

---

**SPIsHSPiS**

***Release 2.11.0***

**Interactive Computer Graphics**

**Sep 07, 2021**



# INTRODUCTION:

<b>1</b>	<b>Main features</b>	<b>3</b>
<b>2</b>	<b>License</b>	<b>5</b>
<b>3</b>	<b>Getting started</b>	<b>7</b>
3.1	SPHSimulator . . . . .	7
3.2	Python bindings . . . . .	8
<b>4</b>	<b>SPlisHSPlasH Scene Files</b>	<b>9</b>
4.1	Configuration . . . . .	9
4.2	FluidBlocks . . . . .	13
4.3	FluidModels . . . . .	14
4.4	Emitters . . . . .	14
4.5	RigidBodies . . . . .	15
4.6	Materials . . . . .	16
4.7	Animation fields . . . . .	19
<b>5</b>	<b>Replicability</b>	<b>21</b>
<b>6</b>	<b>Installation Instructions - Linux</b>	<b>23</b>
6.1	Ubuntu Fresh Install . . . . .	23
<b>7</b>	<b>Installation Instructions - Windows</b>	<b>25</b>
7.1	Visual Studio . . . . .	25
<b>8</b>	<b>CMake Options</b>	<b>27</b>
8.1	USE_DOUBLE_PRECISION . . . . .	27
8.2	USE_AVX . . . . .	27
8.3	USE_OpenMP . . . . .	27
8.4	USE_GPU_NEIGHBORHOOD_SEARCH . . . . .	27
8.5	USE_IMGUI . . . . .	28
8.6	USE_PYTHON_BINDINGS . . . . .	28
8.7	USE_DEBUG_TOOLS . . . . .	28
<b>9</b>	<b>Software Architecture</b>	<b>29</b>
9.1	The Simulation class . . . . .	29
9.2	The TimeStep class . . . . .	30
9.3	The FluidModel class . . . . .	30
9.4	The BoundaryModel class . . . . .	31
<b>10</b>	<b>Implementing a new non-pressure force method</b>	<b>33</b>

10.1	Creating a new class . . . . .	33
10.2	Registering the viscosity method . . . . .	36
<b>11</b>	<b>Implementing a new particle/rigid body data exporter</b>	<b>37</b>
11.1	Creating a new class . . . . .	37
11.2	Registering the exporter . . . . .	39
11.3	Implementing a new exporter in Python . . . . .	40
<b>12</b>	<b>Creating Pressure Solvers</b>	<b>41</b>
12.1	Creating a new class . . . . .	41
12.2	Registering the pressure solver . . . . .	43
<b>13</b>	<b>Macros</b>	<b>45</b>
13.1	Looping over fluid neighbors . . . . .	45
13.2	Looping over boundaries . . . . .	46
13.3	AVX variants . . . . .	48
<b>14</b>	<b>pySPlisHSPlasH</b>	<b>49</b>
14.1	Python bindings for the SPlisHSPlasH library . . . . .	49
14.2	Requirements . . . . .	49
14.3	Installation . . . . .	49
14.4	I want to see something very very quickly . . . . .	50
14.5	Minimal working example . . . . .	50
14.6	SPHSimulator.py . . . . .	51
14.7	Modifying other properties . . . . .	51
<b>15</b>	<b>Embedded Python</b>	<b>53</b>
15.1	Build with embedded Python support . . . . .	53
15.2	Run simulator with embedded Python support . . . . .	53
15.3	Writing a script . . . . .	53
15.4	Example . . . . .	54
<b>16</b>	<b>Creating Scenes</b>	<b>55</b>
16.1	Loading the empty scene . . . . .	55
16.2	Recreating the double dam break scenario . . . . .	55
16.3	Putting it all together . . . . .	56
16.4	Loading a scene from file . . . . .	56
<b>17</b>	<b>Restrictions</b>	<b>57</b>
<b>18</b>	<b>FoamGenerator</b>	<b>59</b>
18.1	Parameters for foam generation . . . . .	59
18.2	Bounding box . . . . .	60
18.3	Frame rate . . . . .	60
18.4	Further parameters . . . . .	60
<b>19</b>	<b>MeshSkinning</b>	<b>63</b>
19.1	Parameters . . . . .	64
<b>20</b>	<b>partio2vtk</b>	<b>65</b>
<b>21</b>	<b>PartioViewer</b>	<b>67</b>
21.1	Command line options: . . . . .	67
21.2	Hotkeys . . . . .	68
<b>22</b>	<b>SurfaceSampling</b>	<b>69</b>

<b>23</b>	<b>VolumeSampling</b>	<b>71</b>
23.1	Command line options: . . . . .	73
23.2	Example: . . . . .	73
<b>24</b>	<b>Library API</b>	<b>75</b>
24.1	Full API . . . . .	75
<b>25</b>	<b>References</b>	<b>237</b>
<b>26</b>	<b>Indices and tables</b>	<b>239</b>
	<b>Bibliography</b>	<b>241</b>
	<b>Index</b>	<b>245</b>





SPlisHSPlasH is an open-source library for the physically-based simulation of fluids. The simulation in this library is based on the Smoothed Particle Hydrodynamics (SPH) method which is a popular meshless Lagrangian approach to simulate complex fluid effects. The SPH formalism allows an efficient computation of a certain quantity of a fluid particle by considering only a finite set of neighboring particles. One of the most important research topics in the field of SPH methods is the simulation of incompressible fluids. SPlisHSPlasH implements current state-of-the-art pressure solvers (WCSPH, PCISPH, PBF, IISPH, DFSPH, PF) to simulate incompressibility. Moreover, the library provides different methods to simulate viscosity, surface tension and vorticity.



---

**CHAPTER  
ONE**

---

**MAIN FEATURES**

- an open-source SPH fluid simulation (2D & 3D)
- neighborhood search on CPU or GPU
- supports vectorization using AVX
- Python binding (thanks to Stefan Jeske)
- supports embedded Python scripts
- several implicit pressure solvers (WCSPH, PCISPH, PBF, IISPH, DFSPH, PF)
- explicit and implicit viscosity methods
- current surface tension approaches
- different vorticity methods
- computation of drag forces
- support for multi-phase simulations
- simulation of deformable solids
- rigid-fluid coupling with static and dynamic bodies
- two-way coupling with deformable solids
- fluid emitters
- scripted animation fields
- a json-based scene file importer
- automatic surface sampling
- a tool for volume sampling of closed geometries
- partio file export of all particle data
- VTK file export of all particle data (enables the data import in ParaView)
- rigid body export
- a Maya plugin to model and generate scene files
- a ParaView plugin to import particle data



---

## **CHAPTER**

## **TWO**

---

## **LICENSE**

The MIT License (MIT)

Copyright (c) 2016-present, SPlisHSPlasH contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



## GETTING STARTED

This page should give you a short overview of SPlisHSPlasH.

SPlisHSPlasH currently consists of a simulators and different tools which are introduced in the following:

### 3.1 SPHSimulator

This application reads a SPlisHSPlasH scene file and performs a simulation of the scene.

The scene file format is explained [here](#).

#### 3.1.1 Command line options:

- -h, --help: Print help text.
- -v, --version: Print version.
- --no-cache: Disable caching of boundary samples/maps.
- --state-file: Load a simulation state of the corresponding scene.
- --output-dir: Output directory for log file and partio files.
- --no-initial-pause: Disable caching of boundary samples/maps.
- --no-gui: Disable graphical user interface. The simulation is run only in the command line without graphical output. The “stopAt” option must be set in the scene file or by the next parameter.
- --stopAt arg: Sets or overwrites the stopAt parameter of the scene.
- --param arg: Sets or overwrites a parameter of the scene.
  - Setting a fluid parameter: ::
    - \* Example: --param Fluid:viscosity:0.01
  - Setting a configuration parameter: :
    - \* Example: --param cflMethod:1

### 3.1.2 Hotkeys

- Space: pause/continunue simulation
- r: reset simulation
- w: wireframe rendering of meshes
- m: recompute min and max values for color-coding the color field in the rendering process
- i: print all field information of the selected particles to the console
- s: save current simulation state
- l: load simulation state (currently only Windows)
- +: perform a single time step
- ESC: exit

## 3.2 Python bindings

SPlisHSPlasH implements bindings for python using `pybind11`. See the [getting started guide](#).

### 3.2.1 Impatient installation guide

In order to install, simply clone the repository and run pip install on the repository. It is recommended, that you set up a **virtual environment** for this, because cache files will be stored in the directory of the python installation along with models and scene files.

```
git clone https://github.com/InteractiveComputerGraphics/SPlisHSPlasH.git  
pip install SPlisHSPlasH/
```

## SPLISHSPLASH SCENE FILES

A SPlisHSPlasH scene file is a json file which can contain the following blocks:

- Configuration
- FluidBlocks
- FluidModels
- Emitters
- RigidBodies
- Fluid parameter block
- Animation fields

### 4.1 Configuration

This part contains the general settings of the simulation and the pressure solver.

Example code:

```
"Configuration":  
{  
    "pause": true,  
    "sim2D": false,  
    "timeStepSize": 0.001,  
    "numberOfStepsPerRenderUpdate": 2,  
    "particleRadius": 0.025,  
    "simulationMethod": 4,  
    "gravitation": [0.0,-9.81,0],  
    "cflMethod": 1,  
    "cflFactor": 1,  
    "cflMaxTimeStepSize": 0.005,  
    "maxIterations": 100,  
    "maxError": 0.01,  
    "maxIterationsV": 100,  
    "maxErrorV": 0.1,  
    "stiffness": 50000,  
    "exponent": 7,  
    "velocityUpdateMethod": 0,  
    "enableDivergenceSolver": true  
}
```

#### 4.1.1 General:

- pause (bool): Pause simulation at beginning.
- pauseAt (float): Pause simulation at the given time. When the value is negative, the simulation is not paused.
- stopAt (float): Stop simulation at the given time and exit. When the value is negative, the simulation is not stopped.
- cameraPosition (vec3): Initial position of the camera.
- cameraLookat (vec3): Lookat point of the camera.

#### 4.1.2 Visualization:

- numberOfStepsPerRenderUpdate (int): Number of simulation steps per rendered frame
- renderWalls (int):
  - 0: None
  - 1: Particles (all)
  - 2: Particles (no walls)
  - 3: Geometry (all)
  - 4: Geometry (no walls)

#### 4.1.3 Export

- enablePartioExport (bool): Enable/disable partio export (default: false).
- enableVTKExport (bool): Enable/disable VTK export (default: false).
- enableRigidBodyExport (bool): Enable/disable rigid body export (default: false).
- enableRigidBodyVTKExport (bool): Enable/disable rigid body VTK export (default: false).
- dataExportFPS (float): Frame rate of particle and rigid body export (default: 25).
- particleAttributes (string): A list of attribute names separated by ";" that should be exported in the particle files (e.g. "velocity;density") (default: "velocity").
- enableStateExport (bool): Enable/disable export of complete simulation state (default: false).
- stateExportFPS (float): Frame rate of simulation state export (default: 1).

#### 4.1.4 Simulation:

- timeStepSize (float): The initial time step size used for the time integration. If you use an adaptive time stepping, this size will change during the simulation (default: 0.001).
- particleRadius (float): The radius of the particles in the simulation (all have the same radius) (default: 0.025).
- sim2D (bool): If this parameter is set to true, a 2D simulation is performed instead of a 3D simulation (default: false).
- enableZSort (bool): Enable z-sort to improve cache hits and therefore to improve the performance (default: true).
- gravitation (vec3): Vector to define the gravitational acceleration (default: [0,-9.81,0]).

- maxIterations (int): Maximal number of iterations of the pressure solver (default: 100).
- maxError (float): Maximal density error in percent which the pressure solver tolerates (default: 0.01).
- boundaryHandlingMethod (int): The boundary handling method that is used in the simulation (default: 2, Volume Maps):
  - 0: particle-based boundaries (Akinci et al. 2012)
  - 1: density maps (Koschier et al. 2017)
  - 2: volume maps (Bender et al. 2019)
- simulationMethod (int): The pressure solver method used in the simulation (default: 4, DFSPH):
  - 0: Weakly compressible SPH for free surface flows (WCSPH)
  - 1: Predictive-corrective incompressible SPH (PCISPH)
  - 2: Position based fluids (PBF)
  - 3: Implicit incompressible SPH (IISPH)
  - 4: Divergence-free smoothed particle hydrodynamics (DFSPH)
  - 5: Projective Fluids (dynamic boundaries not supported yet)
  - 6: Implicit compressible SPH (ICSPH)

#### 4.1.5 WCSPH parameters:

- stiffness (float): Stiffness coefficient of the equation of state.
- exponent (float): Exponent in the equation of state.

#### 4.1.6 PBF parameters:

- velocityUpdateMethod (int):
  - 0: First Order Update
  - 1: Second Order Update

#### 4.1.7 DFSPH parameters:

- enableDivergenceSolver (bool): Turn divergence solver on/off.
- maxIterationsV (int): Maximal number of iterations of the divergence solver.
- maxErrorV (float): Maximal divergence error in percent which the pressure solver tolerates.

#### **4.1.8 Projective Fluids parameters:**

- stiffness (float): Stiffness coefficient used by the pressure solver.

#### **4.1.9 ICSPH parameters:**

- lambda (float): Stiffness coefficient of the equation of state.
- pressureClamping (bool): Enable pressure clamping.

#### **4.1.10 Kernel:**

- kernel (int): Kernel function used in the SPH model.
  - For a 3D simulation:
    - \* 0: Cubic spline
    - \* 1: Wendland quintic C2
    - \* 2: Poly6
    - \* 3: Spiky
    - \* 4: Precomputed cubic spline (faster than cubic spline)
  - For a 2D simulation:
    - \* 0: Cubic spline
    - \* 1: Wendland quintic C2
- gradKernel (int): Gradient of the kernel function used in the SPH model.
  - For a 3D simulation:
    - \* 0: Cubic spline
    - \* 1: Wendland quintic C2
    - \* 2: Poly6
    - \* 3: Spiky
    - \* 4: Precomputed cubic spline (faster than cubic spline)
  - For a 2D simulation:
    - \* 0: Cubic spline
    - \* 1: Wendland quintic C2

#### 4.1.11 CFL:

- cflMethod (int): CFL method used for adaptive time stepping.
  - 0: No adaptive time stepping
  - 1: Use CFL condition
  - 2: Use CFL condition and consider number of pressure solver iterations
- cflFactor (float): Factor to scale the CFL time step size.
- cflMinTimeStepSize (float): Min. allowed time step size.
- cflMaxTimeStepSize (float): Max. allowed time step size.

## 4.2 FluidBlocks

In this part the user can define multiple axis-aligned blocks of fluid particles.

Example code:

```
"FluidBlocks": [
  {
    "denseMode": 0,
    "start": [-2.0, 0.0, -1],
    "end": [-0.5, 1.5, 1],
    "translation": [1.0, 0.0, 0.0],
    "scale": [1, 1, 1]
  }
]
```

- start (vec3): Minimum coordinate of the box which defines the fluid block.
- end (vec3): Maximum coordinate of the box which defines the fluid block.
- translation (vec3): Translation vector of the block.
- scale (vec3): Scaling vector of the block.
- denseMode (int):
  - 0: regular sampling
  - 1: more dense sampling
  - 2: dense sampling
- initialVelocity (vec3): The initial velocity is set for all particles in the block.
- initialAngularVelocity (vec3): The initial angular velocity of the block.
- id (string): This id is used in the “Fluid parameter block” (see below) to define the properties of the fluid block. If no id is defined, then the standard id “Fluid” is used.

## 4.3 FluidModels

This part can be used to import one or more partio particle files in the scene.

Example code:

```
"FluidModels": [
    {
        "particleFile": "../models/bunny.bgeo",
        "translation": [-2.0, 0.1, 0.0],
        "rotationAxis": [0, 1, 0],
        "rotationAngle": 3.14159265359,
        "scale": 1
    }
]
```

- particleFile (string): Path of the partio file which contains the particle data.
- translation (vec3): Translation vector of the fluid model.
- scale (vec3): Scaling vector of the fluid model.
- rotationAxis (vec3): Axis used to rotate the particle data after loading.
- rotationAngle (float): Rotation angle for the initial rotation of the particle data.
- id: This id is used in the “Fluid parameter block” (see below) to define the properties of the fluid block. If no id is defined, then the standard id “Fluid” is used.
- initialVelocity (vec3): The initial velocity is set for all particles in the fluid model.
- initialAngularVelocity (vec3): The initial angular velocity of the fluid model.

## 4.4 Emitters

In this part the user can define one or more emitters which generate fluid particles.

Example code:

```
"Emitters": [
    {
        "width": 5,
        "height": 5,
        "translation": [-1, 0.75, 0.0],
        "rotationAxis": [0, 1, 0],
        "rotationAngle": 3.1415926535897932384626433832795,
        "velocity": 2,
        "emitStartTime": 2,
        "emitEndTime": 6,
        "type": 0
    }
]
```

- type (int): Defines the shape of the emitter (default: 0).
  - 0: box
  - 1: circle
- width (int): Width of the box or radius of the circle emitter in number of particles (default: 5).

- height (int): Height of the box in number of particles (is only used for type 0) (default: 5).
- translation (vec3): Translation vector of the emitter (default: [0,0,0]).
- rotationAxis (vec3): Axis used to rotate the emitter. Note that in 2D simulations the axis is always set to [0,0,1] (default: [0,0,1]).
- rotationAngle (float): Rotation angle for the initial rotation of the emitter (default: 0).
- velocity (float): Initial velocity of the emitted particles in direction of the emitter (default: 1).
- id: This id is used in the “Fluid parameter block” (see below) to define the properties of the fluid block. If no id is defined, then the standard id “Fluid” is used (default: “Fluid”).
- emitStartTime (float): Start time of the emitter (default: 0).
- emitEndTime (float): End time of the emitter (default: REAL\_MAX).

## 4.5 RigidBodies

Here, the static and dynamic rigid bodies are defined which define the boundary in the scene. In case of dynamic rigid bodies, the PositionBasedDynamics library is used for their simulation. Note that in this case the PositionBasedDynamics library also reads this json scene files and picks out the relevant parts. That means if you want to define for example a hinge joint or a motor, then just use the json format of PositionBasedDynamics in this scene file.

Example code:

```
"RigidBodies": [
    {
        "geometryFile": "../models/UnitBox.obj",
        "translation": [0, 2, 0],
        "rotationAxis": [1, 0, 0],
        "rotationAngle": 0,
        "scale": [2.5, 4, 1.0],
        "color": [0.1, 0.4, 0.6, 1.0],
        "isDynamic": false,
        "isWall": true,
        "mapInvert": true,
            "mapThickness": 0.0,
            "mapResolution": [20, 20, 20],
        "samplingMode": 1
    }
]
```

- geometryFile (string): Path to a OBJ file which contains the geometry of the body.
- particleFile (string): Path to a partio file which contains a surface sampling of the body. Note that the surface sampling is done automatically if this parameter is missing.
- translation (vec3): Translation vector of the rigid body.
- scale (vec3): Scaling vector of the rigid body.
- rotationAxis (vec3): Axis used to rotate the rigid body after loading.
- rotationAngle (float): Rotation angle for the initial rotation of the rigid body.
- isDynamic (bool): Defines if the body is static or dynamic.
- isWall (bool): Defines if this is a wall. Walls are typically not rendered. This is the only difference.
- color (vec4): RGBA color of the body.

- mapInvert (bool): Invert the map when using density or volume maps, flips inside/outside (default: false)
- mapThickness (float): Additional thickness of a volume or density map (default: 0.0)
- mapResolution (vec3): Resolution of a volume or density map (default: [20,20,20])
- samplingMode (int): Surface sampling mode. 0 Poisson disk sampling, 1 Regular triangle sampling (default: 0).

## 4.6 Materials

```
"Materials": [
    {
        "id": "Fluid",
        "density0": 1000,
        "colorField": "velocity",
        "colorMapType": 1,
        "renderMinValue": 0.0,
        "renderMaxValue": 5.0,
        "surfaceTension": 0.2,
        "surfaceTensionMethod": 0,
        "viscosity": 0.01,
        "viscosityMethod": 1,
        "vorticityMethod": 1,
        "vorticity": 0.15,
        "viscosityOmega": 0.05,
        "inertiaInverse": 0.5,
        "maxEmitterParticles": 1000,
        "emitterReuseParticles": false,
        "emitterBoxMin": [-4.0,-1.0,-4.0],
        "emitterBoxMax": [0.0,4,4.0]
    }
]
```

### 4.6.1 General

- id (string): Defines the id of the material. You have to give the same id to a FluidBlock, a FluidModel or an Emitter if they should have the defined material behavior.
- density0 (float): Rest density of the corresponding fluid.

### 4.6.2 Particle Coloring

- colorField (string): Choose vector or scalar field for particle coloring.
- colorMapType (int): Selection of a color map for coloring the scalar/vector field.
  - 0: None
  - 1: Jet
  - 2: Plasma
  - 3: CoolWarm
  - 4: BlueWhiteRed

- 5: Seismic
- renderMinValue (float): Minimal value used for color-coding the color field in the rendering process.
- renderMaxValue (float): Maximal value used for color-coding the color field in the rendering process.

### 4.6.3 Viscosity

- viscosityMethod (int): Viscosity method
  - 0: None
  - 1: Standard
  - 2: XSPH
  - 3: Bender and Koschier 2017
  - 4: Peer et al. 2015
  - 5: Peer et al. 2016
  - 6: Takahashi et al. 2015 (improved)
  - 7: Weiler et al. 2018
- viscosity (float): Coefficient for the viscosity force computation
  - “Standard” and “Weiler et al. 2018” use the kinematic viscosity as parameter
  - “Bender and Koschier 2017” and “Peer et al. 2015/2016” use a coefficient in [0,1]
  - For “XSPH” the coefficient is the smoothing parameter in [0,1]
- viscoMaxIter (int): (Implicit solvers) Max. iterations of the viscosity solver.
- viscoMaxError (float): (Implicit solvers) Max. error of the viscosity solver.
- viscoMaxIterOmega (int): (Peer et al. 2016) Max. iterations of the vorticity diffusion solver.
- viscoMaxErrorOmega (float): (Peer et al. 2016) Max. error of the vorticity diffusion solver.
- viscosityBoundary (float): (Weiler et al. 2018) Coefficient for the viscosity force computation at the boundary.

### 4.6.4 Vorticity

- vorticityMethod (int): Vorticity method
  - 0: None
  - 1: Micropolar model
  - 2: Vorticity confinement
- vorticity (float): Coefficient for the vorticity force computation
- viscosityOmega (float): (Micropolar model) Viscosity coefficient for the angular velocity field.
- inertiaInverse (float): (Micropolar model) Inverse microinertia used in the micropolar model.

