean Utke. ADOL–C, a package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software*, 22(2):131–167, 1996.

[LL03] G. R. Liu and M. B. Liu. *Smoothed Particle Hydrodynamics*. Singapore, World Scientific Publishing Co. Pte. Ltd, 2003.

[Mon94] J. J. Monaghan. Simulating free surface flows with SPH. *J. Comput. Phys.*, 110(2):399–406, 1994.

[Wen95] Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Adv. Comput. Math.*, 4(1):389–396, 1995.