#### 4.6.5 Drag force

- dragMethod (int): Drag force method
  - 0: None
  - 1: Macklin et al. 2014
  - 2: Gissler et al. 2017
- drag (float): Coefficient for the drag force computation

#### 4.6.6 Surface tension

- surfaceTensionMethod (int): Surface tension method
  - 0: None
  - 1: Becker & Teschner 2007
  - 2: Akinci et al. 2013
  - 3: He et al. 2014
- surfaceTension (float): Coefficient for the surface tension computation

#### 4.6.7 Elasticity

- elasticityMethod (int): Elasticity method
  - 0: None
  - 1: Becker et al. 2009
  - 2: Peer et al. 2018
- youngsModulus (float): Young's modulus - coefficient for the stiffness of the material (default: 100000.0)
- poisssonsRatio (float): Poisson's ratio - measure of the Poisson effect (default: 0.3)
- alpha (float): Coefficient for zero-energy modes suppression method (default: 0.0)
- elasticityMaxIter (int): (Peer et al. 2018) Maximum solver iterations (default: 100)
- elasticityMaxError (float): (Peer et al. 2019) Maximum elasticity error allowed by the solver (default: 1.0e-4)

#### 4.6.8 Emitters

- maxEmitterParticles (int): Maximum number of particles the emitter generates. Note that reused particles (see below) are not counted here.
- emitterReuseParticles (bool): Reuse particles if they are outside of the bounding box defined by emitterBoxMin, emitterBoxMax
- emitterBoxMin (vec3): Minimum coordinates of an axis-aligned box (used in combination with emitterReuseParticles)
- emitterBoxMax (vec3): Maximum coordinates of an axis-aligned box (used in combination with emitterReuseParticles)

## 4.7 Animation fields

In this part the user can define one or more animation fields which animate fluid particles. The user can define math expressions for the components of the field quantity. The typical math terms like cos,sin,... can be used.

Available expression variables:

- t: Current time.
- dt: Current time step size.
- x, y, z: Position of the particle which is in the animation field.
- vx, vy, vz: Velocity of the particle which is in the animation field.
- valuex, valuey, valuez: Value of the field quantity of the particle which is in the animation field.

Example:

```
"particleField": "angular velocity",
"expression_x": "valuex + cos(2*t)"
```

This means that in each step we add  $\cos(2*t)$  to the x-component of the angular velocity.

Example code:

```
"AnimationFields": [
    {
        "particleField": "velocity",
        "translation": [-0.5, -0.5, 0],
        "rotationAxis": [0, 0, 1],
        "rotationAngle": 0.0,
        "scale": [0.5, 0.25, 0.8],
        "shapeType": 0,
        "expression_x": "cos(2*t)*0.1",
        "expression_y": "",
        "expression_z": ""
    }
]
```

- shapeType (int): Defines the shape of the animation field (default: 0).
  - 0: box
  - 1: sphere
  - 2: cylinder
- particleField (string): Defines the field quantity that should be modified by the field (e.g. velocity, angular velocity, position) (default: velocity)
- translation (vec3): Translation vector of the animation field (default: [0,0,0]).
- rotationAxis (vec3): Axis used to rotate the animation field (default: [0,0,1]).
- rotationAngle (float): Rotation angle for the initial rotation of the animation field (default: 0).
- scale (vec3): Scaling vector of the animation field.
  - shapeType=0 (box): This vector defines the width, height, depth of the box.
  - shapeType=1 (sphere): The x-component of the vector defines the radius of the sphere. The other components are ignored.

- shapeType=2 (cylinder): The x- and y-component of the vector defines the height and radius of the cylinder, respectively. The z-component is ignored.
- expression\_x (string): Math expression for the x-component of the field quantity (default="").
- expression\_y (string): Math expression for the y-component of the field quantity (default="").
- expression\_z (string): Math expression for the z-component of the field quantity (default="").

## REPLICABILITY

The SPPlisHSPlasH library implements the SPH methods developed by our and other research groups (build instructions can be found [here](#)). This allows to reproduce the research results of the corresponding publications. Inspired by the [Graphics Replicability Stamp Initiative](#) we started to add scenes to the repository to reproduce some of the results in our papers:

**Jan Bender, Tassilo Kugelstadt, Marcel Weiler, Dan Koschier, “Implicit Frictional Boundary Handling for SPH”, IEEE Transactions on Visualization and Computer Graphics, 2020**

- Figure 7.a) can be replicated by loading the scene: data/Scenes/GridModel\_Akinci2012.json
- Figure 7.b) can be replicated by loading the scene: data/Scenes/GridModel\_Bender2019.json



## INSTALLATION INSTRUCTIONS - LINUX

### 6.1 Ubuntu Fresh Install

#### 6.1.1 Installation List

```
sudo apt install git cmake xorg-dev freeglut3-dev build-essential
```

#### 6.1.2 Python Bindings

If you plan on using the python bindings by specifying `-DUSE_PYTHON_BINDINGS=On`, then you should also have a working python installation in your path. This installs an additional tool pipx, which allows the installation of packages as executables in virtualized environments.

```
sudo apt install python3-dev python3-pip python3-venv
python3 -m pip install pipx
python3 -m pipx ensurepath
```

Alternatively to this you may also install other Python Distributions such as Anaconda (personal preference).

#### 6.1.3 Building Instructions

```
git clone https://github.com/InteractiveComputerGraphics/SPHsPlasH.git
cd SPHsPlasH
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release -DUSE_PYTHON_BINDINGS=<On|Off> ..
make -j 4
```

#### 6.1.4 Run Executable

```
cd ../bin
./SPHSimulator ../data/Scenes/DoubleDamBreak.json
```

On some systems it may be necessary to define an OpenGL override like so

```
cd ../bin
MESA_GL_VERSION_OVERRIDE=3.3 ./SPHSimulator ../data/Scenes/DoubleDamBreak.json
```

The command loads the selected scene. To start the simulation disable the pause mode by clicking the checkbox or pressing [Space]. More hotkeys are listed [here](#).

### 6.1.5 Using Bindings

Assuming that the python bindings were generated in the default location Project Root/build/lib/pysplishsplash.cpython-38-x86\_64-linux-gnu.so, you can use the bindings by adding this path to sys.path within your python script, or by calling your scripts within the directory containing the .so file. You can test that the bindings work using the following command.

```
cd lib  
python3 -c "import pysplishsplash"
```

### 6.1.6 Installing Bindings

If you followed the above instructions for building SPlisHSPlasH using CMake and generated the python bindings, then these commands should work automatically.

**Note:** You don't have to clone the repository again. This only shows, that the command should be run in the project root directory. It is also recommended, that you create and activate a virtual environment before installing, so that your base python installation is not affected by any new generated files.

```
git clone https://github.com/InteractiveComputerGraphics/SPlisHSPlasH.git  
cd SPlisHSPlasH  
python setup.py bdist_wheel  
pip install build/dist/*.whl
```

If you specified any additional CMake variables in the form of `-DVAR_NAME=Value`, you can just append them after `bdist_wheel`

Alternatively you may also run the following command, which essentially combines all of the above commands into a single command.

```
pip install git+https://github.com/InteractiveComputerGraphics/SPlisHSPlasH.git
```

**Drawbacks:** You lose the ability for incremental rebuilds, i.e. if you want to modify the source code and build the bindings anew, you would have to build the entire project every time.

## INSTALLATION INSTRUCTIONS - WINDOWS

### 7.1 Visual Studio

#### 7.1.1 Dependencies

To build SPlisHSPlasH on Windows you need to install CMake and git.

#### 7.1.2 Python Bindings

If you plan on using the python bindings by specifying `-DUSE_PYTHON_BINDINGS=On`, then you should also have a working Python installation in your path. Moreover, you require the Python Package Installer (pip).

#### 7.1.3 Building Instructions

First, clone the repository by

```
git clone https://github.com/InteractiveComputerGraphics/SPlisHSPlasH.git
```

Then run cmake-gui and set “Where is the source code:” to the [SPlisHSPlasH-dir] and “Where to build the binaries:” to [SPlisHSPlasH-dir]/build.

Now run Configure and select the correct Visual Studio version. Ensure that you choose a x64 build on a 64bit system. Finally, run Generate and open the project. Now you can build the project in Visual Studio. Note that you have to select the “Release” build, if you want to have an optimized executable.

#### 7.1.4 Run Executable

Execute “bin/SPHSimulator.exe” to start the simulator and select a scene file to run the simulation. Alternatively, you can start the simulation in the command line:

```
./SPHSimulator ..../data/Scenes/DoubleDamBreak.json
```

The command loads the selected scene. To start the simulation disable the pause mode by clicking the checkbox or pressing [Space]. More hotkeys are listed [here](#).

### 7.1.5 Using Bindings

Assuming that the python bindings were generated in the default location [SPlisHSPlasH-dir]/build/lib/pysplishsplash.cp37-win\_amd64.pyd, you can use the bindings by adding this path to `sys.path` within your python script, or by calling your scripts within the directory containing the .pyd file. You can test that the bindings work using the following command.

```
cd lib  
python3 -c "import pysplishsplash"
```

### 7.1.6 Installing Bindings

If you followed the above instructions for building SPlisHSPlasH using CMake and generated the python bindings, then these commands should work automatically.

**Note:** You don't have to clone the repository again. This only shows, that the command should be run in the project root directory. It is also recommended, that you create and activate a virtual environment before installing, so that your base python installation is not affected by any new generated files.

```
git clone https://github.com/InteractiveComputerGraphics/SPlisHSPlasH.git  
cd SPlisHSPlasH  
python setup.py bdist_wheel  
pip install build/dist/pySPlisHSPlasH-2.8.3-cp37-cp37m-win_amd64.whl
```

If you specified any additional CMake variables in the form of `-DVAR_NAME=Value`, you can just append them after `bdist_wheel`

Alternatively you may also run the following command, which essentially combines all of the above commands into a single command.

```
pip install git+https://github.com/InteractiveComputerGraphics/SPlisHSPlasH.git
```

**Drawbacks:** You lose the ability for incremental rebuilds, i.e. if you want to modify the source code and build the bindings anew, you would have to build the entire project every time.

## CMAKE OPTIONS

This page should give you a short overview over the CMake options of SPPlisHSPlasH.

### **8.1 USE\_DOUBLE\_PRECISION**

If this flag is enabled, then all computations with floating point values are performed using double precision (double). Otherwise single precision (float) is used.

### **8.2 USE\_AVX**

SPlishSPlasH supports the usage of AVX (Advanced Vector Extensions) which is an extension of modern CPUs to perform a single instruction on multiple data. The extension allows to perform eight floating point operations in parallel. Enabling AVX significantly improves the performance of the simulator. Currently, the following methods have AVX support:

- DFSPH
- the micropolar vorticity model
- the standard viscosity model
- the viscosity model of Weiler et al.

### **8.3 USE\_OpenMP**

Enable the OpenMP parallelization which lets the simulation run in parallel on all available cores of the CPU.

### **8.4 USE\_GPU\_NEIGHBORHOOD\_SEARCH**

As default SPPlisHSPlasH uses `CompactNSearch` as neighborhood search which performs all operations on the CPU. However, with this flag you can switch to `cuNSearch` which is our GPU neighborhood search. In case you want to use the GPU method, you have to install Cuda.

## **8.5 USE\_IMGUI**

We just reimplemented the GUI using `imgui` instead of `AntTweakBar`. If you want to try out the new GUI, enable this flag.

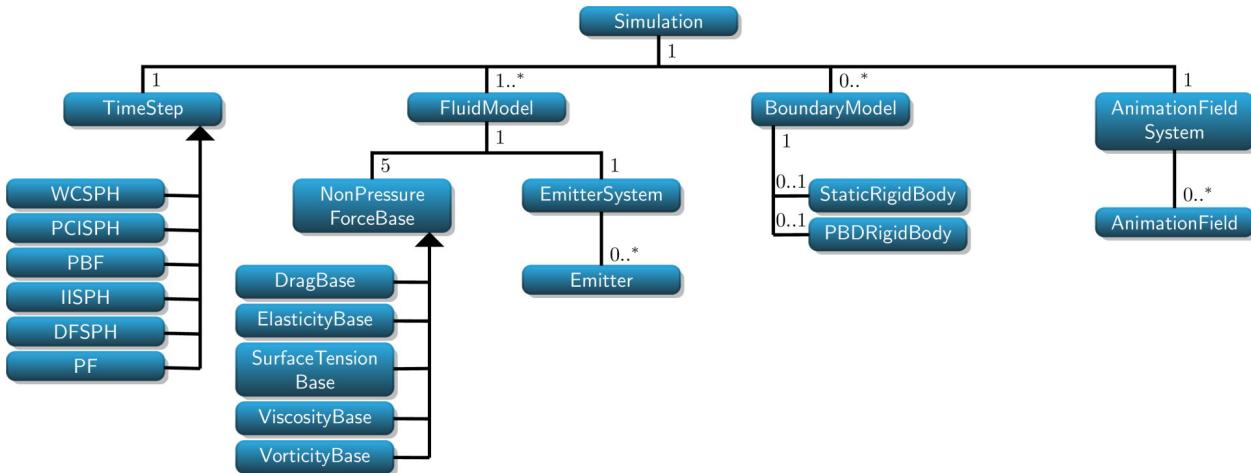
## **8.6 USE\_PYTHON\_BINDINGS**

Generate a shared library object which can be imported into python scripts and exposes C++ functionality to the python interpreter. *Default:On Options:<On|Off>*

## **8.7 USE\_DEBUG\_TOOLS**

Adds a debug tools tab to the graphical user interface which allows to generate additional particle data for debugging. Note that generating the additional data will slightly decrease the performance of the simulation.

## SOFTWARE ARCHITECTURE



SPPlisHSPlasH follows a very intuitive and modular design approach. We want to illustrate part of the software architecture in conjunction with the simplified class diagram above. Note, that this documentation only covers the simulation part of SPPlisHSPlasH. The whole software architecture follows a similar design pattern as the **Model View Controller**.

### 9.1 The Simulation class

The simulation class is the main part of the software. It contains the currently used simulation method (`TimeStep`), all fluids (`FluidModel`), all boundaries (`BoundaryModel`), and a `AnimationFieldSystem`. It is defined as a **singleton**, thus only one simulation instance exists during the runtime. The simulation instance contains:

- exactly one `TimeStep` instance, which defines the simulation loop and contains the pressure solver
- any number of `FluidModel` instances each defining a different fluid phase
- any number of `BoundaryModel` instances representing either dynamic rigid bodies or static boundaries
- exactly one `AnimationFieldSystem` instance which allows to animate particles in a predefined area

The simulation class also implements the following:

- evaluation of the SPH kernel methods
- update of the time step size using a CFL condition
- uniform invocation of all `EmitterSystem` instances
- invocation of `AnimationFieldSystem` instance

- saving & loading the current simulation state

Lastly, the simulation class also contains a well defined interface for the neighborhood search functionalities defined in [CompactNSearch](#) or [cuNSearch](#), which are further needed in the respective algorithm implementations in e.g. the TimeStep or NonPressureForces.

## 9.2 The TimeStep class

The TimeStep class is a **abstract base class** for any subsequent derived simulation method one wants to implement. It implements the required interface for the simulation class, notably the `step()` function containing the simulation algorithm called in the main loop. During execution there exists exactly one instance of a TimeStep class. By default SPlisHSPlasH currently implements the following pressure solvers and the corresponding simulation algorithms:

- WCSPH
- PCISPH
- PBF
- IISPH
- DFSPH
- Projective Fluids

## 9.3 The FluidModel class

A FluidModel instance represents a fluid phase with its respective properties and applied effects to it. SPlisHSPlasH allows for arbitrary many FluidModels inside a simulation as long as there is at least one and they all have a different id (see scene file format). One FluidModel contains the following:

- Physical parameters like rest density, mass, position, velocity, acceleration and current density
- Simulation parameters like the number of particles, their state and ID
- References to the applied non-pressure effects, one for each:
  - Drag
  - Elasticity
  - Surface tension
  - Viscosity
  - Vorticity
- Emitter systems

Concerning the non-pressure effects, each FluidModel can *only* utilize up to one method per non-pressure effect, which will be directly included in the computation inside the `computeNonPressureForces()` method of the Simulation class. Thus having e.g. two different surface tension algorithms inside one FluidModel is **not** possible. However, it is possible to define e.g. two phases, which have a different viscosity model and only one regarding surface tension effects.

The emitters are only stored inside the FluidModels since they are assigned to a fixed FluidModel. Their functionalities are uniformly executed by the Simulation class in the `emitParticles()` step usually invoked at the end of the simulation loop of the current TimeStep instance.

## 9.4 The BoundaryModel class

The `BoundaryModel` class provides a useful base class for any boundary handling methods. It stores a `RigidBodyObject` reference representing the object of the boundary. This can be a stationary or dynamic rigid body, whose coupling effects are handled uniformly. Note that `RigidBodyObject` is an abstract class providing an interface for the two derived classes `StaticRigidBody` and `PBDRigidBody`. The first is handled internally and represent stationary objects. The latter describes a moving rigid body which is simulated externally by the `Position-BasedDynamics` library. SPlisHSPlasH implements three different boundary models:

- Particle-based rigid-fluid coupling [Akinci et al. 2012]
- Density maps [Koschier and Bender 2017]
- Volume maps [Bender et al. 2019]

Finally, SPlisHSPlasH defines a boundary as a list of rigid bodies in conjunction with a rigid-fluid coupling algorithm.



## IMPLEMENTING A NEW NON-PRESSURE FORCE METHOD

Non-pressure forces (e.g. viscosity, vorticity, surface tension or drag forces) are all implemented in the same way in SPLisHSPlasH. In the following we explain the implementation of such a method using as example a new viscosity method.

SPLisHSPlasH organizes the viscosities in /SPLisHSPlasH/Viscosity/ and thus any changes or additions are intended to take place in this directory. The user can add new viscosity methods by creating new or copying and modifying existing viscosity class files and registering these inside the build system and the source code.

### 10.1 Creating a new class

If you want to create a new viscosity class from scratch, you should consider reading the doxygen documentation on the `ViscosityBase` class and several of its derived classes. In short, every viscosity method inherits from the base class `ViscosityBase`, which itself inherits from `NonPressureForceBase`. A minimal working derived class would look like this:

#### MyViscosity.h

```
#ifndef __MyViscosity_h__
#define __MyViscosity_h__

#include "SPLisHSPlasH/Common.h"
#include "SPLisHSPlasH/FluidModel.h"
#include "ViscosityBase.h"

namespace SPH
{
    class MyViscosity : public ViscosityBase
    {
        protected:
            virtual void initParameters();

        public:
            MyViscosity(FluidModel *model);
            virtual ~MyViscosity(void);

            static NonPressureForceBase* creator(FluidModel* model) { return new
                MyViscosity(model); }

            virtual void step();
            virtual void reset();
    };
}
```

(continues on next page)

(continued from previous page)

```
#endif
```

### MyViscosity.cpp

```
#include "MyViscosity.h"

MyViscosity::MyViscosity(FluidModel *model) :
    ViscosityBase(model)
{
    [...]
}

MyViscosity::~MyViscosity(void)
{
    [...]
}

void MyViscosity::initParameters()
{
    ViscosityBase::initParameters();

    [...]
}

void MyViscosity::step()
{
    [...]
}

void MyViscosity::reset()
{
    [...]
}
```

including the following:

- a constructor with `FluidModel*` as the sole parameter `MyViscosity(FluidModel *model)`
- a `initParameters()` method calling the base class method for parameter setup
- a step function `void step()` called in each timestep for the associated fluid
- a reset function `void reset()` called on every reset of the simulation

#### 10.1.1 Customizing your class

##### Neighborhood search sort

The user is also free to add and save additional per particle data inside the viscosity method, but has to ensure that these are also included in the *neighborhood search sort*. Sorting is required if the data is used over multiple simulations steps. The neighborhood search performs a z-sort every n steps to improve the number of cache hits. Since all particles are resorted, also their data must be resorted. For this, the user has to override the `performNeighborhoodSearchSort()` method. A minimal example would look like the following:

```
void MyViscosity::performNeighborhoodSearchSort ()
{
    Simulation *sim = Simulation::getCurrent ();
    auto const& d = sim->getNeighborhoodSearch ()->point_set (m_model-
    →getPointSetIndex ());
    d.sort_field (&m_myParticleViscosityData [0]);
}
```

## Additional particle fields

For visualization and/or debugging purposes, the user may also want to subject the particle data to SPlisHSPlasH's particle informations. To do this, the user has to add the particle data field to the list of fields inside each `FluidModel`. This can be for example done in the constructor by adding the `addField(const FieldDescription &field)` of the corresponding `FluidModel`. The fields can be used to define the color of a particle, they can be exported to bgeo or ParaView and in the simulator the user can output the field data of the selected particles by pressing "i".

For more information, please refer to the doxygen documentation and maybe take a look at the already existing implementations. Adding a field has the following form:

```
model->addField({ "myFieldName", <FieldType>, <lambda expression returning reference_
    →to the data field>}, <save state (boolean)>);
```

Here is an example:

```
model->addField({ "myFieldName", FieldType::Vector3, [&] (const unsigned int i) ->
    →Real* { return &m_myFieldValues[i] [0]; }, true});
```

The field name is used in the GUI and when exporting the data. The boolean at the end determines if this field should be stored when the simulation state is saved. This should only be done if the value is not recomputed in each simulation step so that the value of the last step is required.

Also don't forget to remove the field, when the instance of the viscosity method is destroyed:

```
m_model->removeFiledByName ("myFieldName");
```

## Deferred initialization

The user can override the `deferredInit()` method. This function is called after the simulation scene is loaded and all parameters are initialized. While reading a scene file several parameters can change. The `deferredInit()` function should initialize all values which depend on these parameters.

```
void MyViscosity::deferredInit ()
{
    initMyViscosity ();
}
```

## 10.2 Registering the viscosity method

To add our new viscosity method, we have to integrate it into the build process and the source code.

### 10.2.1 Adding to the build process

Simply add the class files `MyViscosity.h` and `MyViscosity.cpp` to the `CMakeLists.txt` in the `/SPlisHSPlasH/` directory. This can be done by adding the relative file paths to the respective variables `VISCOSITY_HEADER_FILES` and `VISCOSITY_SOURCE_FILES`:

```
set(VISCOSITY_HEADER_FILES
    [...]
    Viscosity/MyViscosity.h
)

set(VISCOSITY_SOURCE_FILES
    [...]
    Viscosity/MyViscosity.cpp
)
```

### 10.2.2 Integration in the source code

Any non-pressure force method is registered in the file `NonPressureForceRegistration.cpp`, which can be found in the `/SPlisHSPlasH/` directory. Adding our new viscosity method is done by adding the following line to the function `void Simulation::registerNonpressureForces()`:

```
addViscosityMethod("My viscosity method", MyViscosity::creator);
```

and including `Viscosity/MyViscosity.h`.

After these additions and building `SPlisHSPlasH`, our new viscosity method is available inside the simulation.

## IMPLEMENTING A NEW PARTICLE/RIGID BODY DATA EXPORTER

All exporters are implemented in the same way in SPlisHSPlasH. In the following we explain the implementation of such an exporter method using as example a new rigid body exporter.

SPlisHSPlasH organizes the exporters in `/Simulator/Exporter/` and thus any changes or additions are intended to take place in this directory. The user can add new data exporters by creating new or copying and modifying existing exporter class files and registering these inside the build system and the source code.

### 11.1 Creating a new class

If you want to create a new exporter class from scratch, you should take a look at existing exporters in SPlisHSPlasH. In short, every exporter inherits from the base class `ExporterBase`. A minimal working derived class would look like this:

#### RigidBodyExporter\_MyFormat.h

```
#ifndef __RigidBodyExporter_MyFormat_h__
#define __RigidBodyExporter_MyFormat_h__

#include "ExporterBase.h"

namespace SPH
{
    /** \brief Rigid body exporter for the OBJ format.
     */
    class RigidBodyExporter_MyFormat : public ExporterBase
    {
    protected:
        bool m_isFirstFrame;
        std::string m_exportPath;

    public:
        RigidBodyExporter_MyFormat(SimulatorBase* base);
        RigidBodyExporter_MyFormat(const RigidBodyExporter_MyFormat&) = delete;
        RigidBodyExporter_MyFormat& operator=(const RigidBodyExporter_MyFormat&) = delete;
        virtual ~RigidBodyExporter_MyFormat(void);

        virtual void init(const std::string& outputPath);
        virtual void step(const unsigned int frame);
        virtual void reset();
        virtual void setActive(const bool active);
```

(continues on next page)

(continued from previous page)

```

    };
}

#endif

```

**RigidBodyExporter\_MyFormat.cpp**

```

#include "RigidBodyExporter_MyFormat.h"
#include <Utilities/Logger.h>
#include <Utilities/FileSystem.h>
#include "SPlisHSPlasH/Simulation.h"

using namespace SPH;
using namespace Utilities;

RigidBodyExporter_MyFormat::RigidBodyExporter_MyFormat(SimulatorBase* base) :
    ExporterBase(base)
{
    m_isFirstFrame = true;
}

RigidBodyExporter_MyFormat::~RigidBodyExporter_MyFormat(void)
{
}

void RigidBodyExporter_MyFormat::init(const std::string& outputPath)
{
    // define output path for the data
    m_exportPath = FileSystem::normalizePath(outputPath + "/my_format");
}

void RigidBodyExporter_MyFormat::step(const unsigned int frame)
{
    // check if the exporter is active
    if (!m_active)
        return;

    // check if we have a static model
    bool isStatic = true;
    for (unsigned int i = 0; i < sim->numberOfBoundaryModels(); i++)
    {
        BoundaryModel* bm = sim->getBoundaryModel(i);
        if (bm->getRigidBodyObject()->isDynamic())
        {
            isStatic = false;
            break;
        }
    }

    // If we have a static model, write the data only for the first frame.
    // Otherwise each frame is exported.
    if (m_isFirstFrame || !isStatic)
    {
        [...]
    }
    m_isFirstFrame = false;
}

```

(continues on next page)

(continued from previous page)

```

void RigidBodyExporter_MyFormat::reset()
{
    m_isFirstFrame = true;
}

void RigidBodyExporter_MyFormat::setActive(const bool active)
{
    ExporterBase::setActive(active);
    // create output folder
    if (m_active)
        FileSystem::makeDirs(m_exportPath);
}

```

including the following:

- a constructor with `SimulatorBase*` as the sole parameter `RigidBodyExporter_MyFormat (SimulatorBase* base)`
- a `init(const std::string& outputPath)` method which should define the export path, `outputPath` contains the path of the current output directory of SPlisHSPlasH
- a step function `void step()` called for each frame that should be exported
- a reset function `void reset()` called on every reset of the simulation
- a function `void setActive(const bool active)` which is called when the user activates the exporter

In our example the exporter path is defined int the function `init`. When the user activates the exporter, e.g. in the GUI, the corresponding directory is created. In the function `step` the rigid body data can be written. Note that we added some code so that static rigid bodies are only exported once and not for each frame since they never change.

## 11.2 Registering the exporter

To add our new exporter, we have to integrate it into the build process and the source code.

### 11.2.1 Adding to the build process

Simply add the class files `RigidBodyExporter_MyFormat.h` and `RigidBodyExporter_MyFormat.cpp` to the `CMakeLists.txt` in the `/Simulator/` directory. This can be done by adding the relative file paths to the respective variables `EXPORTER_HEADER_FILES` and `EXPORTER_SOURCE_FILES`:

```

set(EXPORTER_HEADER_FILES
    [...]
    Exporter/RigidBodyExporter_MyFormat.h
)

set(EXPORTER_SOURCE_FILES
    [...]
    Exporter/RigidBodyExporter_MyFormat.cpp
)

```

### 11.2.2 Integration in the source code

Any exporter is registered in the file ExporterRegistration.cpp, which can be found in the / Simulator/ directory. Adding our new exporter is done by adding the following line to the function void SimulatorBase::createExporters():

```
addRigidBodyExporter("enableRigidBodyMyFormatExport", "Rigid Body MyFormat Exporter",
    "Enable/disable rigid body My Format export.", new RigidBodyExporter_
    MyFormat(this));
```

and including Exporter/RigidBodyExporter\_MyFormat.h. The first string defines a key which can be used in the json scene files to activate your exporter. The second string defines the name of your exporter which will appear in the GUI. This name can also be used to activate your exporter in C++ or Python. The last string contains a description of the exporter which is used as tool tip in the GUI.

After these additions and building SPlisHSPlasH, our new exporter is available inside the simulation.

## 11.3 Implementing a new exporter in Python

You can also implement a new exporter using our Python interface. You can find an example here: pySPlisHSPlasH\examples\custom\_exporter.py.

## CREATING PRESSURE SOLVERS

SPlisHSPlasH organizes the pressure solvers in their respective folders inside the `/SPlisHSPlasH/` directory. For example DFSPH can be found inside `/SPlisHSPlasH/DFSPH/`. We highly suggest the user to follow our file organization scheme. The user can also add new pressure solvers by creating new or copying and modifying existing classes and then adding them to the build system plus additionally registering in the source code.

Note that we do not strictly distinguish the pressure solver from the simulation algorithm. Each `TimeStep` class implements a whole time step including the pressure solver. The non-pressure forces are decoupled in their respective classes and only implicitly called. Thus for implementing a new pressure solver, we suggest copying the files from for example WCSPH and replacing the pressure solver by your own one. Note further, that we usually decouple data from the algorithm with the `SimulationData` classes. We strongly recommend doing the same with your implementation.

### 12.1 Creating a new class

Again, we want to stress that copying and modifying existent methods is easier than writing a new class from scratch. However, if you want to do so, be sure to implement every abstract method inherited from `TimeStep`. These include:

- `void step()`, the simulation step function
- `void resize()`, a method to initialize and resize any used field

Albeit being not necessary, the user may also want to override/redefine the following methods:

- `void init()`, the initialization method. It is **important to call** `TimeStep::init()` inside this method
- `void reset()`, the method invoked on every reset command
- `void computeDensities()`, if the user does not want to utilize the given density computation

A minimal working example of a derive class is shown below:

#### `TimeStepMyPressureSolver.h`

```
#ifndef __TimeStepMyPressureSolver_h__
#define __TimeStepMyPressureSolver_h__

#include "SPlisHSPlasH/Common.h"
#include "SPlisHSPlasH/TimeStep.h"
#include "SPlisHSPlasH/SPHKernels.h"

namespace SPH
{
    class TimeStepMyPressureSolver : public TimeStep
    {
```

(continues on next page)

(continued from previous page)

```

public:
    TimeStepMyPressureSolver();
    virtual ~TimeStepMyPressureSolver();

    virtual void step();
    virtual void resize();
};

}

#endif

```

**TimeStepMyPressureSolve.cpp**

```

#include "TimeStepMyPressureSolve.h"

using namespace SPH;
using namespace GenParam;

TimeStepMyPressureSolve::TimeStepMyPressureSolve() :
    TimeStep()
{
    [...]
}

TimeStepMyPressureSolve::~TimeStepMyPressureSolve(void)
{
    [...]
}

void TimeStepMyPressureSolve::step()
{
    [...]
}

void TimeStepMyPressureSolve::resize()
{
    [...]
}

```

SPlisHSPlasH assumes your simulation method allows for operator splitting, thus usually dividing the simulation into non-pressure forces and the pressure solver plus advection. The latter is subject of the TimeStep class. It is still possible to implement these together inside your own TimeStep class, but it contradicts SPlisHSPlasH's design principles. Since the `step()` method is forwarded to the main loop by the simulation class, its purpose is to define the simulation algorithm. For guidance, we also provide a simple SPH simulation algorithm outline:

```

void TimeStepWCSPH::step()
{
    Simulation *sim = Simulation::getCurrent();
    const unsigned int nModels = sim->numberOfFluidModels();
    TimeManager *tm = TimeManager::getCurrent();
    const Real h = tm->getTimeStepSize();

    // 1. Perform a neighborhood search
    performNeighborhoodSearch();
}

```

(continues on next page)

(continued from previous page)

```

// 2. Compute non-pressure forces and SPH densities
    for (unsigned int fluidModelIndex = 0; fluidModelIndex < nModels; _
→fluidModelIndex++)
    {
        clearAccelerations(fluidModelIndex);
        computeDensities(fluidModelIndex);
    }
    sim->computeNonPressureForces();

// 3. Compute pressure forces
    computePressureForces();

// 4. Update time step size with CFL condition
    sim->updateTimeStepSize();

// 5. Advect particles
    advectParticles();

// 6. Emit and/or animate particles if necessary
    sim->emitParticles();
    sim->animateParticles();

// 7. Advect time
    tm->setTime(tm->getTime() + h);
}

```

where `computeDensities(...)` and `clearAcceleration(...)` are already defined by the base class.

We recommend the user to split the simulation algorithm and its data into two separate classes as it is the case for our already implemented ones.

## 12.2 Registering the pressure solver

To add our new simulation method, we have to integrate it into the build process and the source code.

### 12.2.1 Adding to the build process

Simply add all of your class files to the `CMakeLists.txt` in the `/SPlisHSPlasH/` directory. We suggest creating new variables for the header and source files and adding these to the `add_library()` as well as to new `source_group()` calls. A possible implementation following our class file conventions would look like the following:

```

set(MYPRESSURESOLVER_HEADER_FILES
    MyPressureSolver/SimulationDataMyPressureSolver.h
    MyPressureSolver/TimeStepMyPressureSolver.h
)

set(MYPRESSURESOLVER_SOURCE_FILES
    MyPressureSolver/SimulationDataMyPressureSolver.cpp
    MyPressureSolver/TimeStepMyPressureSolver.cpp
)

add_library(SPlisHSPlasH

```

(continues on next page)

(continued from previous page)

```
[...]
${MYPRESSURESOLVER_HEADER_FILES}
${MYPRESSURESOLVER_SOURCE_FILES}
)

source_group("Header Files\\MyPressureSolver" FILES ${MYPRESSURESOLVER_HEADER_FILES})
source_group("Source Files\\MyPressureSolver" FILES ${MYPRESSURESOLVER_SOURCE_FILES})
```

## 12.2.2 Integration in the source code

Any timestep method and thus any pressure solver is registered in the `Simulation.h` and `Simulation.cpp` files, which can be found in the `/SPlisHSPlasH/` directory. Adding a new method comprises of the following steps:

- Adding a new enum in `SimulationMethods`
- Creating a new static variable `static int ENUM_SIMULATION_MYPRESSURESOLVER` for the Generic-Parameter system and initializing it in `Simulation.cpp`
- Including `SPlisHSPlasH/MyPressureSolver/TimeStepMyPressureSolver.h` in `Simulation.cpp`
- Adding a new enum value for `SIMULATION_METHOD` inside `Simulation::initParameters()` using the following line:

```
enumParam->addEnumValue("MyPressureSolverName", ENUM_SIMULATION_MYPRESSURESOLVER);
```

- Adding the pressure solver to `Simulation::setSimulationMethod(...)`, thus making it available for the simulation using the following:

```
else if (method == SimulationMethods::MyPressureSolver)
{
    m_timeStep = new TimeStepMyPressureSolver();
    m_timeStep->init();
    setValue(Simulation::KERNEL_METHOD, <desired standard SPH kernel>);
    setValue(Simulation::GRAD_KERNEL_METHOD, <desired standard SPH gradient kernel>);
}
```

After these additions and building SPlisHSPlasH, our new pressure solver is available inside the simulation.

## MACROS

SPlisHSPlasH defines useful macros to e.g. iterate over all neighboring particles inside the neighborhood of the current one. These can be found in `Simulation.h`. In the following, we want to give a short overview over these macros. For further information, please refer to the api documentation.

### 13.1 Looping over fluid neighbors

An essential part of SPH computation is to use the properties of neighboring particles to compute the desired value. SPlisHSPlasH provides macros iterating over every fluid neighbor, which can be used like predefined for-loop constructs. These include the following:

#### 13.1.1 forall\_fluid\_neigbors

```
#define forall_fluid_neighbors(code) \
    for (unsigned int pid = 0; pid < nFluids; pid++) \
    { \
        FluidModel *fm_neighbor = sim->getFluidModelFromPointSet(pid); \
        for (unsigned int j = 0; j < sim->numberOfNeighbors(fluidModelIndex, \
→pid, i); j++) \
        { \
            const unsigned int neighborIndex = sim-> \
→getNeighbor(fluidModelIndex, pid, i, j); \
            const Vector3r &xj = fm_neighbor->getPosition(neighborIndex); \
→ \
            code \
        } \
    }
```

`forall_fluid_neigbors` loops over every fluid particle (in all fluid phases) in the neighborhood region of the current one. Note that this does **not** include boundary particles. The user can use this macro by writing the desired code inside the brackets. For the usage of most of the macros, some additional variables have to be predefined. These include in this case:

- `Simulation *sim = Simulation::getCurrent()`, the current simulation instance
- `unsigned int nFluids`, the amount of `FluidModel` instances
- `unsigned int fluidModelIndex`, the index of the `FluidModel` of the current particle
- `unsigned int i`, the index of the current particle inside the `FluidModel` with index `fluidModelIndex`

Further, this macro also defines certain variables, which can be accessed inside the code given to the macro:

- `unsigned int pid`, the index of the FluidModel of the neighboring particle
- `FluidModel *fm_neighbor`, the FluidModel reference of the neighboring particle
- `const unsigned int neighborIndex`, the particle index of the neighboring particle
- `const Vector3r &xj`, the position of the neighboring particle

Henceforth, we denote the required additional variables by **Requires** and the by the macro defined ones by **Defines**.

### 13.1.2 forall\_fluid\_neighbors\_in\_same\_phase

```
#define forall_fluid_neighbors_in_same_phase(code) \
    for (unsigned int j = 0; j < sim->numberOfNeighbors(fluidModelIndex, \
    fluidModelIndex, i); j++) \
    { \
        const unsigned int neighborIndex = sim->getNeighbor(fluidModelIndex, \
        fluidModelIndex, i, j); \
        const Vector3r &xj = model->getPosition(neighborIndex); \
        code \
    }
```

`forall_fluid_neighbors_in_same_phase` loops over every fluid particle in the neighborhood region considering only neighbors from the **same** FluidModel as the current one.

- **Requires:**

- `Simulation *sim = Simulation::getCurrent()`
- `unsigned int fluidModelIndex`
- `unsigned int i`

- **Defines:**

- `const unsigned int neighborIndex`
- `const Vector3r &xj`

## 13.2 Looping over boundaries

### 13.2.1 forall\_boundary\_neighbors

```
#define forall_boundary_neighbors(code) \
    for (unsigned int pid = nFluids; pid < sim->numberOfPointSets(); pid++) \
    { \
        BoundaryModel_Akinci2012 *bm_neighbor = static_cast<BoundaryModel_ \
        Akinci2012*>(sim->getBoundaryModelFromPointSet(pid)); \
        for (unsigned int j = 0; j < sim->numberOfNeighbors(fluidModelIndex, pid, \
        i); j++) \
        { \
            const unsigned int neighborIndex = sim-> \
            getNeighbor(fluidModelIndex, pid, i, j); \
            const Vector3r &xj = bm_neighbor->getPosition(neighborIndex); \
            code \
        } \
    }
```

`forall_boundary_neighbors` loops over all boundary neighbors casting them to the Akinci 2012 boundary model.

- **Requires:**

- `Simulation *sim = Simulation::getCurrent()`
- `unsigned int nFluids`
- `unsigned int fluidModelIndex`
- `unsigned int i`

- **Defines:**

- `unsigned int pid`, the index of the FluidModel associated with the BoundaryModel
- `BoundaryModel_Akinci2012 *bm_neighbor`, the BoundaryModel reference of the neighboring particle
- `const unsigned int neighborIndex`, the particle index of the neighboring particle
- `const Vector3r &xj`, the position of the neigboring particle

### 13.2.2 `forall_density_maps`

```
#define forall_density_maps(code) \
for (unsigned int pid = 0; pid < nBoundaries; pid++) \
{ \
    BoundaryModel_Koschier2017 *bm_neighbor = static_cast<BoundaryModel_ \
→Koschier2017*>(sim->getBoundaryModel(pid)); \
    const Real rho = bm_neighbor->getBoundaryDensity(fluidModelIndex, i); \
    if (rho != 0.0) \
    { \
        const Vector3r &gradRho = bm_neighbor-> \
→getBoundaryDensityGradient(fluidModelIndex, i).cast<Real>(); \
        const Vector3r &xj = bm_neighbor->getBoundaryXj(fluidModelIndex, i); \
        code \
    } \
}
```

`forall_density_maps` loops over all boundary neighbors casting them to the Koschier 2017 boundary model.

- **Requires:**

- `Simulation *sim = Simulation::getCurrent()`
- `unsigned int nBoundaries`
- `unsigned int fluidModelIndex`
- `unsigned int i`

- **Defines:**

- `unsigned int pid`
- `BoundaryModel_Koschier2017 *bm_neighbor`
- `const Real rho`, the boundary density given by the density map
- `const Vector3r &gradRho`, the boundary density gradient
- `const Vector3r &xj`

### 13.2.3 forall\_volume\_maps

```
#define forall_volume_maps(code) \
    for (unsigned int pid = 0; pid < nBoundaries; pid++) \
    { \
        BoundaryModel_Bender2019 *bm_neighbor = static_cast<BoundaryModel_ \
→Bender2019*>(sim->getBoundaryModel(pid)); \
        const Real Vj = bm_neighbor->getBoundaryVolume(fluidModelIndex, i); \
        if (Vj > 0.0) \
        { \
            const Vector3r &xj = bm_neighbor->getBoundaryXj(fluidModelIndex, \
→i); \
            code \
        } \
    }
```

forall\_volume\_maps loops over all boundary neighbors casting them to the Bender 2019 boundary model.

- **Requires:**

- Simulation \*sim = Simulation::getCurrent()
- unsigned int nBoundaries
- unsigned int fluidModelIndex
- unsigned int i

- **Defines:**

- unsigned int pid
- BoundaryModel\_Koschier2019 \*bm\_neighbor
- const Real Vj, the boundary volume given by the volume map
- const Vector3r &xj

## 13.3 AVX variants

SPlisHSPlasH also defines versions using AVX optimizations for some of the macros. These can be used if the respective CMake option is set in the building process. Note that many of the aforementioned by the macro defined variables are given in AVX compatible data types, if you choose to use the AVX version of these macros.

## PYSPLISHSPLASH

### 14.1 Python bindings for the SPLisHSPlasH library

#### 14.2 Requirements

Currently the generation of python bindings is only tested on

- Linux Debian, gcc 8.3, Python 3.7/3.8 (Anaconda), CMake 3.13
- Windows 10, Visual Studio 15/17/19, Python 3.7/3.8 (Anaconda), CMake 3.13

Note that the compiler, the python installation as well as cmake have to be available from the command line for the installation process to work. MacOS builds should work but have not been tested.

#### 14.3 Installation

In order to install it is advised that you create a new virtual environment so that any faults during installation can not mess up your python installation. This is done as follows for

##### conda

```
conda create --name venv python=3.7
conda activate venv
```

##### virtualenv

```
python3 -m virtualenv venv --python=python3.7
source venv/bin/activate
```

Now you can clone the repository by

```
git clone https://github.com/InteractiveComputerGraphics/SPLisHSPlasH.git
```

And finally you should be able to install SPLisHSPlasH using pip. **The trailing slash is important** otherwise pip will try to download the package, which is not supported yet at least. Also note, that `pip install SPLisHSPlasH` should be called from **one directory above** the cloned source directory and **not within** the directory itself.

```
pip install SPLisHSPlasH/
```

While `pip install` is useful if SPLisHSPlasH should only be installed once, for development purposes it might be more sensible to build differently. Change into the SPLisHSPlasH directory and build a python wheel file as follows

```
cd SPlisHSPlasH
python setup.py bdist_wheel
pip install -I build/dist/*.whl
```

When building a new version of SPlisHSPlasH simply run these commands again and the installation will be updated. The compile times will be lower, because the build files from previous installations remain. If you are getting compile errors please try to compile the pysplishsplash target of the CMake project separately.

Now check your installation by running

```
python -c "import pysplishsplash"
```

**Note:** You may have to install numpy. Future releases may already contain numpy as a dependency.

```
pip install numpy
```

## 14.4 I want to see something very very quickly

If you're very impatient, just run the following command after installing

```
splash
```

You will be prompted to select a preconfigured scene file which will then be run in a User Interface. For more options and functionality run. The keybindings in the GUI are the same as for the regular SPlisHSPlasH version.

```
splash --help
```

## 14.5 Minimal working example

The following examples should work, if SPlisHSPlasH was installed correctly. If you want to load other scene files, be sure to place them into the SPlisHSPlasH data directory structure.

### With GUI

```
import pysplishsplash as sph

def main():
    base = sph.Exec.SimulatorBase()
    base.init()
    gui = sph.GUI.Simulator_GUI_TweakBar(base)
    base.setGui(gui)
    base.run()

if __name__ == "__main__":
    main()
```

### Without GUI

```
import pysplishsplash as sph

def main():
    base = sph.Exec.SimulatorBase()
```

(continues on next page)

(continued from previous page)

```

base.init(useGui=False)
base.setValueFloat(base.STOP_AT, 10.0) # Important to have the dot to denote a_
↪float
base.run()

if __name__ == "__main__":
    main()

```

### Outputting the results to a specific directory without GUI

```

import pysplishsplash as sph
from pysplishsplash.Extras import Scenes
import os

def main():
    base = sph.Exec.SimulatorBase()
    output_dir = os.path.abspath("where/you/want/the/data")
    base.init(useGui=False, outputDir=output_dir, sceneFile=Scenes.DoubleDamBreak)
    base.setValueFloat(base.STOP_AT, 20.0) # Important to have the dot to denote a_
↪float
    base.setValueBool(base.VTK_EXPORT, True)
    # Uncomment the next line to set the output FPS value (must be float)
    # base.setValueFloat(base.DATA_EXPORT_FPS, 10000.)
    base.run()

if __name__ == "__main__":
    main()

```

## 14.6 SPHSimulator.py

If you want to start the simulator in the same way as the C++ version, just use the SPHSimulator.py in the examples directory.

## 14.7 Modifying other properties

The bindings cover most of the public interface of the SPlisHSPlasH library. As such, it is possible to change components of the simulation dynamically. In the following example, the second cube in the well known double dam break scenario is replaced with a slightly larger cube.

```

import pysplishsplash
import pysplishsplash.Utilities.SceneLoaderStructs as Scene

def main():
    base = pysplishsplash.Exec.SimulatorBase()
    args = base.init()
    gui = pysplishsplash.GUI.Simulator_GUI_TweakBar(base)
    base.setGui(gui)
    scene = base.getScene()
    add_block = Scene.FluidBlock('Fluid', Scene.Box([0.0, 0.0, 0.0], [1.0, 1.0, 1.0]),
↪ 0, [0.0, 0.0, 0.0])
    scene.fluidBlocks[1] = add_block # In Place construction not supported yet
    base.run()

```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    main()
```

## EMBEDDED PYTHON

### 15.1 Build with embedded Python support

To enable the embedded Python support just activate the CMake option USE\_EMBEDDED\_PYTHON which is by default turned off. Please ensure that CMake finds the Python interpreter. This can be achieved by setting the PYTHON\_EXECUTABLE to the file path of the python interpreter.

### 15.2 Run simulator with embedded Python support

Make sure that the environment variables PYTHONHOME and PYTHONPATH are set to the directory of your Python installation. Also make sure that the pythonXX.dll (where XX defines the version) is in your path.

When running the simulator with embedded Python support, the new tab ‘Scripts’ should appear in the GUI. Here you can load a script file or reload it.

Alternatively, you can load the Python script directly in a scene. Just use the json key `scriptFile` in the scene file to define the location of the script. If a relative path is used, the simulator assumes that it is relative to the scene file.

### 15.3 Writing a script

First, you have to import the module `splishsplash`, e.g. by: `import splishsplash as sph`

When the script is loaded, the simulator will call the function `init()` automatically. If you defined a function `step()`, it will be called in each simulation step. If you defined a function `reset()`, it will be called when the simulation is reset. Moreover, you can define additional functions that can be called using the GUI.

#### 15.3.1 `init(base)`

If this function is defined, it is called automatically when the script is loaded or reloaded. The parameter `base` contains the current `SimulationBase` object.

### 15.3.2 step()

If this function is defined, it is called automatically in each simulation step.

### 15.3.3 reset()

If this function is defined, it is called automatically in each simulation reset.

### 15.3.4 Additional commands

Additional commands that can be executed via the GUI (buttons will be added) can be defined by a list of strings. The list must be called `function_list` and must contain the names of functions that should be called, e.g.

```
function_list = ['command', 'command2']
```

## 15.4 Example

This is a simple example script which prints the current simulation time, the position of particle 0 and a counter value in each step:

```
import splishsplash as sph
import numpy as np

counter = 0
function_list = ['command', 'command2']

def init(base):
    global counter
    print("init test")
    counter = 1

def step():
    global counter
    sim = sph.Simulation.getCurrent()
    fluid = sim.getFluidModel(0)
    tm = sph.TimeManager.getCurrent()
    print(fluid.getPosition(0))
    print(tm.getTime())
    print(counter)
    counter += 1
    print("----")

def reset():
    print("reset test")

def command():
    print("tst cmd")

def command2():
    print("tst cmd2")
```

## CREATING SCENES

### 16.1 Loading the empty scene

Right now the easiest way to create a custom scene without specifying a `Scene.json` file, is to load the predefined empty scene.

```
import pysplishsplash as sph
import pysplishsplash.Utilities.SceneLoaderStructs as Scenes

base = sph.Exec.SimulatorBase()
base.init(sceneFile=Scenes.Empty)
```

This scene will set the default simulation method to be DFSPH and some other default values, which can all be changed later on.

### 16.2 Recreating the double dam break scenario

In order to recreate the double dam break scenario, we need to add a bounding box as well as two fluid cubes. The bounding box can be added as follows

```
scene = base.getScene()
scene.boundaryModels.append(Scenes.BoundaryData(meshFile="..../models/UnitBox.obj",
    ↪ translation=[0., 3.0, 0.], scale=[4., 6., 4.], color=[0.1, 0.4, 0.5, 1.0],
    ↪ isWall=True, mapInvert=True, mapResolution=[25, 25, 25]))
```

The two fluid blocks can at the end be added using

```
scene.fluidBlocks.append(Scenes.FluidBlock(id='Fluid', box=Scenes.Box([-1.5, 0.0, -1.
    ↪ 5], [-0.5, 2.0, -0.5]), mode=0, initialVelocity=[0.0, 0.0, 0.0]))
scene.fluidBlocks.append(Scenes.FluidBlock(id='Fluid', box=Scenes.Box([0.5, 0.0, 0.5],
    ↪ [1.5, 2.0, 1.5]), mode=0, initialVelocity=[0.0, 0.0, 0.0]))
```

This will recreate a somewhat larger scene than the default double dam break

## 16.3 Putting it all together

The following shows a script detailing how to build and run a custom double dam break. Follow the instruction from before to activate/ deactivate the GUI.

```
import pysplishsplash as sph
import pysplishsplash.Utilities.SceneLoaderStructs as Scenes

def main():
    # Set up the simulator
    base = sph.Exec.SimulatorBase()
    base.init(useGui=True, sceneFile=sph.Extras.Scenes.Empty)

    # Create a tweak bar simulator
    gui = sph.GUI.Simulator_GUI_TweakBar(base)
    base.setGui(gui)

    # Get the scene and add objects
    scene = base.getScene()
    scene.boundaryModels.append(Scenes.BoundaryData(meshFile="../models/UnitBox.obj",
                                                    translation=[0., 3.0, 0.], scale=[4., 6., 4.], color=[0.1, 0.4, 0.5, 1.0],
                                                    isWall=True, mapInvert=True, mapResolution=[25, 25, 25]))
    scene.fluidBlocks.append(Scenes.FluidBlock(id='Fluid', box=Scenes.Box([-1.5, 0.0, -1.5], [-0.5, 2.0, -0.5]), mode=0, initialVelocity=[0.0, 0.0, 0.0]))
    scene.fluidBlocks.append(Scenes.FluidBlock(id='Fluid', box=Scenes.Box([0.5, 0.0, 0.5], [1.5, 2.0, 1.5]), mode=0, initialVelocity=[0.0, 0.0, 0.0]))

    # Run the GUI
    base.run()

if __name__ == "__main__":
    main()
```

## 16.4 Loading a scene from file

Loading a scene from a file is as simple as simply specifying a custom scene file in the init function. This must be an **absolute path!**

```
custom_scene = os.path.abspath("scene.json")
base.init(sceneFile=custom_scene)
```

If you want to use a gui to locate the scene file you may want to use tkinter

```
import tkinter as tk
from tkinter import filedialog

tk.Tk().withdraw() # Dont show main window
custom_scene = filedialog.askopenfilename()
base.init(sceneFile=custom_scene)
```

---

CHAPTER  
**SEVENTEEN**

---

## **RESTRICTIONS**

- When modifying simulation parameters this is the recommended structure, as modification will only work after `base.initSimulation()` has been called.

```
base.initSimulation()
sim = sph.Simulation.getCurrent()
sim.setValue...()
base.runSimulation()
base.cleanup()
```

- `setValue...()` and `getValue...()` functions cannot accept vectors as arguments yet



---

CHAPTER  
**EIGHTEEN**

---

## **FOAMGENERATOR**

The foam generator is a command line tool to generate spray, foam and bubble particles in a postprocessing step which improves the visual realism of the simulation results. It takes a sequences of particle files and generates a sequence of new particles representing spray, foam and air bubbles. These additional particles are advected using the velocity field of the fluid. Below are two examples which were generated using the foam generator tool:



The tool implements the methods of:

- Markus Ihmsen, Nadir Akinci, Gizem Akinci, Matthias Teschner. Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer* 28(6), 2012
- Jan Bender, Dan Koschier, Tassilo Kugelstadt and Marcel Weiler. Turbulent Micropolar SPH Fluids with Foam. *IEEE Transactions on Visualization and Computer Graphics* 25(6), 2019

### **18.1 Parameters for foam generation**

The foam generator first analyzes the complete simulation data before the generation starts. During this analysis the tool determines the maximum values per frame for the potentials as proposed by Bender et al. [2019]. The resulting values are used for an automatic configuration of the parameters `-ta`, `-wc`, `-vo` and the limits. If you want to set these parameters manually (like in the original method of Ihmsen et al. [2012]), then you have to use the command line parameter “`-no-auto`” and set the parameters `-ta`, `-wc`, `-vo` and `-limits`.

## 18.2 Bounding box

Moreover, it is possible to define a bounding box for the foam particles. Foam particles are advected only using the velocity field of the fluid. However, there is no boundary handling since this would be quite expensive. Hence, particles can go through the boundary. A simple solution is to define a bounding box and clamp the particles which leave the box or kill these particles or steal their lifetime.

## 18.3 Frame rate

We recommend to generate the fluid sequence with 50 fps. Therefore, by default the time step size of the generator is set to 0.02s. If you use another frame rate, you have to adapt this parameter.

## 18.4 Further parameters

- The parameter `-buoyancy` defines the buoyancy of air bubbles. Higher values let them go up faster.
- The parameter `-drag` defines the coefficient of a drag force between the fluid particles and the foam particles.
- The parameter `-foamscale` defines how many foam particles are generated per frame.
- The parameter `-lifetime` defines the minimum and maximum lifetime of the foam particles in seconds.
- The parameter `-skipframes` allows you the skip frames when writing the foam data, e.g. if you have a 50fps fluid sequence and want to write a 25fps foam sequence.
- The parameters `-splittypes` and `-splitgenerators` can be used if you want to split the output in spray, foam and air bubble particles.

### 18.4.1 Command line options:

- `-h, -help`: Print help
- `-i, -input arg`: Input file (partio)
- `-o, -output arg`: Output file (partio or vtk)
- `-q, -query`: Query mode: determines max/avg values
- `-no-auto`: Disable automatic mode. Limits and factors ta, wc, vo must be set manually.
- `-splittypes`: Output each foam type to a different file
- `-splitgenerators`: Output different foam files depending on which potential generated the foam. Overrides `-splittypes`.
- `-s, -startframe arg`: Start frame (default: 1)
- `-e, -endframe arg`: End frame
- `-r, -radius arg`: Particle radius (default: 0.025)
- `-t, -timestepsize arg`: Time step size (default: 0.02)
- `-k, -kernel arg`: 0: Cubic spline, 1: Ihmsen et al. 2012 (default: 0)
- `-l, -limits arg`: Limits (min/max) for potentials (trapped air, wave crest, vorticity, kinetic energy) (default: 5,20,2,8,5,20,5,50)

- `-lifetime` arg: Lifetime (min/max) (default: 2.0,5.0)
- `-b`, `-buoyancy` arg: Buoyancy (default: 2.0)
- `-d`, `-drag` arg: Drag (default: 0.8)
- `-ta` arg: Trapped air factor (default: 4000)
- `-wc` arg: Wave crest factor (default: 50000)
- `-vo` arg: Vorticity factor (default: 4000)
- `-bbsize` arg: minimum and maximum coordinates of and axis aligned bounding-box (minX, minY, minZ, maxX, maxY, maxZ)
- `-bbtype` arg: chose how the bounding-box is used [kill | lifesteal | clamp]. Use in combination with `-bbsize`.
- `-skipframes` arg: number of frames to skip when writing foam (default: 0)
- `-f`, `-foamscale` arg: Global multiplier for number of generated foam particles (default: 1000)

#### 18.4.2 Example:

```
FoamGenerator -s 1 -e 500 -r 0.025 --foamscale 1000 -i output\DamBreakModelDragons\  
  partio\ParticleData_Fluid_#.bgeo -o output\DamBreakModelDragons\foam\foam_#.bgeo
```



---

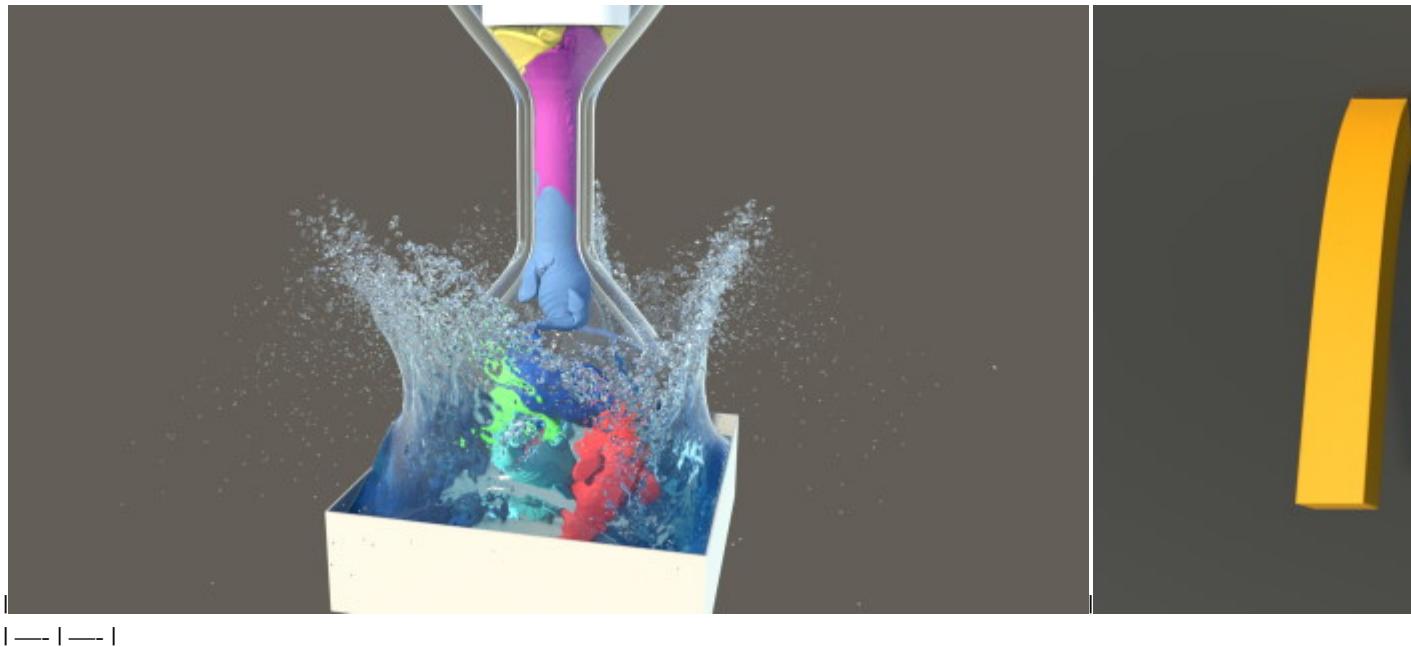
CHAPTER  
NINETEEN

---

## MESHSKINNING

MeshSkinning is a command line tool to generate a sequence of deformed meshes from a sequence of particle files of an elastic model. When simulating an elastic solid using SPlisHSPlasH, we only get particle data. If this data is exported using the PartioExported, the MeshSkinning tool is able to generate deformed triangle meshes in a post-processing step. The tool requires a triangle mesh of the reference configuration of the deformable solid. This mesh is then deformed according to the particle data.

Below are two examples which were generated using the mesh skinning tool:



The tool implements the methods of:

- Tassilo Kugelstadt, Jan Bender, José Antonio Fernández-Fernández, Stefan Rhys Jeske, Fabian Löschner, and Andreas Longva. Fast Corotated Elastic SPH Solids with Implicit Zero-Energy Mode Control. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2021

## 19.1 Parameters

The mesh skinning tool can either use the scene file as input. Then you have to define the required triangle mesh in the scene file for each elastic model:

```
"visMesh": ".../models/beam.obj"
```

All transformations and required info is then automatically extracted from the scene file (assuming that the partio files are located in the output folder) which is the simplest way to use this tool.

Alternatively you can also define everything manually. Then you have to provide the partio path, mesh file as well as translation, rotation and scaling of the mesh file to fit the reference configuration of the particles.

### 19.1.1 Command line options:

- -i, --input arg: Input file
- -o, --output arg: Output file
- -m, --mesh arg: Mesh file
- --scene arg: Scene file (all settings are imported from the scene file)
- --partioPath arg: Path of the partio files (when using a scene file). If not set, it is assumed that the files are in the standard output path.
- --scale arg: Scaling of input geometry (e.g. --scale "2 1 2") (default: 1 1 1)
- -t, --translation arg: Translation of input geometry (e.g. --translation "2 1 2") (default: 1 1 1)
- --axis arg: Rotation axis of input geometry (e.g. --axis "1 0 0") (default: 1 0 0)
- --angle arg: Angle of input geometry (e.g. --angle 1) (default: 0.0)
- -s, --startframe arg: Start frame (default: 1)
- -e, --endframe arg: End frame
- -r, --radius arg: Particle radius (default: 0.025)
- --supportRadiusFactor arg: The support radius is defined as factor\*particleRadius (default: 6.0)
- --maxNeighbors arg: The maximum number of neighbors that are used for the interpolation. (default: 60)
- --splitting: Read a scene which used the object splitting export option.
- --overwrite: Overwrite existing files.
- -h, --help: Print help

### 19.1.2 Example:

```
MeshSkinning --splitting --overwrite --scene ..\data\Scenes\Beam.json
```

---

**CHAPTER  
TWENTY**

---

**PARTIO2VTK**

A tool to convert partion files in vtk files. In this way the particle data which is exported from SPlisHSPlasH can be converted to the vtk format. This is useful to import the data in ParaView for visualization.



---

CHAPTER  
**TWENTYONE**

---

**PARTIOVIEWER**

The simulators can export the particle simulation data using the partio file format. The PartioViewer can read such a file and render the particle data using OpenGL. This tool is able to handle multiphase data and rigid body data. It can create image sequences and movies (using ffmpeg).

To visualize a sequence of partio files or a single file, call (the index in the file name is used for the sequence):

```
PartioViewer fluid_data_1.bgeo
```

This tool is also able to read a complete output directory:

```
PartioViewer output/DamBreakModel
```

In this case the tool searches for the partio files of multiple phases in the subdirectory “partio” and for rigid body data in “rigid\_bodies”.

Note: To generate videos you must tell PartioViewer where it can find the ffmpeg executable.

## 21.1 Command line options:

- -h, --help: Print help
- --renderSequence: Render a sequence from startFrame to endFrame as jpeg.
- --renderVideo: Render a sequence from startFrame to endFrame as video. This function requires ffmpeg which must be in the PATH or the ffmpegPath parameter must be set.
- --noOverwrite: Do not overwrite existing frames when using --renderSequence option. Existing frames are not loaded at all which accelerates the image sequence generation.
- -o, --outdir arg: Output directory for images
- --rbData arg: Rigid body data to visualize (bin file)
- --ffmpegPath arg: Path of the ffmpeg executable.
- --width arg: Width of the image in pixels. (default: 1024)
- --height arg: Height of the image in pixels. (default: 768)
- --fps arg: Frame rate of video. (default: 25)
- -r, --radius arg: Particle radius (default: 0.025)
- -s, --startFrame arg: Start frame (only used if value is >= 0) (default: -1)
- -e, --endFrame arg: End frame (only used if value is >= 0) (default: -1)

- `--colorField arg`: Name of field that is used for the color. (default: velocity)
- `--colorMapType arg`: Color map (0=None, 1=Jet, 2=Plasma) (default: 1)
- `--renderMinValue arg`: Min value of field. (default: 0.0)
- `--renderMaxValue arg`: Max value of field. (default: 10.0)
- `--camPos arg`: Camera position (e.g. `--camPos "0 1 5"`) (default: 0 3 10)
- `--camLookat arg`: Camera lookat (e.g. `--camLookat "0 0 0"`) (default: 0 0 0)

## 21.2 Hotkeys

- Space: pause/continue simulation
- r: reset simulation
- w: wireframe rendering of meshes
- i: print all field information of the selected particles to the console
- s: save current frame as jpg image
- v: generate video
- j: generate image sequence
- +: step to next frame
- -: step to previous frame
- ESC: exit

---

**CHAPTER**  
**TWENTYTWO**

---

## **SURFACESAMPLING**

A popular boundary handling method which is also implemented in SPlisHSPlasH uses a particle sampling of the surfaces of all boundary objects. This command line tool can generate such a surface sampling. Note that the same surface sampling is also integrated in the simulators and the samplings are generated automatically if they are required. However, if you want to generate a surface sampling manually, then you can use this tool.



---

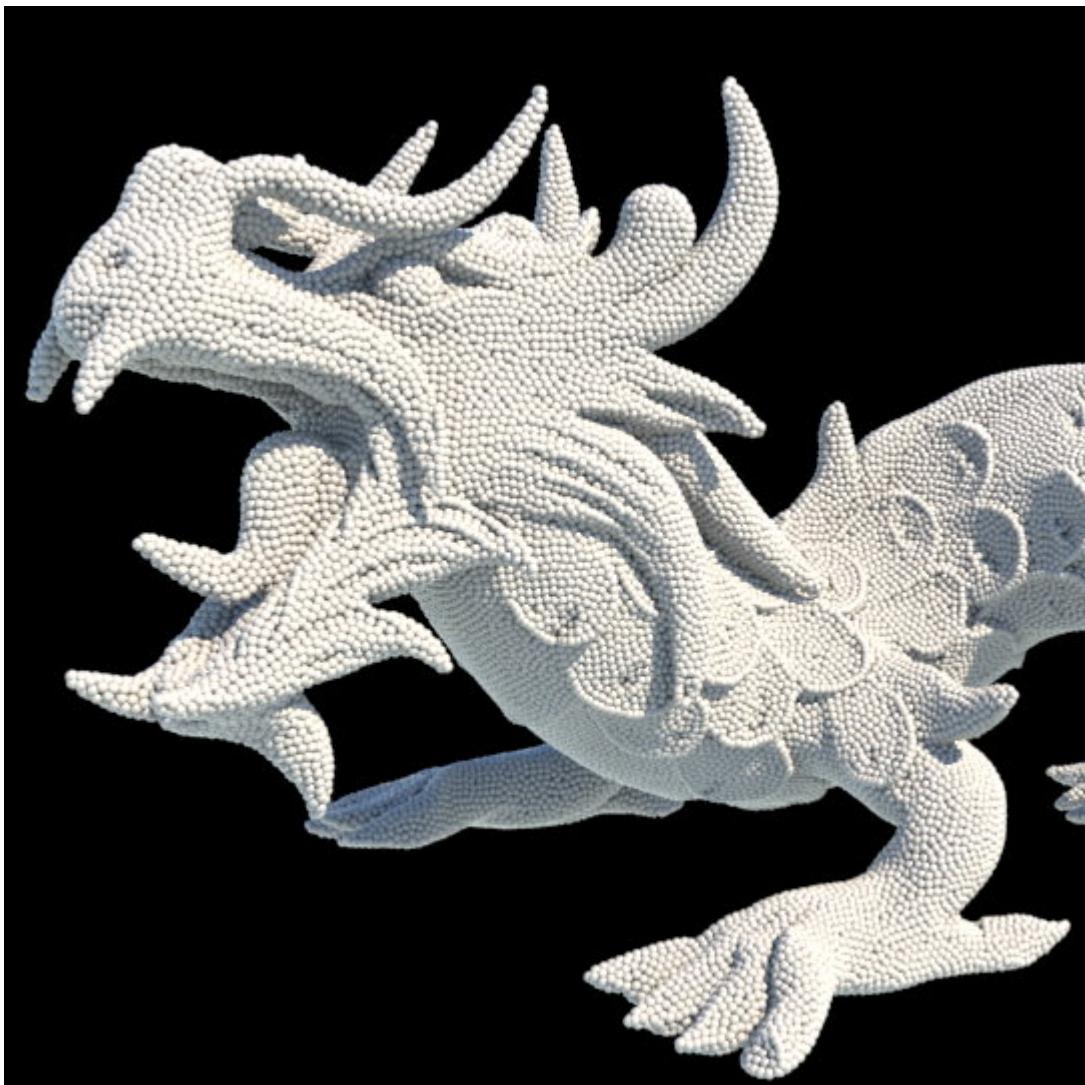
CHAPTER  
**TWENTYTHREE**

---

## **VOLUMESAMPLING**

The simulator can load particle data from partio files. This particle data then defines the initial configuration of the particles in the simulation. The VolumeSampling tool allows you to sample a volumetric object with particle data. This means you can load an OBJ file with a closed surface geometry and sample the interior with particles using different methods. Especially when simulating elastic solids a good sampling is beneficial as shown by Kugelstadt et al. [2021].

Below are two examples which were generated using the volume sampling tool:





||—————|—————|—————|

The tool implements the methods of:

- M. Jiang, Y. Zhou, R. Wang, R. Southern, J. J. Zhang. Blue noise sampling using an SPH-based method. ACM Transactions on Graphics, 2015
- Tassilo Kugelstadt, Jan Bender, José Antonio Fernández-Fernández, Stefan Rhys Jeske, Fabian Lüscher, and Andreas Longva. Fast Corotated Elastic SPH Solids with Implicit Zero-Energy Mode Control. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2021

## 23.1 Command line options:

- -h, --help: Print help
- -i, --input arg: Input file (obj)
- -o, --output arg: Output file (bgeo or vtk)
- -r, --radius arg: Particle radius (default: 0.025)
- -s, --scale arg: Scaling of input geometry (e.g. --scale "2 1 2") (default: 1 1 1)
- -m, --mode arg: Mode (regular=0, almost dense=1, dense=2, Jiang et al. 2015=3, Kugelstadt et al. 2021=4) (default: 4)
- --region arg: Region to fill with particles (e.g. --region "0 0 0 1 1 1")
- --steps arg: SPH time steps (default: 100)
- --cflFactor arg: CFL factor (default: 0.25)
- --viscosity arg: Viscosity coefficient (XSPH) (default: 0.25)
- --cohesion arg: Cohesion coefficient
- --adhesion arg: Adhesion coefficient
- --stiffness arg: Stiffness coefficient (only mode 3) (default: 10000.0)
- --dt arg: Time step size (only mode 3) (default: 0.0005)
- --res arg: Resolution of the Signed Distance Field (e.g. --res "30 30 30")
- --invert: Invert the SDF to sample the outside of the object in the bounding box/region
- --no-cache: Disable caching of SDF.

## 23.2 Example:

```
VolumeSampling.exe --mode 4 -i ..\data\models\bunny.obj -o bunny.vtk
```



---

CHAPTER  
**TWENTYFOUR**

---

**LIBRARY API**

## 24.1 Full API

### 24.1.1 Namespaces

**Namespace @63**

**Namespace chrono**

**Namespace Eigen**

**Contents**

- *Namespaces*

**Namespaces**

- *Namespace Eigen::internal*

**Namespace Eigen::internal**

**Contents**

- *Classes*

## Classes

- *Template Struct generic\_product\_impl< MatrixReplacement, Rhs, SparseShape, DenseShape, GemvProduct >*
- *Template Struct traits< SPH::MatrixReplacement >*

## Namespace GenParam

## Namespace SPH

### Contents

- *Classes*
- *Enums*
- *Variables*

## Classes

- *Struct Elasticity\_Kugelstadt2021::ElasticObject*
- *Struct Elasticity\_Kugelstadt2021::Factorization*
- *Struct FieldDescription*
- *Struct PoissonDiskSampling::CellPosHasher*
- *Struct PoissonDiskSampling::HashEntry*
- *Struct PoissonDiskSampling::InitialPointInfo*
- *Struct Simulation::FluidInfo*
- *Struct Simulation::NonPressureForceMethod*
- *Class AdhesionKernel*
- *Class AnimationField*
- *Class AnimationFieldSystem*
- *Class BinaryFileReader*
- *Class BinaryFileWriter*
- *Class BlockJacobiPreconditioner3D*
- *Class BoundaryModel*
- *Class BoundaryModel\_Akinci2012*
- *Class BoundaryModel\_Bender2019*
- *Class BoundaryModel\_Koschier2017*
- *Class CholeskyAVXSolver*
- *Class CholeskySparseMatrixfAVX*
- *Class CohesionKernel*

- *Class CubicKernel*
- *Class CubicKernel2D*
- *Class DebugTools*
- *Class DragBase*
- *Class DragForce\_Gissler2017*
- *Class DragForce\_Macklin2014*
- *Class Elasticity\_Becker2009*
- *Class Elasticity\_Kugelstadt2021*
- *Class Elasticity\_Peer2018*
- *Class ElasticityBase*
- *Class Emitter*
- *Class EmitterSystem*
- *Class FluidModel*
- *Class GaussQuadrature*
- *Class JacobiPreconditioner1D*
- *Class JacobiPreconditioner3D*
- *Class MathFunctions*
- *Class MatrixReplacement*
- *Class MicropolarModel\_Bender2017*
- *Class NonPressureForceBase*
- *Class PoissonDiskSampling*
- *Class Poly6Kernel*
- *Template Class PrecomputedKernel*
- *Class RegularSampling2D*
- *Class RegularTriangleSampling*
- *Class RigidBodyObject*
- *Class SimpleQuadrature*
- *Class Simulation*
- *Class SimulationDataDFSPH*
- *Class SimulationDataICSPH*
- *Class SimulationDataIISPH*
- *Class SimulationDataPBF*
- *Class SimulationDataPCISPH*
- *Class SimulationDataPF*
- *Class SimulationDataWCSPH*
- *Class SpikyKernel*

- *Class StaticRigidBody*
- *Class SurfaceTension\_Akinci2013*
- *Class SurfaceTension\_Becker2007*
- *Class SurfaceTension\_He2014*
- *Class SurfaceTension\_ZorillaRitter2020*
- *Class SurfaceTensionBase*
- *Class TimeIntegration*
- *Class TimeManager*
- *Class TimeStep*
- *Class TimeStepDFSPH*
- *Class TimeStepICSPH*
- *Class TimeStepIISPH*
- *Class TimeStepPBF*
- *Class TimeStepPCISPH*
- *Class TimeStepPF*
- *Class TimeStepWCSPH*
- *Class TriangleMesh*
- *Class Viscosity\_Bender2017*
- *Class Viscosity\_Peer2015*
- *Class Viscosity\_Peer2016*
- *Class Viscosity\_Standard*
- *Class Viscosity\_Takahashi2015*
- *Class Viscosity\_Weiler2018*
- *Class Viscosity\_XSPH*
- *Class ViscosityBase*
- *Class VorticityBase*
- *Class VorticityConfinement*
- *Class WendlandQuinticC2Kernel*
- *Class WendlandQuinticC2Kernel2D*

## Enums

- *Enum BoundaryHandlingMethods*
- *Enum FieldType*
- *Enum ParticleState*
- *Enum SimulationMethods*
- *Enum SurfaceSamplingMode*

## Variables

- *Variable SPH::gaussian\_abscissae\_1*
- *Variable SPH::gaussian\_n\_1*
- *Variable SPH::gaussian\_weights\_1*

## Namespace std

### Namespace Utilities

#### Contents

- *Classes*
- *Enums*
- *Variables*

## Classes

- *Struct AverageCount*
- *Struct AverageTime*
- *Struct MeshFaceIndices*
- *Struct SceneLoader::AnimationFieldData*
- *Struct SceneLoader::BoundaryData*
- *Struct SceneLoader::Box*
- *Struct SceneLoader::EmitterData*
- *Struct SceneLoader::FluidBlock*
- *Struct SceneLoader::FluidData*
- *Struct SceneLoader::MaterialData*
- *Struct SceneLoader::Scene*
- *Struct TimingHelper*
- *Class ConsoleSink*

- *Class Counting*
- *Class FileSink*
- *Class FileSystem*
- *Class IDFactory*
- *Class Logger*
- *Class LogSink*
- *Class LogStream*
- *Class OBJLoader*
- *Class PartioReaderWriter*
- *Class SceneLoader*
- *Class SDFFunctions*
- *Class StringTools*
- *Class SystemInfo*
- *Class Timing*
- *Class VolumeSampling*
- *Class WindingNumbers*

## Enums

- *Enum LogLevel*

## Variables

- *Variable Utilities::logger*

### 24.1.2 Classes and Structs

#### Template Struct AlignmentAllocator::rebind

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

#### Nested Relationships

This struct is a nested type of *Template Class AlignmentAllocator*.

## Struct Documentation

```
template<typename T2>
struct AlignmentAllocator::rebind
```

### Public Types

```
typedef AlignmentAllocator<T2, N> other
```

**Template Struct generic\_product\_impl< MatrixReplacement, Rhs, SparseShape, DenseShape, GemvProduct >**

- Defined in file\_SPlisHSPlasH\_Utilsities\_MatrixFreeSolver.h

## Inheritance Relationships

### Base Type

- public generic\_product\_base< MatrixReplacement, Rhs, generic\_product< MatrixReplacement, Rhs > >

## Struct Documentation

```
template<typename Rhs>
struct Eigen::internal::generic_product_impl<MatrixReplacement, Rhs, SparseShape, DenseShape, GemvProduct>
Implementation of the matrix-free matrix vector product
```

### Public Types

```
typedef Product<MatrixReplacement, Rhs>::Scalar Scalar
```

### Public Static Functions

```
template<typename Dest>
static inline void scaleAndAddTo (Dest &dst, const MatrixReplacement &lhs, const Rhs
&rhs, const Scalar &alpha)
```

**Template Struct traits< SPH::MatrixReplacement >**

- Defined in file\_SPlisHSPlasH\_Utilsities\_MatrixFreeSolver.h

## Inheritance Relationships

### Base Type

- public Eigen::internal::traits< SystemMatrixType >

### Struct Documentation

```
template<>
struct traits<SPH::MatrixReplacement> : public Eigen::internal::traits<SystemMatrixType>
```

### Struct Elasticity\_Kugelstadt2021::ElasticObject

- Defined in file\_SPlisHSPlasH\_Elasticity\_Elasticity\_Kugelstadt2021.h

### Nested Relationships

This struct is a nested type of [Class Elasticity\\_Kugelstadt2021](#).

### Struct Documentation

```
struct SPH::Elasticity_Kugelstadt2021::ElasticObject
```

#### Public Functions

```
inline ElasticObject ()
inline ~ElasticObject ()
```

#### Public Members

```
std::string m_md5
std::vector<unsigned int> m_particleIndices
unsigned int m_nFixed
std::shared_ptr<Factorization> m_factorization
std::vector<Scalarf8, AlignmentAllocator<Scalarf8, 32>> m_f_avx
std::vector<Scalarf8, AlignmentAllocator<Scalarf8, 32>> m_sol_avx
std::vector<Scalarf8, AlignmentAllocator<Scalarf8, 32>> m_v_avx
std::vector<Scalarf8, AlignmentAllocator<Scalarf8, 32>> m_RHS
std::vector<Quaternion8f, AlignmentAllocator<Quaternion8f, 32>> m_quats
VectorXr m_rhs
VectorXr m_sol
```

## Struct Elasticity\_Kugelstadt2021::Factorization

- Defined in file\_SPlisHSPlasH\_Elasticity\_Elasticity\_Kugelstadt2021.h

### Nested Relationships

This struct is a nested type of *Class Elasticity\_Kugelstadt2021*.

### Struct Documentation

```
struct SPH::Elasticity_Kugelstadt2021::Factorization
```

#### Public Functions

```
inline Factorization ()
inline ~Factorization ()
```

#### Public Members

```
Real m_dt
Real m_mu
Eigen::SparseMatrix<Real, Eigen::RowMajor> m_DT_K
Eigen::SparseMatrix<Real, Eigen::RowMajor> m_D
Eigen::SparseMatrix<Real, Eigen::ColMajor> m_dampingMatrix
Eigen::SparseMatrix<Real, Eigen::ColMajor> m_matHTH
CholeskyAVXSolver *m_cholesky
```

## Struct FieldDescription

- Defined in file\_SPlisHSPlasH\_FluidModel.h

### Struct Documentation

```
struct SPH::FieldDescription
```

#### Public Functions

```
inline FieldDescription (const std::string &n, const FieldType &t, const
                      std::function<void*> const unsigned int
                      > &fct, const bool s = false)
```

## Public Members

```
std::string name
FieldType type
std::function<void* (const unsigned int) > getFct
bool storeData
```

## Struct PoissonDiskSampling::CellPosHasher

- Defined in file\_SPlisHSPlasH\_Utilsities\_PoissonDiskSampling.h

## Nested Relationships

This struct is a nested type of *Class PoissonDiskSampling*.

## Struct Documentation

```
struct SPH::PoissonDiskSampling::CellPosHasher
```

### Public Functions

```
inline std::size_t operator() (const CellPos &k) const
```

## Struct PoissonDiskSampling::HashEntry

- Defined in file\_SPlisHSPlasH\_Utilsities\_PoissonDiskSampling.h

## Nested Relationships

This struct is a nested type of *Class PoissonDiskSampling*.

## Struct Documentation

```
struct SPH::PoissonDiskSampling::HashEntry
```

Struct to store the hash entry (spatial hashing)

### Public Functions

```
inline HashEntry()
```

## Public Members

```
std::vector<unsigned int> samples  
unsigned int startIndex
```

### Struct PoissonDiskSampling::InitialPointInfo

- Defined in file\_SPlisHSPlasH\_Utils\_PoissonDiskSampling.h

## Nested Relationships

This struct is a nested type of *Class PoissonDiskSampling*.

## Struct Documentation

```
struct SPH::PoissonDiskSampling::InitialPointInfo
```

Struct to store the information of the initial points.

## Public Members

```
CellPos cP  
Vector3r pos  
unsigned int ID
```

### Struct Simulation::FluidInfo

- Defined in file\_SPlisHSPlasH\_Simulation.h

## Nested Relationships

This struct is a nested type of *Class Simulation*.

## Struct Documentation

```
struct SPH::Simulation::FluidInfo
```

Fluid object information

## Public Functions

```
inline bool hasSameParticleSampling(const FluidInfo &other)
```

## Public Members

```
int type
int numParticles
AlignedBox3r box
std::string id
std::string samplesFile
std::string visMeshFile
Vector3r translation
Matrix3r rotation
Vector3r scale
Vector3r initialVelocity
Vector3r initialAngularVelocity
unsigned char mode
bool invert
std::array<unsigned int, 3> resolutionSDF
unsigned int emitter_width
unsigned int emitter_height
Real emitter_velocity
Real emitter_emitStartTime
Real emitter_emitEndTime
unsigned int emitter_type
```

## Struct Simulation::NonPressureForceMethod

- Defined in file\_SPlisHSPlasH\_Simulation.h

## Nested Relationships

This struct is a nested type of *Class Simulation*.

## Struct Documentation

```
struct SPH::Simulation::NonPressureForceMethod
```

### Public Members

```
std::string m_name
std::function<NonPressureForceBase* (FluidModel*)> m_creator
int m_id
```

## Struct AverageCount

- Defined in file\_Utility\_Counting.h

## Struct Documentation

```
struct Utilities::AverageCount
```

### Public Members

```
Real sum
unsigned int numberOfCalls
```

## Struct AverageTime

- Defined in file\_Utility\_Timing.h

## Struct Documentation

```
struct Utilities::AverageTime
```

Struct to store the total time and the number of steps in order to compute the average time.

### Public Members

```
double totalTime
unsigned int counter
std::string name
```

## Struct MeshFaceIndices

- Defined in file\_Utilitys\_OBJLoader.h

### Struct Documentation

```
struct Utilities::MeshFaceIndices
```

Struct to store the position and normal indices.

#### Public Members

```
int posIndices[3]  
int texIndices[3]  
int normalIndices[3]
```

## Struct SceneLoader::AnimationFieldData

- Defined in file\_SPlisHSPlasH\_Utilitys\_SceneLoader.h

### Nested Relationships

This struct is a nested type of *Class SceneLoader*.

### Struct Documentation

```
struct Utilities::SceneLoader::AnimationFieldData
```

Struct to store an animation field object.

#### Public Members

```
std::string particleFieldName  
std::string expression[3]  
unsigned int shapeType  
Vector3r x  
Matrix3r rotation  
Vector3r scale  
Real startTime  
Real endTime
```

## Struct SceneLoader::BoundaryData

- Defined in file\_SPlisHSPlasH\_Utilsies\_SceneLoader.h

### Nested Relationships

This struct is a nested type of *Class SceneLoader*.

### Struct Documentation

```
struct Utilities::SceneLoader::BoundaryData
    Struct to store a boundary object.
```

#### Public Members

```
std::string samplesFile
std::string meshFile
Vector3r translation
Matrix3r rotation
Vector3r scale
Real density
bool dynamic
bool isWall
Eigen::Matrix<float, 4, 1, Eigen::DontAlign> color
void *rigidBody
std::string mapFile
bool mapInvert
Real mapThickness
Eigen::Matrix<unsigned int, 3, 1, Eigen::DontAlign> mapResolution
unsigned int samplingMode
bool isAnimated
```

## Struct SceneLoader::Box

- Defined in file\_SPlisHSPlasH\_Utilsies\_SceneLoader.h

## Nested Relationships

This struct is a nested type of [Class SceneLoader](#).

### Struct Documentation

```
struct Utilities::SceneLoader::Box
    Struct for an AABB.
```

#### Public Members

```
Vector3r m_minX
Vector3r m_maxX
```

## Struct SceneLoader::EmitterData

- Defined in file\_SPlisHSPlasH\_Utils\_SceneLoader.h

## Nested Relationships

This struct is a nested type of [Class SceneLoader](#).

### Struct Documentation

```
struct Utilities::SceneLoader::EmitterData
    Struct to store an emitter object.
```

#### Public Members

```
std::string id
unsigned int width
unsigned int height
Vector3r x
Real velocity
Matrix3r rotation
Real emitStartTime
Real emitEndTime
unsigned int type
```

## Struct SceneLoader::FluidBlock

- Defined in file\_SPlisHSPlasH\_Utilsies\_SceneLoader.h

### Nested Relationships

This struct is a nested type of *Class SceneLoader*.

### Struct Documentation

```
struct Utilities::SceneLoader::FluidBlock
```

Struct to store a fluid block.

#### Public Members

```
std::string id  
std::string visMeshFile  
Box box  
unsigned char mode  
Vector3r initialVelocity  
Vector3r initialAngularVelocity
```

## Struct SceneLoader::FluidData

- Defined in file\_SPlisHSPlasH\_Utilsies\_SceneLoader.h

### Nested Relationships

This struct is a nested type of *Class SceneLoader*.

### Struct Documentation

```
struct Utilities::SceneLoader::FluidData
```

Struct to store a fluid object.

#### Public Members

```
std::string id  
std::string samplesFile  
std::string visMeshFile  
Vector3r translation  
Matrix3r rotation  
Vector3r scale
```

```
Vector3r initialVelocity  
Vector3r initialAngularVelocity  
unsigned char mode  
bool invert  
std::array<unsigned int, 3> resolutionSDF
```

### Struct SceneLoader::MaterialData

- Defined in file\_SPlisHSPlasH\_Utilsies\_SceneLoader.h

#### Nested Relationships

This struct is a nested type of *Class SceneLoader*.

#### Struct Documentation

```
struct Utilities::SceneLoader::MaterialData
```

Struct to store particle coloring information.

#### Public Members

```
std::string id  
std::string colorField  
unsigned int colorMapType  
Real minValue  
Real maxValue  
unsigned int maxEmitterParticles  
bool emitterReuseParticles  
Vector3r emitterBoxMin  
Vector3r emitterBoxMax
```

### Struct SceneLoader::Scene

- Defined in file\_SPlisHSPlasH\_Utilsies\_SceneLoader.h

## Nested Relationships

This struct is a nested type of [Class SceneLoader](#).

### Struct Documentation

```
struct Utilities::SceneLoader::Scene
```

Struct to store scene information.

#### Public Members

```
std::vector<BoundaryData*> boundaryModels
std::vector<FluidData*> fluidModels
std::vector<FluidBlock*> fluidBlocks
std::vector<EmitterData*> emitters
std::vector<AnimationFieldData*> animatedFields
std::vector<MaterialData*> materials
Real particleRadius
bool sim2D
Real timeStepSize
Vector3r camPosition
Vector3r camLookat
```

## Struct TimingHelper

- Defined in file\_Utility\_Timing.h

### Struct Documentation

```
struct Utilities::TimingHelper
```

Struct to store a time measurement.

#### Public Members

```
std::chrono::time_point<std::chrono::high_resolution_clock> start
std::string name
```

## Template Class AlignmentAllocator

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

### Nested Relationships

#### Nested Types

- Template Struct AlignmentAllocator::rebind*

### Class Documentation

```
template<typename T, std::size_t N = 32>
class AlignmentAllocator
```

#### Public Types

```
typedef T value_type
typedef std::size_t size_type
typedef std::ptrdiff_t difference_type
typedef T *pointer
typedef const T *const_pointer
typedef T &reference
typedef const T &const_reference
```

#### Public Functions

```
inline AlignmentAllocator()

template<typename T2>
inline AlignmentAllocator(const AlignmentAllocator<T2, N>&)

inline ~AlignmentAllocator()

inline pointer adress(reference r)
inline const_pointer adress(const_reference r) const

inline pointer allocate(size_type n)

inline void deallocate(pointer p, size_type)

inline void construct(pointer p, const value_type &wert)

inline void destroy(pointer p)

inline size_type max_size() const

inline bool operator!=(const AlignmentAllocator<T, N> &other) const
inline bool operator==(const AlignmentAllocator<T, N> &other) const

template<typename T2>
```

---

```
struct rebind
```

### Public Types

```
typedef AlignmentAllocator<T2, N> other
```

## Class Matrix3f8

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Class Documentation

```
class Matrix3f8
```

#### Public Functions

```
inline Matrix3f8()
inline Matrix3f8(const Matrix3f &x)
inline Matrix3f8(const Vector3f8 &m1, const Vector3f8 &m2, const Vector3f8 &m3)
inline void setZero()
inline Scalarf8 &operator()(int i, int j)
inline void setCol(int i, const Vector3f8 &v)
inline void setCol(int i, const Scalarf8 &x, const Scalarf8 &y, const Scalarf8 &z)
inline Matrix3f8 operator*(const Scalarf8 &b) const
inline Vector3f8 operator*(const Vector3f8 &b) const
inline Matrix3f8 operator*(const Matrix3f8 &b) const
inline Matrix3f8 &operator+=(const Matrix3f8 &a)
inline Matrix3f8 transpose() const
inline Scalarf8 determinant() const
inline void store(std::vector<Matrix3r> &Mf) const
inline Matrix3r reduce() const
```

#### Public Members

*Scalarf8 m[3][3]*

## Class Quaternion8f

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

### Class Documentation

```
class Quaternion8f
```

#### Public Functions

```
inline Quaternion8f()
inline Quaternion8f(Scalarf8 x, Scalarf8 y, Scalarf8 z, Scalarf8 w)
inline Quaternion8f(Vector3f8 &v)
inline Scalarf8 &operator[](int i)
inline Scalarf8 operator[](int i) const
inline Scalarf8 &x()
inline Scalarf8 &y()
inline Scalarf8 &z()
inline Scalarf8 &w()
inline Scalarf8 x() const
inline Scalarf8 y() const
inline Scalarf8 z() const
inline Scalarf8 w() const
inline const Quaternion8f operator*(const Quaternion8f &a) const
inline void toRotationMatrix(Matrix3f8 &R)
inline void toRotationMatrix(Vector3f8 &R1, Vector3f8 &R2, Vector3f8 &R3)
inline void store(std::vector<Quaternionr> &qf) const
inline void set(const Quaternionr *qf)
```

#### Public Members

```
Scalarf8 q[4]
```

## Class Scalarf8

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Class Documentation

```
class Scalarf8
```

#### Public Functions

```
inline Scalarf8 ()
inline Scalarf8 (float f)
inline Scalarf8 (Real f0, Real f1, Real f2, Real f3, Real f4, Real f5, Real f6, Real f7)
inline Scalarf8 (float const *p)
inline Scalarf8 (_m256 const &x)
inline void setZero ()
inline Scalarf8 &operator= (_m256 const &x)
inline Scalarf8 sqrt () const
inline Scalarf8 rsqrt () const
inline Scalarf8 &load (float const *p)
inline void store (float *p) const
inline float reduce () const
```

#### Public Members

```
_m256 v
```

## Class AdhesionKernel

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

### Class Documentation

```
class SPH::AdhesionKernel
```

Adhesion kernel used for the surface tension method of Akinci et al. [ATT13].

References:

- [AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. ACM Trans. Graph., 32(6):182:1-182:8, November 2013. URL: <http://doi.acm.org/10.1145/2508363.2508395>

## Public Static Functions

```
static inline Real getRadius()
static inline void setRadius(Real val)
static inline Real W(const Real r)
    W(r,h) = (0.007/h^3.25)(-4r^2/h + 6r -2h)^0.25 if h/2 < r <= h
static inline Real W(const Vector3r &r)
static inline Real W_zero()
```

## Protected Static Attributes

```
static Real m_radius
static Real m_k
static Real m_W_zero
```

## Class AnimationField

- Defined in file\_SPlisHSPlasH\_AnimationField.h

## Class Documentation

```
class SPH::AnimationField
```

### Public Functions

```
AnimationField(const std::string &particleFieldName, const Vector3r &pos, const Matrix3r
    &rotation, const Vector3r &scale, const std::string expression[3], const un-
    signed int type = 0)
virtual ~AnimationField()
inline void setStartTime(Real val)
inline void setEndTime(Real val)
void step()
virtual void reset()
```

### Protected Functions

```
inline FORCE_INLINE bool inBox (const Vector3r &x, const Vector3r &xBox, const Matrix3
    inline FORCE_INLINE bool inCylinder (const Vector3r &x, const Vector3r &xCyl, const Ma
    inline FORCE_INLINE bool inSphere (const Vector3r &x, const Vector3r &pos, const Matrix3
    inline FORCE_INLINE bool inShape (const int type, const Vector3r &x, const Vector3r &p
```

## Protected Attributes

```
std::string m_particleFieldName
Vector3r m_x
Matrix3r m_rotation
Vector3r m_scale
std::string m_expression[3]
unsigned int m_type
Real m_startTime
Real m_endTime
```

## Class AnimationFieldSystem

- Defined in file\_SPlisHSPlasH\_AnimationFieldSystem.h

### Class Documentation

```
class SPH::AnimationFieldSystem
```

#### Public Functions

```
AnimationFieldSystem()
virtual ~AnimationFieldSystem()
void addAnimationField(const std::string &particleFieldName, const Vector3r &pos, const
                      Matrix3r &rotation, const Vector3r &scale, const std::string expres-
                      sion[3], const unsigned int type)
inline unsigned int numAnimationFields() const
inline std::vector<AnimationField*> &getAnimationFields()
void step()
void reset()
```

#### Protected Attributes

```
std::vector<AnimationField*> m_fields
```

## Class BinaryFileReader

- Defined in file\_Utilities\_BinaryFileReaderWriter.h

### Class Documentation

```
class SPH::BinaryFileReader
```

#### Public Functions

```
inline bool openFile (const std::string &fileName)
inline void closeFile ()
inline void readBuffer (char *buffer, size_t size)
template<typename T>
inline void read (T &v)

inline void read (std::string &str)
template<typename T>
inline void readMatrix (T &m)

template<typename T, int Rows, int Cols>
inline void readMatrixX (Eigen::Matrix<T, Rows, Cols> &m)

template<typename T, int Options, typename StorageIndex>
inline void readSparseMatrix (Eigen::SparseMatrix<T, Options, StorageIndex> &m)

template<typename T>
inline void readVector (std::vector<T> &m)
```

#### Public Members

```
std::ifstream m_file
```

## Class BinaryFileWriter

- Defined in file\_Utilities\_BinaryFileReaderWriter.h

### Class Documentation

```
class SPH::BinaryFileWriter
```

## Public Functions

```
inline bool openFile (const std::string &fileName)
inline void closeFile ()
inline void writeBuffer (const char *buffer, size_t size)
template<typename T>
inline void write (const T &v)
inline void write (const std::string &str)
template<typename T>
inline void writeMatrix (const T &m)
template<typename T, int Rows, int Cols>
inline void writeMatrixX (const Eigen::Matrix<T, Rows, Cols> &m)
template<typename T, int Options, typename StorageIndex>
inline void writeSparseMatrix (Eigen::SparseMatrix<T, Options, StorageIndex> &m)
template<typename T>
inline void writeVector (const std::vector<T> &m)
```

## Public Members

std::ofstream **m\_file**

## Class BlockJacobiPreconditioner3D

- Defined in file\_SPlisHSPlasH\_Utilsities\_MatrixFreeSolver.h

## Class Documentation

```
class SPH::BlockJacobiPreconditioner3D
Matrix-free 3x3 block Jacobi preconditioner
```

## Public Types

```
enum [anonymous]
Values:
enumerator ColsAtCompileTime
enumerator MaxColsAtCompileTime
typedef SystemMatrixType::StorageIndex StorageIndex
typedef void (*DiagonalMatrixElementFct) (const unsigned int, Matrix3r&, void*)
```

## Public Functions

```
inline BlockJacobiPreconditioner3D()

inline void init(const unsigned int dim, DiagonalMatrixElementFct fct, void *userData)

inline Eigen::Index rows() const

inline Eigen::Index cols() const

inline Eigen::ComputationInfo info()

template<typename MatType>
inline BlockJacobiPreconditioner3D &analyzePattern(const MatType&)

template<typename MatType>
inline BlockJacobiPreconditioner3D &factorize(const MatType &mat)

template<typename MatType>
inline BlockJacobiPreconditioner3D &compute(const MatType &mat)

template<typename Rhs, typename Dest>
inline void _solve_impl(const Rhs &b, Dest &x) const

template<typename Rhs>
inline const Eigen::Solve<BlockJacobiPreconditioner3D, Rhs> solve(const
Eigen::MatrixBase<Rhs>
&b) const
```

## Protected Attributes

```
unsigned int m_dim

DiagonalMatrixElementFct m_diagonalElementFct
    diagonal matrix element callback

void *mUserData

std::vector<Matrix3r> m_invDiag
```

## Class BoundaryModel

- Defined in file\_SPlisHSPlasH\_BoundaryModel.h

## Inheritance Relationships

### Derived Types

- public SPH::BoundaryModel\_Akinci2012 (*Class BoundaryModel\_Akinci2012*)
- public SPH::BoundaryModel\_Bender2019 (*Class BoundaryModel\_Bender2019*)
- public SPH::BoundaryModel\_Koschier2017 (*Class BoundaryModel\_Koschier2017*)

## Class Documentation

**class SPH::BoundaryModel**

The boundary model stores the information required for boundary handling.

Subclassed by *SPH::BoundaryModel\_Akinci2012*, *SPH::BoundaryModel\_Bender2019*,  
*SPH::BoundaryModel\_Koschier2017*

### Public Functions

```
BoundaryModel()
virtual ~BoundaryModel()
virtual void reset()
inline virtual void performNeighborhoodSearchSort()
inline virtual void saveState (BinaryFileWriter &binWriter)
inline virtual void loadState (BinaryFileReader &binReader)
inline RigidBodyObject *getRigidBodyObject()
inline FORCE_INLINE void addForce (const Vector3r &pos, const Vector3r &f)
inline FORCE_INLINE void getPointVelocity (const Vector3r &x, Vector3r &res)
void getForceAndTorque (Vector3r &force, Vector3r &torque)
void clearForceAndTorque()
```

### Protected Attributes

```
RigidBodyObject *m_rigidBody
std::vector<Vector3r> m_forcePerThread
std::vector<Vector3r> m_torquePerThread
```

#### Class BoundaryModel\_Akinci2012

- Defined in file \_SPlisHSPlasH\_BoundaryModel\_Akinci2012.h

### Inheritance Relationships

#### Base Type

- public SPH::BoundaryModel (*Class BoundaryModel*)

## Class Documentation

```
class SPH::BoundaryModel_Akinci2012 : public SPH::BoundaryModel
```

The boundary model stores the information required for boundary handling using the approach of Akinci et al. 2012 [AIA+12].

References:

- [AIA+12] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible SPH. ACM Trans. Graph., 31(4):62:1-62:8, July 2012. URL: <http://doi.acm.org/10.1145/2185520.2185558>

## Public Functions

```
BoundaryModel_Akinci2012()

virtual ~BoundaryModel_Akinci2012()

inline unsigned int numberOfParticles() const

inline unsigned int getPointSetIndex() const

inline bool isSorted() const

void computeBoundaryVolume()

void resize(const unsigned int numBoundaryParticles)

virtual void reset()

virtual void performNeighborhoodSearchSort()

virtual void saveState(BinaryFileWriter &binWriter)

virtual void loadState(BinaryFileReader &binReader)

void initModel(RigidBodyObject *rbo, const unsigned int numBoundaryParticles, Vector3r
               *boundaryParticles)

inline FORCE_INLINE Vector3r & getPosition0 (const unsigned int i)

inline FORCE_INLINE const Vector3r & getPosition0 (const unsigned int i) const

inline FORCE_INLINE void setPosition0 (const unsigned int i, const Vector3r &pos)

inline FORCE_INLINE Vector3r & getPosition (const unsigned int i)

inline FORCE_INLINE const Vector3r & getPosition (const unsigned int i) const

inline FORCE_INLINE void setPosition (const unsigned int i, const Vector3r &pos)

inline FORCE_INLINE Vector3r & getVelocity (const unsigned int i)

inline FORCE_INLINE const Vector3r & getVelocity (const unsigned int i) const

inline FORCE_INLINE void setVelocity (const unsigned int i, const Vector3r &vel)

inline FORCE_INLINE const Real & getVolume (const unsigned int i) const

inline FORCE_INLINE Real & getVolume (const unsigned int i)

inline FORCE_INLINE void setVolume (const unsigned int i, const Real &val)
```

## Protected Attributes

```
bool m_sorted
unsigned int m_pointSetIndex
std::vector<Vector3r> m_x0
std::vector<Vector3r> m_x
std::vector<Vector3r> m_v
std::vector<Real> m_V
```

## Class BoundaryModel\_Bender2019

- Defined in file \_SPlisHSPlasH\_BoundaryModel\_Bender2019.h

## Inheritance Relationships

### Base Type

- public SPH::BoundaryModel (*Class BoundaryModel*)

## Class Documentation

```
class SPH::BoundaryModel_Bender2019 : public SPH::BoundaryModel
```

The boundary model stores the information required for boundary handling using the approach of Bender et al. 2019 [BWK19].

References:

- [BWK19] Jan Bender, Tassilo Kugelstadt, Marcel Weiler, and Dan Koschier. Volume maps: an implicit boundary representation for SPH. In Proceedings of ACM SIGGRAPH Conference on Motion, Interaction and Games, MIG '19. ACM, 2019. URL: <https://dl.acm.org/doi/10.1145/3359566.3360077>

## Public Functions

```
BoundaryModel_Bender2019()
virtual ~BoundaryModel_Bender2019()
void initModel (RigidBodyObject *rbo)
virtual void reset ()
inline Discregrid::DiscreteGrid *getMap ()
inline void setMap (Discregrid::DiscreteGrid *map)
inline Real getMaxDist () const
inline void setMaxDist (Real val)
inline Real getMaxVel () const
inline void setMaxVel (Real val)
inline FORCE_INLINE const Real & getBoundaryVolume (const unsigned int fluidIndex, const
```

```
inline FORCE_INLINE Real & getBoundaryVolume (const unsigned int fluidIndex, const unsig  
inline FORCE_INLINE void setBoundaryVolume (const unsigned int fluidIndex, const unsig  
inline FORCE_INLINE Vector3r & getBoundaryXj (const unsigned int fluidIndex, const unsig  
inline FORCE_INLINE const Vector3r & getBoundaryXj (const unsigned int fluidIndex, const unsig  
inline FORCE_INLINE void setBoundaryXj (const unsigned int fluidIndex, const unsigned
```

## Protected Attributes

```
Discregrid::DiscreteGrid *m_map  
std::vector<std::vector<Real>> m_boundaryVolume  
std::vector<std::vector<Vector3r>> m_boundaryXj  
Real m_maxDist  
Real m_maxVel
```

## Class BoundaryModel\_Koschier2017

- Defined in file\_SPlisHSPlasH\_BoundaryModel\_Koschier2017.h

## Inheritance Relationships

### Base Type

- public SPH::BoundaryModel (*Class BoundaryModel*)

## Class Documentation

```
class SPH::BoundaryModel_Koschier2017 : public SPH::BoundaryModel
```

The boundary model stores the information required for boundary handling using the approach of Koschier and Bender 2017 [KB17].

References:

- [KB17] Dan Koschier and Jan Bender. Density maps for improved SPH boundary handling. In ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 1-10. July 2017. URL: <http://dx.doi.org/10.1145/3099564.3099565>

## Public Functions

```
BoundaryModel_Koschier2017()  
virtual ~BoundaryModel_Koschier2017()  
void initModel (RigidBodyObject *rbo)  
virtual void reset()  
inline Discregrid::DiscreteGrid *getMap()  
inline void setMap (Discregrid::DiscreteGrid *map)
```

```

inline Real getMaxDist () const
inline void setMaxDist (Real val)
inline Real getMaxVel () const
inline void setMaxVel (Real val)

inline FORCE_INLINE const Real & getBoundaryDensity (const unsigned int fluidIndex, co
inline FORCE_INLINE Real & getBoundaryDensity (const unsigned int fluidIndex, const un
inline FORCE_INLINE void setBoundaryDensity (const unsigned int fluidIndex, const unsi
inline FORCE_INLINE Vector3r & getBoundaryDensityGradient (const unsigned int fluidInd
inline FORCE_INLINE const Vector3r & getBoundaryDensityGradient (const unsigned int fl
inline FORCE_INLINE void setBoundaryDensityGradient (const unsigned int fluidIndex, co
inline FORCE_INLINE Vector3r & getBoundaryXj (const unsigned int fluidIndex, const unsi
inline FORCE_INLINE const Vector3r & getBoundaryXj (const unsigned int fluidIndex, const
inline FORCE_INLINE void setBoundaryXj (const unsigned int fluidIndex, const unsigned

```

## Protected Attributes

```

Discregrid::DiscreteGrid *m_map
std::vector<std::vector<Real>> m_boundaryDensity
std::vector<std::vector<Vector3r>> m_boundaryDensityGradient
std::vector<std::vector<Vector3r>> m_boundaryXj

Real m_maxDist
Real m_maxVel

```

## Class CholeskyAVXSolver

- Defined in file\_SPlisHSPlasH\_Elasticity\_CholeskyAVXSolver.h

## Class Documentation

```
class SPH::CholeskyAVXSolver
```

Cholesky solver which uses AVX instructions. Written by José Antonio Fernández-Fernández.

## Public Functions

```

CholeskyAVXSolver() = default
~CholeskyAVXSolver() = default
CholeskyAVXSolver(const Eigen::SparseMatrix<float> &lhs)
CholeskyAVXSolver(const Eigen::SparseMatrix<double> &lhs)
void solve (float *solution, const float *rhs, const int stride)
void save (SPH::BinaryFileWriter &binWriter)

```

```
void load (SPH::BinaryFileReader &binReader)
```

### Public Members

```
CholeskySparseMatrixfAVX L
CholeskySparseMatrixfAVX LT
std::vector<int> perm
std::vector<int> perm_inv
int n_rhs_lines = -1
int ndofs = -1
```

## Class CholeskySparseMatrixfAVX

- Defined in file\_SPlisHSPlasH\_Elasticity\_CholeskyAVXSolver.h

### Class Documentation

```
class SPH::CholeskySparseMatrixfAVX
```

### Public Functions

```
inline CholeskySparseMatrixfAVX ()
CholeskySparseMatrixfAVX (const Eigen::SparseMatrix<float, Eigen::RowMajor> &lhs)
void save (SPH::BinaryFileWriter &binWriter)
void load (SPH::BinaryFileReader &binReader)
```

### Public Members

```
std::vector<Scalarf8, AlignmentAllocator<Scalarf8, 32>> vals
std::vector<int> cols
std::vector<int> rows_offset
std::vector<float> diagonal_inv
int n_rhs_lines = -1
```

## Class CohesionKernel

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

### Class Documentation

```
class SPH::CohesionKernel
```

Cohesion kernel used for the surface tension method of Akinci et al. [ATT13].

References:

- [AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. ACM Trans. Graph., 32(6):182:1-182:8, November 2013. URL: <http://doi.acm.org/10.1145/2508363.2508395>

### Public Static Functions

```
static inline Real getRadius()  
static inline void setRadius(Real val)  
static inline Real W(const Real r)  
    W(r,h) = (32/(pi h^9))(h-r)^3*r^3 if h/2 < r <= h (32/(pi h^9))(2*(h-r)^3*r^3 - h^6/64 if 0 < r <= h/2  
static inline Real W(const Vector3r &r)  
static inline Real W_zero()
```

### Protected Static Attributes

```
static Real m_radius  
static Real m_k  
static Real m_c  
static Real m_W_zero
```

## Class CubicKernel

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

### Class Documentation

```
class SPH::CubicKernel
```

Cubic spline kernel.

## Public Static Functions

```
static inline Real getRadius()
static inline void setRadius(Real val)
static inline Real W(const Real r)
static inline Real W(const Vector3r &r)
static inline Vector3r gradW(const Vector3r &r)
static inline Real W_zero()
```

## Protected Static Attributes

```
static Real m_radius
static Real m_k
static Real m_l
static Real m_W_zero
```

## Class CubicKernel2D

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

## Class Documentation

```
class SPH::CubicKernel2D
    Cubic spline kernel (2D).
```

## Public Static Functions

```
static inline Real getRadius()
static inline void setRadius(Real val)
static inline Real W(const Real r)
static inline Real W(const Vector3r &r)
static inline Vector3r gradW(const Vector3r &r)
static inline Real W_zero()
```

## Protected Static Attributes

```
static Real m_radius
static Real m_k
static Real m_l
static Real m_W_zero
```

## Class DebugTools

- Defined in file\_SPlisHSPlasH\_Utilsities\_DebugTools.h

## Inheritance Relationships

### Base Type

- public ParameterObject

## Class Documentation

```
class SPH::DebugTools : public ParameterObject
```

### Public Functions

```
DebugTools()
~DebugTools()
void init()
void cleanup()
void step()
void reset()
void performNeighborhoodSearchSort()
void emittedParticles (FluidModel *model, const unsigned int startIndex)
```

### Public Static Attributes

```
static int DETERMINE_THREAD_IDS = -1
static int DETERMINE_NUM_NEIGHBORS = -1
static int DETERMINE_VELOCITY_CHANGES = -1
```

### Protected Functions

```
virtual void initParameters ()  
void determineThreadIds ()  
void determineNumNeighbors ()  
void determineVelocityChanges ()
```

### Protected Attributes

```
bool m_determineThreadIds  
std::vector<std::vector<unsigned int>> m_threadIds  
bool m_determineNumNeighbors  
std::vector<std::vector<unsigned int>> m_numNeighbors  
bool m_determineVelocityChanges  
std::vector<std::vector<Vector3r>> m_vOld  
std::vector<std::vector<Vector3r>> m_velocityChanges
```

## Class DragBase

- Defined in file \_SPlisHSPlasH\_Drag\_DragBase.h

### Inheritance Relationships

#### Base Type

- public SPH::NonPressureForceBase (*Class NonPressureForceBase*)

#### Derived Types

- public SPH::DragForce\_Gissler2017 (*Class DragForce\_Gissler2017*)
- public SPH::DragForce\_Macklin2014 (*Class DragForce\_Macklin2014*)

### Class Documentation

**class** SPH::**DragBase** : **public** SPH::*NonPressureForceBase*  
Base class for all drag force methods.

Subclassed by *SPH::DragForce\_Gissler2017*, *SPH::DragForce\_Macklin2014*

## Public Functions

```
DragBase (FluidModel *model)
virtual ~DragBase (void)
```

## Public Static Attributes

```
static int DRAG_COEFFICIENT = -1
```

## Protected Functions

```
virtual void initParameters()
```

## Protected Attributes

```
Real m_dragCoefficient
```

## Class DragForce\_Gissler2017

- Defined in file\_SPlisHSPlasH\_Drag\_DragForce\_Gissler2017.h

## Inheritance Relationships

### Base Type

- public SPH::DragBase (*Class DragBase*)

## Class Documentation

```
class SPH::DragForce_Gissler2017 : public SPH::DragBase
```

This class implements the drag force computation introduced by Gissler et al. [GPB+17].

References:

- [GPB+17] Christoph Gissler, Stefan Band, Andreas Peer, Markus Ihmsen, and Matthias Teschner. Approximate air-fluid interactions for SPH. In Virtual Reality Interactions and Physical Simulations, 1-10. April 2017. URL: <http://dx.doi.org/10.2312/vrphys.20171081>

## Public Functions

```
DragForce_Gissler2017 (FluidModel *model)
virtual ~DragForce_Gissler2017 (void)
virtual void step()
virtual void reset()
```

## Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)
```

## Protected Attributes

```
const Real rho_a = static_cast<Real>(1.2041)
const Real sigma = static_cast<Real>(0.0724)
const Real mu_1 = static_cast<Real>(0.00102)
const Real C_F = static_cast<Real>(1.0 / 3.0)
const Real C_k = static_cast<Real>(8.0)
const Real C_d = static_cast<Real>(5.0)
const Real C_b = static_cast<Real>(0.5)
const Real mu_a = static_cast<Real>(0.00001845)
```

## Class DragForce\_Macklin2014

- Defined in file\_SPlisHSPlasH\_Drag\_DragForce\_Macklin2014.h

## Inheritance Relationships

### Base Type

- public SPH::DragBase (*Class DragBase*)

## Class Documentation

```
class SPH::DragForce_Macklin2014 : public SPH::DragBase
```

This class implements the drag force computation introduced by Macklin et al. [MMCK14].

References:

- [MMCK14] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified Particle Physics for Real-Time Applications. ACM Trans. Graph., 33(4):1-12, 2014. URL: <http://doi.acm.org/10.1145/2601097.2601152>

## Public Functions

```
DragForce_Macklin2014 (FluidModel *model)
virtual ~DragForce_Macklin2014 (void)
virtual void step ()
virtual void reset ()
```

## Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
```

## Class Elasticity\_Becker2009

- Defined in file\_SPlisHSPlasH\_Elasticity\_Elasticity\_Becker2009.h

## Inheritance Relationships

### Base Type

- public SPH::ElasticityBase (*Class ElasticityBase*)

## Class Documentation

```
class SPH::Elasticity_Becker2009 : public SPH::ElasticityBase
```

This class implements the corotated SPH method for deformable solids introduced by Becker et al. [BIT09].

References:

- [BIT09] Markus Becker, Markus Ihmsen, and Matthias Teschner. Corotated SPH for deformable solids. In Proceedings of Eurographics Conference on Natural Phenomena, 27-34. 2009. URL: <http://dx.doi.org/10.2312/EG/DL/conf/EG2009/nph/027-034>

## Public Functions

```
Elasticity_Becker2009 (FluidModel *model)
virtual ~Elasticity_Becker2009 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
virtual void saveState (BinaryFileWriter &binWriter)
virtual void loadState (Binary.FileReader &binReader)
```

## Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
```

## Public Static Attributes

```
static int ALPHA = -1
```

## Protected Functions

```
void initValues ()  
void computeRotations ()  
void computeStress ()  
void computeForces ()  
virtual void initParameters ()  
inline FORCE_INLINE void symMatTimesVec (const Vector6r &M, const Vector3r &v, Vector3r &w)
```

## Protected Attributes

```
std::vector<unsigned int> m_current_to_initial_index  
std::vector<unsigned int> m_initial_to_current_index  
std::vector<std::vector<unsigned int>> m_initialNeighbors  
std::vector<Real> m_restVolumes  
std::vector<Matrix3r> m_rotations  
std::vector<Vector6r> m_stress  
std::vector<Matrix3r> m_F  
Real m_alpha
```

## Class Elasticity\_Kugelstadt2021

- Defined in file\_SPlisHSPlasH\_Elasticity\_Elasticity\_Kugelstadt2021.h

## Nested Relationships

### Nested Types

- Struct Elasticity\_Kugelstadt2021::ElasticObject*
- Struct Elasticity\_Kugelstadt2021::Factorization*

## Inheritance Relationships

### Base Type

- public SPH::ElasticityBase (*Class ElasticityBase*)

## Class Documentation

**class** SPH::**Elasticity\_Kugelstadt2021** : **public** SPH::*ElasticityBase*

This class implements the implicit SPH formulation for incompressible linearly elastic solids introduced by Kugelstadt et al. [KBF+21].

References:

- [KBF+21] Tassilo Kugelstadt, Jan Bender, José Antonio Fernández-Fernández, Stefan Rhys Jeske, Fabian Lüschnner, Andreas Longva. Fast Corotated Elastic SPH Solids with Implicit Zero-Energy Mode Control. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2021. URL: <http://dx.doi.org/10.1145/3480142>

### Public Functions

```
Elasticity_Kugelstadt2021 (FluidModel *model)
virtual ~Elasticity_Kugelstadt2021 (void)
virtual void step ()
    Perform a step of the elasticity solver.

virtual void reset ()
virtual void performNeighborhoodSearchSort ()
virtual void saveState (BinaryFileWriter &binWriter)
virtual void loadState (Binary.FileReader &binReader)
void computeRotations ()
    Extract rotation matrices from deformation gradients.
```

### Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
static void matrixVecProd (const Real *vec, Real *result, void *userData)
    Matrix vector product used by the matrix-free conjugate gradient solver to solve the system in Eq. 30.
```

## Public Static Attributes

```
static int ITERATIONS_V = -1  
static int MAX_ITERATIONS_V = -1  
static int MAX_ERROR_V = -1  
static int ALPHA = -1  
static int MASS_DAMPING_COEFF = -1  
static int STIFFNESS_DAMPING_COEFF = -1  
static int MAX_NEIGHBORS = -1
```

## Protected Types

```
typedef Eigen::ConjugateGradient<MatrixReplacement, Eigen::Lower | Eigen::Upper, Eigen::IdentityPreconditioner> solve
```

## Protected Functions

```
void computeRHS (VectorXr &rhs)
```

Compute right hand side of the linear system of the volume solver (Eq. 30).

```
std::string computeMD5 (const unsigned int objIndex)
```

Compute an MD4 check sum using the neighborhood structure in order to recognize known particle models (cache).

```
void initValues ()
```

Initialize the particle neighborhoods in the reference configuration. Fix particles which lie in the user-defined bounding box. Find out if there are multiple separate objects in the phase. Finally, compute kernel gradient correction matrices and factorization.

```
void initSystem ()
```

Initialize the solver for the linear system by either computing a factorization or loading a factorization from the cache.

```
void initFactorization (std::shared_ptr<Factorization> factorization, std::vector<unsigned int>  
    &particleIndices, const unsigned int nFixed, const Real dt, const Real mu)
```

Compute the factorization of the linear system matrix. This is only done once at the beginning of the simulation.

```
void findObjects ()
```

Find separate objects by object id.

```
void computeMatrixL ()
```

Compute kernel gradient correction matrices (Eq. 8).

```
inline void APP_Newton_AVX (const Vector3f8 &F1, const Vector3f8 &F2, const Vector3f8  
    &F3, Quaternion8f &q)
```

```
void precomputeValues ()
```

Precompute some values and products to improve the performance of the solvers.

```
void stepElasticitySolver ()
```

Solve the linear system for the stretching forces including zero energy mode control using the precomputed matrix factorization.

```

void stepVolumeSolver()
    Solver for the volume conservation forces (Eq. 30).

virtual void initParameters()

virtual void deferredInit()
    This function is called after the simulation scene is loaded and all parameters are initialized. While reading a scene file several parameters can change. The deferred init function should initialize all values which depend on these parameters.

inline FORCE_INLINE void symMatTimesVec (const Vector6r &M, const Vector3r &v, Vector3r &res)

void rotationMatricesToAVXQuaternions()
    Convert all rotation matrices to AVX quaternions.

```

## Protected Attributes

```

std::vector<unsigned int> m_current_to_initial_index
std::vector<unsigned int> m_initial_to_current_index
std::vector<std::vector<unsigned int>> m_initialNeighbors
std::vector<Real> m_restVolumes
std::vector<Matrix3r> m_rotations
std::vector<Real> m_stress
std::vector<Matrix3r> m_L
std::vector<Matrix3r> m_F
std::vector<Vector3r> m_vDiff
std::vector<Matrix3r> m_RL
unsigned int m_iterationsV
unsigned int m_maxIterV
Real m_maxErrorV
Real m_alpha
Real m_massDampingCoeff
Real m_stiffnessDampingCoeff
int m_maxNeighbors
unsigned int m_totalNeighbors
std::vector<ElasticObject*> m_objects
Real m_lambda
Real m_mu
std::vector<Vector3r, Eigen::aligned_allocator<Vector3r>> m_precomp_RL_gradW
std::vector<Vector3r, Eigen::aligned_allocator<Vector3r>> m_precomp_RLj_gradW
std::vector<unsigned int> m_precomputed_indices
Solver m_solver

```

## Protected Static Functions

```
static inline void computeF (const unsigned int i, const Real *x, Elasticity_Kugelstadt2021
                           *e)
struct ElasticObject
```

## Public Functions

```
inline ElasticObject ()
inline ~ElasticObject ()
```

## Public Members

```
std::string m_md5
std::vector<unsigned int> m_particleIndices
unsigned int m_nFixed
std::shared_ptr<Factorization> m_factorization
std::vector<Scalarrf8, AlignmentAllocator<Scalarrf8, 32>> m_f_avx
std::vector<Scalarrf8, AlignmentAllocator<Scalarrf8, 32>> m_sol_avx
std::vector<Scalarrf8, AlignmentAllocator<Scalarrf8, 32>> m_v_avx
std::vector<Scalarrf8, AlignmentAllocator<Scalarrf8, 32>> m_RHS
std::vector<Quaternion8f, AlignmentAllocator<Quaternion8f, 32>> m_quats
VectorXr m_rhs
VectorXr m_sol
struct Factorization
```

## Public Functions

```
inline Factorization ()
inline ~Factorization ()
```

## Public Members

```
Real m_dt
Real m_mu
Eigen::SparseMatrix<Real, Eigen::RowMajor> m_DT_K
Eigen::SparseMatrix<Real, Eigen::RowMajor> m_D
Eigen::SparseMatrix<Real, Eigen::ColMajor> m_dampingMatrix
Eigen::SparseMatrix<Real, Eigen::ColMajor> m_matHTH
CholeskyAVXSolver *m_cholesky
```

## Class Elasticity\_Peer2018

- Defined in file\_SPlisHSPlasH\_Elasticity\_Elasticity\_Peer2018.h

## Inheritance Relationships

### Base Type

- public SPH::ElasticityBase (*Class ElasticityBase*)

## Class Documentation

```
class SPH::Elasticity_Peer2018 : public SPH::ElasticityBase
```

This class implements the implicit SPH formulation for incompressible linearly elastic solids introduced by Peer et al. [PGBT17].

References:

- [PGBT17] Andreas Peer, Christoph Gissler, Stefan Band, and Matthias Teschner. An implicit SPH formulation for incompressible linearly elastic solids. Computer Graphics Forum, 2017. URL: <http://dx.doi.org/10.1111/cgf.13317>

### Public Functions

```
Elasticity_Peer2018 (FluidModel *model)
virtual ~Elasticity_Peer2018 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
virtual void saveState (BinaryFileWriter &binWriter)
virtual void loadState (Binary.FileReader &binReader)
```

### Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
static void matrixVecProd (const Real *vec, Real *result, void *userData)
```

### Public Static Attributes

```
static int ITERATIONS = -1
static int MAX_ITERATIONS = -1
static int MAX_ERROR = -1
static int ALPHA = -1
```

## Protected Types

```
typedef Eigen::ConjugateGradient<MatrixReplacement, Eigen::Lower | Eigen::Upper, Eigen::IdentityPreconditioner> Solver
```

## Protected Functions

```
void initValues ()  
void computeMatrixL ()  
void computeRotations ()  
void computeRHS (VectorXr &rhs)  
virtual void initParameters ()  
virtual void deferredInit ()
```

This function is called after the simulation scene is loaded and all parameters are initialized. While reading a scene file several parameters can change. The deferred init function should initialize all values which depend on these parameters.

```
inline FORCE_INLINE void symMatTimesVec (const Vector6r &M, const Vector3r &v, Vector3r &rv)
```

## Protected Attributes

```
std::vector<unsigned int> m_current_to_initial_index  
std::vector<unsigned int> m_initial_to_current_index  
std::vector<std::vector<unsigned int>> m_initialNeighbors  
std::vector<Real> m_restVolumes  
std::vector<Matrix3r> m_rotations  
std::vector<Vector6r> m_stress  
std::vector<Matrix3r> m_L  
std::vector<Matrix3r> m_RL  
std::vector<Matrix3r> m_F  
unsigned int m_iterations  
unsigned int m_maxIter  
Real m_maxError  
Real m_alpha  
Solver m_solver
```

## Class ElasticityBase

- Defined in file\_SPlisHSPlasH\_Elasticity\_ElasticityBase.h

## Inheritance Relationships

### Base Type

- public SPH::NonPressureForceBase (*Class NonPressureForceBase*)

### Derived Types

- public SPH::Elasticity\_Becker2009 (*Class Elasticity\_Becker2009*)
- public SPH::Elasticity\_Kugelstadt2021 (*Class Elasticity\_Kugelstadt2021*)
- public SPH::Elasticity\_Peer2018 (*Class Elasticity\_Peer2018*)

## Class Documentation

```
class SPH::ElasticityBase : public SPH::NonPressureForceBase
Base class for all elasticity methods.
```

Subclassed by *SPH::Elasticity\_Becker2009*, *SPH::Elasticity\_Kugelstadt2021*, *SPH::Elasticity\_Peer2018*

### Public Functions

```
ElasticityBase (FluidModel *model)
virtual ~ElasticityBase (void)
```

### Public Static Attributes

```
static int YOUNGS_MODULUS = -1
static int POISSON_RATIO = -1
static int FIXED_BOX_MIN = -1
static int FIXED_BOX_MAX = -1
```

### Protected Functions

```
virtual void initParameters ()
void determineFixedParticles ()
Mark all particles in the bounding box as fixed.
```

## Protected Attributes

```
Real m_youngsModulus  
Real m_poissonRatio  
Vector3r m_fixedBoxMin  
Vector3r m_fixedBoxMax
```

## Class Emitter

- Defined in file\_SPlisHSPlasH\_Emitter.h

## Class Documentation

```
class SPH::Emitter
```

### Public Functions

```
Emitter (FluidModel *model, const unsigned int width, const unsigned int height, const Vector3r  
         &pos, const Matrix3r &rotation, const Real velocity, const unsigned int type = 0)  
virtual ~Emitter ()  
void emitParticles (std::vector<unsigned int> &reusedParticles, unsigned int &indexReuse, un-  
signed int &numEmittedParticles)  
void emitParticlesCircle (std::vector<unsigned int> &reusedParticles, unsigned int &in-  
dexReuse, unsigned int &numEmittedParticles)  
inline Real getNextEmitTime () const  
inline void setNextEmitTime (Real val)  
inline void setEmitStartTime (Real val)  
inline void setEmitEndTime (Real val)  
void step (std::vector<unsigned int> &reusedParticles, unsigned int &indexReuse, unsigned int &nu-  
mEmittedParticles)  
virtual void reset ()  
void saveState (BinaryFileWriter &binWriter)  
void loadState (BinaryFileReader &binReader)  
inline const Vector3r &getPosition () const  
inline void setPosition (const Vector3r &x)  
inline const Matrix3r &getRotation () const  
inline void setRotation (const Matrix3r &r)  
inline const Real getVelocity () const  
inline void setVelocity (const Real v)  
inline const unsigned int getObjectId () const  
inline void setObjectId (const unsigned int v)
```

## Public Static Functions

```
static Vector3r getSize (const Real width, const Real height, const int type)
```

## Protected Functions

```
inline FORCE_INLINE bool inBox (const Vector3r &x, const Vector3r &xBox, const Matrix3r &rotation)
inline FORCE_INLINE bool inCylinder (const Vector3r &x, const Vector3r &xCyl, const Matrix3r &rotation)
```

## Protected Attributes

```
FluidModel *m_model
unsigned int m_width
unsigned int m_height
Vector3r m_x
Matrix3r m_rotation
Real m_velocity
unsigned int m_type
Real m_nextEmitTime
Real m_emitStartTime
Real m_emitEndTime
unsigned int m_emitCounter
unsigned int m_objectId
```

## Class EmitterSystem

- Defined in file\_SPlisHSPlasH\_EmitterSystem.h

## Class Documentation

```
class SPH::EmitterSystem
```

### Public Functions

```
EmitterSystem (FluidModel *model)
virtual ~EmitterSystem ()
void enableReuseParticles (const Vector3r &boxMin = Vector3r(-1, -1, -1), const Vector3r &boxMax = Vector3r(1, 1, 1))
void disableReuseParticles ()
void addEmitter (const unsigned int width, const unsigned int height, const Vector3r &pos, const Matrix3r &rotation, const Real velocity, const unsigned int type)
inline unsigned int numEmitters () const
```

```
inline std::vector<Emitter*> &getEmitters ()  
inline unsigned int numReusedParticles () const  
inline unsigned int numEmittedParticles () const  
void step ()  
void reset ()  
void saveState (BinaryFileWriter &binWriter)  
void loadState (BinaryFileReader &binReader)
```

### Protected Functions

```
void reuseParticles ()
```

### Protected Attributes

```
FluidModel *m_model  
bool m_reuseParticles  
Vector3r m_boxMin  
Vector3r m_boxMax  
unsigned int m_numberOfEmittedParticles  
unsigned int m_numReusedParticles  
std::vector<unsigned int> m_reusedParticles  
std::vector<Emitter*> m_emitters
```

### Protected Static Attributes

```
static const unsigned int m_maxParticlesToReusePerStep = 50000
```

## Class FluidModel

- Defined in file\_SPlisHSPlasH\_FluidModel.h

### Inheritance Relationships

#### Base Type

- public ParameterObject

## Class Documentation

```
class SPH::FluidModel : public ParameterObject
    The fluid model stores the particle and simulation information.
```

### Public Functions

```
FluidModel()
FluidModel(const FluidModel&) = delete
FluidModel &operator=(const FluidModel&) = delete
virtual ~FluidModel()
void init()
void deferredInit()
    This function is called after the simulation scene is loaded and all parameters are initialized. While reading a scene file several parameters can change. The deferred init function should initialize all values which depend on these parameters.
inline std::string getId() const
inline FORCE_INLINE Real getDensity0 () const
void setDensity0 (const Real v)
inline unsigned int getPointSetIndex () const
void addField(const FieldDescription &field)
inline const std::vector<FieldDescription> &getFields()
inline const FieldDescription &getField (const unsigned int i)
const FieldDescription &getField (const std::string &name)
inline const unsigned int numberofFields ()
void removeFieldByName (const std::string &fieldName)
void setNumActiveParticles (const unsigned int num)
inline unsigned int numberOfParticles () const
inline EmitterSystem *getEmitterSystem ()
virtual void reset ()
void performNeighborhoodSearchSort ()
void initModel (const std::string &id, const unsigned int nFluidParticles, Vector3r *fluidParticles,
Vector3r *fluidVelocities, unsigned int *fluidObjectIds, const unsigned int nMax-
EmitterParticles)
inline const unsigned int numParticles () const
unsigned int numActiveParticles () const
inline unsigned int getNumActiveParticles0 () const
inline void setNumActiveParticles0 (unsigned int val)
void emittedParticles (const unsigned int startIndex)
inline unsigned int getSurfaceTensionMethod () const
```

```
void setSurfaceTensionMethod(const std::string &val)
void setSurfaceTensionMethod(const unsigned int val)
inline unsigned int getViscosityMethod() const
void setViscosityMethod(const std::string &val)
void setViscosityMethod(const unsigned int val)
inline unsigned int getVorticityMethod() const
void setVorticityMethod(const std::string &val)
void setVorticityMethod(const unsigned int val)
inline unsigned int getDragMethod() const
void setDragMethod(const std::string &val)
void setDragMethod(const unsigned int val)
inline unsigned int getElasticityMethod() const
void setElasticityMethod(const std::string &val)
void setElasticityMethod(const unsigned int val)
inline SurfaceTensionBase *getSurfaceTensionBase()
inline ViscosityBase *getViscosityBase()
inline VorticityBase *getVorticityBase()
inline DragBase *getDragBase()
inline ElasticityBase *getElasticityBase()
void setDragMethodChangedCallback(std::function<void>
    > const &callBackFct)
void setSurfaceMethodChangedCallback(std::function<void>
    > const &callBackFct)
void setViscosityMethodChangedCallback(std::function<void>
    > const &callBackFct)
void setVorticityMethodChangedCallback(std::function<void>
    > const &callBackFct)
void setElasticityMethodChangedCallback(std::function<void>
    > const &callBackFct)
void computeSurfaceTension()
void computeViscosity()
void computeVorticity()
void computeDragForce()
void computeElasticity()
void saveState(BinaryFileWriter &binWriter)
void loadState(BinaryFileReader &binReader)
inline FORCE_INLINE Vector3r & getPosition0 (const unsigned int i)
inline FORCE_INLINE const Vector3r & getPosition0 (const unsigned int i) const
```

```
inline FORCE_INLINE void setPosition0 (const unsigned int i, const Vector3r &pos)
inline FORCE_INLINE Vector3r & getPosition (const unsigned int i)
inline FORCE_INLINE const Vector3r & getPosition (const unsigned int i) const
inline FORCE_INLINE void setPosition (const unsigned int i, const Vector3r &pos)
inline FORCE_INLINE Vector3r & getVelocity (const unsigned int i)
inline FORCE_INLINE const Vector3r & getVelocity (const unsigned int i) const
inline FORCE_INLINE void setVelocity (const unsigned int i, const Vector3r &vel)
inline FORCE_INLINE Vector3r & getVelocity0 (const unsigned int i)
inline FORCE_INLINE const Vector3r & getVelocity0 (const unsigned int i) const
inline FORCE_INLINE void setVelocity0 (const unsigned int i, const Vector3r &vel)
inline FORCE_INLINE Vector3r & getAcceleration (const unsigned int i)
inline FORCE_INLINE const Vector3r & getAcceleration (const unsigned int i) const
inline FORCE_INLINE void setAcceleration (const unsigned int i, const Vector3r &accel)
inline FORCE_INLINE const Real getMass (const unsigned int i) const
inline FORCE_INLINE Real & getMass (const unsigned int i)
inline FORCE_INLINE void setMass (const unsigned int i, const Real mass)
inline FORCE_INLINE const Real & getDensity (const unsigned int i) const
inline FORCE_INLINE Real & getDensity (const unsigned int i)
inline FORCE_INLINE void setDensity (const unsigned int i, const Real &val)
inline FORCE_INLINE unsigned int & getParticleId (const unsigned int i)
inline FORCE_INLINE const unsigned int & getParticleId (const unsigned int i) const
inline FORCE_INLINE unsigned int & getObjectId (const unsigned int i)
inline FORCE_INLINE const unsigned int & getObjectId (const unsigned int i) const
inline FORCE_INLINE void setObjectId (const unsigned int i, const unsigned int val)
inline FORCE_INLINE const ParticleState & getParticleState (const unsigned int i) const
inline FORCE_INLINE ParticleState & getParticleState (const unsigned int i)
inline FORCE_INLINE void setParticleState (const unsigned int i, const ParticleState &
inline FORCE_INLINE const Real getVolume (const unsigned int i) const
inline FORCE_INLINE Real & getVolume (const unsigned int i)
```

## Public Static Attributes

```
static int NUM_PARTICLES = -1  
static int NUM_REUSE_PARTICLES = -1  
static int DENSITY0 = -1  
static int DRAG_METHOD = -1  
static int SURFACE_TENSION_METHOD = -1  
static int VISCOSITY_METHOD = -1  
static int VORTICITY_METHOD = -1  
static int ELASTICITY_METHOD = -1
```

## Protected Functions

```
virtual void initParameters()  
void initMasses()  
  
virtual void resizeFluidParticles(const unsigned int newSize)  
    Resize the arrays containing the particle data.  
  
virtual void releaseFluidParticles()  
    Release the arrays containing the particle data.
```

## Protected Attributes

```
std::string m_id  
EmitterSystem *m_emitterSystem  
std::vector<Real> m_masses  
std::vector<Vector3r> m_a  
std::vector<Vector3r> m_v0  
std::vector<Vector3r> m_x0  
std::vector<Vector3r> m_x  
std::vector<Vector3r> m_v  
std::vector<Real> m_density  
std::vector<unsigned int> m_particleId  
std::vector<unsigned int> m_objectId  
std::vector<unsigned int> m_objectId0  
std::vector<ParticleState> m_particleState  
  
Real m_V  
unsigned int m_surfaceTensionMethod  
SurfaceTensionBase *m_surfaceTension  
unsigned int m_viscosityMethod
```

```

ViscosityBase *m_viscosity
unsigned int m_vorticityMethod
VorticityBase *m_vorticity
unsigned int m_dragMethod
DragBase *m_drag
unsigned int m_elasticityMethod
ElasticityBase *m_elasticity
std::vector<FieldDescription> m_fields
std::function<void ()> m_dragMethodChanged
std::function<void ()> m_surfaceTensionMethodChanged
std::function<void ()> m_viscosityMethodChanged
std::function<void ()> m_vorticityMethodChanged
std::function<void ()> m_elasticityMethodChanged
Real m_density0
unsigned int m_pointSetIndex
unsigned int m_numActiveParticles
unsigned int m_numActiveParticles0

```

## Class GaussQuadrature

- Defined in file\_SPlisHSPlasH\_Utilsities\_GaussQuadrature.h

## Class Documentation

```
class SPH::GaussQuadrature
```

### Public Types

```

using Integrand = std::function<double (Eigen::Vector3d const&)>
using Domain = Eigen::AlignedBox3d

```

### Public Static Functions

```

static double integrate (Integrand integrand, Domain const &domain, unsigned int p)
static void exportSamples (unsigned int p)

```

## Class JacobiPreconditioner1D

- Defined in file\_SPlisHSPlasH\_Utilsies\_MatrixFreeSolver.h

### Class Documentation

```
class SPH::JacobiPreconditioner1D
Matrix-free Jacobi preconditioner
```

#### Public Types

```
enum [anonymous]
Values:
enumerator ColsAtCompileTime
enumerator MaxColsAtCompileTime
typedef SystemMatrixType::StorageIndex StorageIndex
typedef void (*DiagonalMatrixElementFct)(const unsigned int, Real&, void*)
```

#### Public Functions

```
inline JacobiPreconditioner1D()
inline void init(const unsigned int dim, DiagonalMatrixElementFct fct, void *userData)
inline Eigen::Index rows() const
inline Eigen::Index cols() const
inline Eigen::ComputationInfo info()
template<typename MatType>
inline JacobiPreconditioner1D &analyzePattern(const MatType&)
template<typename MatType>
inline JacobiPreconditioner1D &factorize(const MatType &mat)
template<typename MatType>
inline JacobiPreconditioner1D &compute(const MatType &mat)
template<typename Rhs, typename Dest>
inline void _solve_impl(const Rhs &b, Dest &x) const
template<typename Rhs>
inline const Eigen::Solve<JacobiPreconditioner1D, Rhs> solve(const
Eigen::MatrixBase<Rhs> &b)
const
```

## Protected Attributes

```
unsigned int m_dim
DiagonalMatrixElementFct m_diagonalElementFct
    diagonal matrix element callback
void *m(userData
VectorXr m_invDiag
```

## Class JacobiPreconditioner3D

- Defined in file\_SPlisHSPlasH\_Utils\_MatrixFreeSolver.h

### Class Documentation

```
class SPH::JacobiPreconditioner3D
Matrix-free Jacobi preconditioner
```

#### Public Types

```
enum [anonymous]
Values:
enumerator ColsAtCompileTime
enumerator MaxColsAtCompileTime
typedef SystemMatrixType::StorageIndex StorageIndex
typedef void (*DiagonalMatrixElementFct)(const unsigned int, Vector3r&, void*)
```

#### Public Functions

```
inline JacobiPreconditioner3D()
inline void init(const unsigned int dim, DiagonalMatrixElementFct fct, void *userData)
inline Eigen::Index rows() const
inline Eigen::Index cols() const
inline Eigen::ComputationInfo info()
template<typename MatType>
inline JacobiPreconditioner3D &analyzePattern(const MatType&)
template<typename MatType>
inline JacobiPreconditioner3D &factorize(const MatType &mat)
template<typename MatType>
inline JacobiPreconditioner3D &compute(const MatType &mat)
template<typename Rhs, typename Dest>
inline void _solve_impl(const Rhs &b, Dest &x) const
template<typename Rhs>
```

```
inline const Eigen::Solve<JacobiPreconditioner3D, Rhs> solve (const  
Eigen::MatrixBase<Rhs> &b)  
const
```

## Protected Attributes

```
unsigned int m_dim  
DiagonalMatrixElementFct m_diagonalElementFct  
diagonal matrix element callback  
void *m(userData  
VectorXr m_invDiag
```

## Class MathFunctions

- Defined in file\_SPlisHSPlasH\_Utilsies\_MathFunctions.h

## Class Documentation

```
class SPH::MathFunctions
```

### Public Static Functions

```
static void extractRotation (const Matrix3r &A, Quaternionr &q, const unsigned int max-  
Iter)
```

Implementation of the paper:

Matthias Müller, Jan Bender, Nuttapong Chentanez and Miles Macklin, “A Robust Method to Extract the Rotational Part of Deformations”, ACM SIGGRAPH Motion in Games, 2016

```
static void pseudoInverse (const Matrix3r &a, Matrix3r &res)
```

```
static void svdWithInversionHandling (const Matrix3r &A, Vector3r &sigma, Matrix3r  
&U, Matrix3r &VT)
```

Perform a singular value decomposition of matrix A:  $A = U * \sigma * V^T$ . This function returns two proper rotation matrices U and  $V^T$  which do not contain a reflection. Reflections are corrected by the inversion handling proposed by Irving et al. 2004.

```
static void eigenDecomposition (const Matrix3r &A, Matrix3r &eigenVecs, Vector3r &eigen-  
Vals)
```

```
static void jacobiRotate (Matrix3r &A, Matrix3r &R, int p, int q)
```

```
static void getOrthogonalVectors (const Vector3r &vec, Vector3r &x, Vector3r &y)
```

Returns two orthogonal vectors to vec which are also orthogonal to each other.

## Class MatrixReplacement

- Defined in file\_SPlisHSPlasH\_Utils\_MatrixFreeSolver.h

## Inheritance Relationships

### Base Type

- public Eigen::EigenBase< MatrixReplacement >

## Class Documentation

```
class SPH::MatrixReplacement : public Eigen::EigenBase<MatrixReplacement>
    Replacement of the matrix in the linear system which is required for a matrix-free solver.
```

### Public Types

```
enum [anonymous]
    Values:
        enumerator ColsAtCompileTime
        enumerator MaxColsAtCompileTime
        enumerator IsRowMajor

typedef Real Scalar
typedef Real RealScalar
typedef int StorageIndex
typedef void (*MatrixVecProdFct)(const Real*, Real*, void*)
```

### Public Functions

```
inline Index rows() const
inline Index cols() const
template<typename Rhs>
inline Eigen::Product<MatrixReplacement, Rhs, Eigen::AliasFreeProduct> operator*(const
    Eigen::MatrixBase<Rhs>
    &x)
const
inline MatrixReplacement (const unsigned int dim, MatrixVecProdFct fct, void *userData)
inline void *getUserData()
inline MatrixVecProdFct getMatrixVecProdFct()
```

### Protected Attributes

```
unsigned int m_dim
void *m(userData
MatrixVecProdFct m_matrixVecProdFct
matrix vector product callback
```

## Class MicropolarModel\_Bender2017

- Defined in file\_SPlisHSPlasH\_Vorticity\_MicropolarModel\_Bender2017.h

### Inheritance Relationships

#### Base Type

- public SPH::VorticityBase (*Class VorticityBase*)

### Class Documentation

```
class SPH::MicropolarModel_Bender2017 : public SPH::VorticityBase
```

This class implements the micropolar material model introduced by Bender et al. [BKKW17].

References:

- [BKKW17] Jan Bender, Dan Koschier, Tassilo Kugelstadt, and Marcel Weiler. A micropolar material model for turbulent SPH fluids. In ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '17. ACM, 2017. URL: <http://doi.acm.org/10.1145/3099564.3099578>

### Public Functions

```
MicropolarModel_Bender2017 (FluidModel *model)
virtual ~MicropolarModel_Bender2017 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
inline FORCE_INLINE const Vector3r & getAngularAcceleration (const unsigned int i) const
inline FORCE_INLINE Vector3r & getAngularAcceleration (const unsigned int i)
inline FORCE_INLINE void setAngularAcceleration (const unsigned int i, const Vector3r &val)
inline FORCE_INLINE const Vector3r & getAngularVelocity (const unsigned int i) const
inline FORCE_INLINE Vector3r & getAngularVelocity (const unsigned int i)
inline FORCE_INLINE void setAngularVelocity (const unsigned int i, const Vector3r &val)
```

## Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)
```

## Public Static Attributes

```
static int VISCOSITY_OMEGA = -1
static int INERTIA_INVERSE = -1
```

## Protected Functions

```
virtual void initParameters()
```

## Protected Attributes

```
std::vector<Vector3r> m_angularAcceleration
std::vector<Vector3r> m_omega
Real m_viscosityOmega
Real m_inertiaInverse
```

## Class NonPressureForceBase

- Defined in file\_SPlisHSPlasH\_NonPressureForceBase.h

## Inheritance Relationships

### Base Type

- public ParameterObject

### Derived Types

- public SPH::DragBase (*Class DragBase*)
- public SPH::ElasticityBase (*Class ElasticityBase*)
- public SPH::SurfaceTensionBase (*Class SurfaceTensionBase*)
- public SPH::ViscosityBase (*Class ViscosityBase*)
- public SPH::VorticityBase (*Class VorticityBase*)

## Class Documentation

```
class SPH::NonPressureForceBase : public ParameterObject
    Base class for all non-pressure force methods.

    Subclassed by SPH::DragBase, SPH::ElasticityBase, SPH::SurfaceTensionBase, SPH::ViscosityBase,
    SPH::VorticityBase
```

### Public Functions

```
NonPressureForceBase (FluidModel *model)
NonPressureForceBase (const NonPressureForceBase&) = delete
NonPressureForceBase &operator= (const NonPressureForceBase&) = delete
virtual ~NonPressureForceBase (void)
virtual void step () = 0
inline virtual void reset ()
inline virtual void performNeighborhoodSearchSort ()
inline virtual void emittedParticles (const unsigned int startIndex)
inline virtual void saveState (BinaryFileWriter &binWriter)
inline virtual void loadState (BinaryFileReader &binReader)
inline FluidModel *getModel ()
virtual void init ()
inline virtual void deferredInit ()
    This function is called after the simulation scene is loaded and all parameters are initialized. While reading
    a scene file several parameters can change. The deferred init function should initialize all values which
    depend on these parameters.
```

### Protected Attributes

```
FluidModel *m_model
```

## Class PoissonDiskSampling

- Defined in file\_SPlisHSPlasH\_Utilsities\_PoissonDiskSampling.h

### Nested Relationships

#### Nested Types

- Struct PoissonDiskSampling::CellPosHasher
- Struct PoissonDiskSampling::HashEntry
- Struct PoissonDiskSampling::InitialPointInfo

## Class Documentation

**class SPH::PoissonDiskSampling**

This class implements a Poisson disk sampling for the surface of 3D models.

### Public Functions

**PoissonDiskSampling()**

**void sampleMesh (const unsigned int numVertices, const Vector3r \*vertices, const unsigned int numFaces, const unsigned int \*faces, const Real minRadius, const unsigned int numTrials, unsigned int distanceNorm, std::vector<Vector3r> &samples)**

Performs the poisson sampling with the respective parameters. Compare [http://graphics.cs.umass.edu/pubs/sa\\_2010.pdf](http://graphics.cs.umass.edu/pubs/sa_2010.pdf)

#### Parameters

- **mesh** – mesh data of sampled body
- **vertices** – vertex data of sampled data
- **sampledVertices** – sampled vertices that will be returned
- **minRadius** – minimal distance of sampled vertices
- **numTestpointsPerFace** – # of generated test points per face of body
- **distanceNorm** – 0: euclidean norm, 1: approx geodesic distance
- **numTrials** – # of iterations used to find samples

### Public Static Functions

**static inline FORCE\_INLINE int floor (const Real v)**

**struct HashEntry**

Struct to store the hash entry (spatial hashing)

### Public Functions

**inline HashEntry()**

### Public Members

std::vector<unsigned int> **samples**

unsigned int **startIndex**

**struct InitialPointInfo**

Struct to store the information of the initial points.

## Public Members

CellPos **cP**  
*Vector3r* **pos**  
unsigned int **ID**

## Class Poly6Kernel

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

## Class Documentation

```
class SPH::Poly6Kernel
    Poly6 kernel.
```

### Public Static Functions

```
static inline Real getRadius()
static inline void setRadius(Real val)
static inline Real W(const Real r)
    W(r,h) = (315/(64 pi h^9))(h^2-|r|^2)^3 = (315/(64 pi h^9))(h^2-r*r)^3
static inline Real W(const Vector3r &r)
static inline Vector3r gradW(const Vector3r &r)
    grad(W(r,h)) = r(-945/(32 pi h^9))(h^2-|r|^2)^2 = r(-945/(32 pi h^9))(h^2-r*r)^2
static inline Real laplacianW(const Vector3r &r)
    laplacian(W(r,h)) = (-945/(32 pi h^9))(h^2-|r|^2)(-7|r|^2+3h^2) = (-945/(32 pi h^9))(h^2-r*r)(3 h^2-7 r*r)
static inline Real W_zero()
```

### Protected Static Attributes

```
static Real m_radius
static Real m_k
static Real m_l
static Real m_m
static Real m_W_zero
```

## Template Class PrecomputedKernel

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

### Class Documentation

```
template<typename KernelType, unsigned int resolution = 10000u>
class SPH::PrecomputedKernel
```

Precomputed kernel which is based on a lookup table as described by Bender and Koschier [BK15,BK17].

The lookup tables can be used in combination with any kernel.

References:

- [BK15] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '15, 147-155. New York, NY, USA, 2015. ACM. URL: <http://doi.acm.org/10.1145/2786784.2786796>
- [BK17] Jan Bender and Dan Koschier. Divergence-free SPH for incompressible and viscous fluids. IEEE Transactions on Visualization and Computer Graphics, 23(3):1193-1206, 2017. URL: <http://dx.doi.org/10.1109/TVCG.2016.2578335>

### Public Static Functions

```
static inline Real getRadius()
static inline void setRadius(Real val)
static inline Real W(const Vector3r &r)
static inline Real W(const Real r)
static inline Vector3r gradW(const Vector3r &r)
static inline Real W_zero()
```

### Protected Static Attributes

```
static Real m_W[resolution]
static Real m_gradW[resolution + 1]
static Real m_radius
static Real m_radius2
static Real m_invStepSize
static Real m_W_zero
```

## Class RegularSampling2D

- Defined in file\_SPlisHSPlasH\_Utilsies-RegularSampling2D.h

### Class Documentation

```
class SPH::RegularSampling2D
```

This class implements a per-triangle regular sampling for the surface of 3D models.

#### Public Functions

```
RegularSampling2D()
```

#### Public Static Functions

```
static void sampleMesh(const Matrix3r &rotation, const Vector3r &translation, const unsigned numVertices, const Vector3r *vertices, const unsigned int numFaces, const unsigned int *faces, const Real maxDistance, std::vector<Vector3r> &samples)
```

Performs the poisson sampling with the respective parameters. Compare [http://graphics.cs.umass.edu/pubs/sa\\_2010.pdf](http://graphics.cs.umass.edu/pubs/sa_2010.pdf)

##### Parameters

- rotation** – rotation of the mesh
- translation** – translation of the mesh
- numVertices** – number of mesh vertices
- vertices** – vertex data of sampled data
- numFaces** – number of faces in the mesh
- faces** – face data of sampled mesh
- maxDistance** – maximal distance of sampled vertices
- samples** – vector to store the samples

## Class RegularTriangleSampling

- Defined in file\_SPlisHSPlasH\_Utilsies-RegularTriangleSampling.h

### Class Documentation

```
class SPH::RegularTriangleSampling
```

This class implements a per-triangle regular sampling for the surface of 3D models.

## Public Functions

`RegularTriangleSampling()`

## Public Static Functions

```
static void sampleMesh (const unsigned int numVertices, const Vector3r *vertices, const unsigned int numFaces, const unsigned int *faces, const Real maxDistance, std::vector<Vector3r> &samples)
```

Performs the poisson sampling with the respective parameters. Compare [http://graphics.cs.umass.edu/pubs/sa\\_2010.pdf](http://graphics.cs.umass.edu/pubs/sa_2010.pdf)

### Parameters

- **numVertices** – number of mesh vertices
- **vertices** – vertex data of sampled data
- **numFaces** – number of faces in the mesh
- **faces** – face data of sampled mesh
- **maxDistance** – maximal distance of sampled vertices
- **samples** – vector to store the samples

## Class RigidBodyObject

- Defined in file\_SPlisHSPlasH\_RigidBodyObject.h

## Inheritance Relationships

### Derived Type

- public SPH::StaticRigidBody (*Class StaticRigidBody*)

## Class Documentation

`class SPH::RigidBodyObject`

Base class for rigid body objects.

Subclassed by `SPH::StaticRigidBody`

## Public Functions

```
inline RigidBodyObject ()
inline virtual ~RigidBodyObject ()
virtual bool isDynamic () const = 0
inline bool isAnimated () const
inline virtual void setIsAnimated (const bool b)
virtual Real const getMass () const = 0
```

```
virtual Vector3r const &getPosition() const = 0
virtual void setPosition(const Vector3r &x) = 0
virtual Vector3r getWorldSpacePosition() const = 0
virtual Vector3r const &getVelocity() const = 0
virtual void setVelocity(const Vector3r &v) = 0
virtual Quaternionr const &getRotation() const = 0
virtual void setRotation(const Quaternionr &q) = 0
virtual Matrix3r getWorldSpaceRotation() const = 0
virtual Vector3r const &getAngularVelocity() const = 0
virtual void setAngularVelocity(const Vector3r &v) = 0
virtual void addForce(const Vector3r &f) = 0
virtual void addTorque(const Vector3r &t) = 0
virtual void updateMeshTransformation() = 0
virtual const std::vector<Vector3r> &getVertices() const = 0
virtual const std::vector<Vector3r> &getVertexNormals() const = 0
virtual const std::vector<unsigned int> &getFaces() const = 0
```

### Protected Attributes

```
bool m_isAnimated
```

## Class SimpleQuadrature

- Defined in file\_SPlisHSPlasH\_Utilsities\_SimpleQuadrature.h

### Class Documentation

```
class SPH::SimpleQuadrature
```

#### Public Types

```
using Integrand = std::function<double (Eigen::Vector3d const&) >
using Domain = Eigen::AlignedBox3d
```

## Public Static Functions

```
static void determineSamplePointsInSphere (const double radius, unsigned int p)
static void determineSamplePointsInCircle (const double radius, unsigned int p)
static double integrate (Integrand integrand)
```

## Public Static Attributes

```
static std::vector<Eigen::Vector3d> m_samplePoints
static double m_volume = 0.0
```

## Class Simulation

- Defined in file\_SPlisHSPlasH\_Simulation.h

## Nested Relationships

### Nested Types

- Struct Simulation::FluidInfo*
- Struct Simulation::NonPressureForceMethod*

## Inheritance Relationships

### Base Type

- public ParameterObject

## Class Documentation

```
class SPH::Simulation : public ParameterObject
```

Class to manage the current simulation time and the time step size. This class is a singleton.

## Public Types

```
typedef PrecomputedKernel<CubicKernel, 10000> PrecomputedCubicKernel
```

## Public Functions

```
Simulation()
Simulation(const Simulation&) = delete
Simulation &operator=(const Simulation&) = delete
~Simulation()

void init(const Real particleRadius, const bool sim2D)

void deferredInit()
    This function is called after the simulation scene is loaded and all parameters are initialized. While reading a scene file several parameters can change. The deferred init function should initialize all values which depend on these parameters.

void reset()

void addFluidModel(const std::string &id, const unsigned int nFluidParticles, Vector3r *fluidParticles, Vector3r *fluidVelocities, unsigned int *fluidObjectIds, const unsigned int nMaxEmitterParticles)
inline FluidModel *getFluidModel(const unsigned int index)
inline FluidModel *getFluidModelFromPointSet(const unsigned int pointSetIndex)
inline const unsigned int numberOfFluidModels() const
void addBoundaryModel(BoundaryModel *bm)
inline BoundaryModel *getBoundaryModel(const unsigned int index)
inline BoundaryModel *getBoundaryModelFromPointSet(const unsigned int pointSetIndex)
inline const unsigned int numberOfBoundaryModels() const
void updateBoundaryVolume()

inline void addFluidInfo(FluidInfo &info)
inline std::vector<FluidInfo> &getFluidInfos()
inline FluidInfo &getFluidInfo(const unsigned int i)
inline AnimationFieldSystem *getAnimationFieldSystem()
inline BoundaryHandlingMethods getBoundaryHandlingMethod() const
inline void setBoundaryHandlingMethod(BoundaryHandlingMethods val)
inline int getKernel() const
void setKernel(int val)
inline int getGradKernel() const
void setGradKernel(int val)
inline int isSimulationInitialized() const
void setSimulationInitialized(int val)
inline FORCE_INLINE Real W_zero() const
inline FORCE_INLINE Real W(const Vector3r &r) const
inline FORCE_INLINE Vector3r gradW(const Vector3r &r)
```

```

inline int getSimulationMethod() const
void setSimulationMethod(const int val)
void setSimulationMethodChangedCallback(std::function<void()
    > const &callBackFct
inline TimeStep *getTimeStep()
inline bool is2DSimulation()
inline bool zSortEnabled()
void initKernels()
void setParticleRadius(Real val)
inline Real getParticleRadius() const
inline Real getSupportRadius() const
void updateTimeStepSize()
    Update time step size depending on the chosen method.
void updateTimeStepSizeCFL()
    Update time step size by CFL condition.
virtual void performNeighborhoodSearch()
    Perform the neighborhood search for all fluid particles.
void performNeighborhoodSearchSort()
void computeNonPressureForces()
void animateParticles()
void emitParticles()
virtual void emittedParticles(FluidModel *model, const unsigned int startIndex)
inline NeighborhoodSearch *getNeighborhoodSearch()
inline void setCachePath(const std::string &cachePath)
inline const std::string &getCachePath() const
inline void setUseCache(const bool useCache)
inline const bool getUseCache() const
void saveState(BinaryFileWriter &binWriter)
void loadState(BinaryFileReader &binReader)
inline void addDragMethod(const std::string &name, const
    std::function<NonPressureForceBase*> FluidModel* > &creator
inline std::vector<NonPressureForceMethod> &getDragMethods()
inline void addElasticityMethod(const std::string &name, const
    std::function<NonPressureForceBase*> FluidModel* > &creator
inline std::vector<NonPressureForceMethod> &getElasticityMethods()
inline void addSurfaceTensionMethod(const std::string &name, const
    std::function<NonPressureForceBase*> FluidModel* > &creator

```

```
inline std::vector<NonPressureForceMethod> &getSurfaceTensionMethods()

inline void addViscosityMethod(const std::string &name, const
                               std::function<NonPressureForceBase*) FluidModel*
> &creator

inline std::vector<NonPressureForceMethod> &getViscosityMethods()

inline void addVorticityMethod(const std::string &name, const
                               std::function<NonPressureForceBase*) FluidModel*
> &creator

inline std::vector<NonPressureForceMethod> &getVorticityMethods()

inline FORCE_INLINE unsigned int numberOfPointSets () const

inline FORCE_INLINE unsigned int numberOfNeighbors (const unsigned int pointSetIndex, const
                                                    unsigned int neighborIndex) const

inline FORCE_INLINE unsigned int getNeighbor (const unsigned int pointSetIndex, const
                                              unsigned int neighborIndex) const

inline FORCE_INLINE const unsigned int * getNeighborList (const unsigned int pointSetIndex,
                                                       const unsigned int neighborIndex) const
```

### Public Static Functions

```
static Simulation *getCurrent()

static void setCurrent (Simulation *tm)

static bool hasCurrent()
```

### Public Static Attributes

```
static int SIM_2D = -1

static int PARTICLE_RADIUS = -1

static int GRAVITATION = -1

static int CFL_METHOD = -1

static int CFL_FACTOR = -1

static int CFL_MIN_TIMESTEPSIZE = -1

static int CFL_MAX_TIMESTEPSIZE = -1

static int ENABLE_Z_SORT = -1

static int KERNEL_METHOD = -1

static int GRAD_KERNEL_METHOD = -1

static int ENUM_KERNEL_CUBIC = -1

static int ENUM_KERNEL_WENDLANDQUINTICC2 = -1

static int ENUM_KERNEL_POLY6 = -1

static int ENUM_KERNEL_SPIKY = -1

static int ENUM_KERNEL_PRECOMPUTED_CUBIC = -1

static int ENUM_KERNEL_CUBIC_2D = -1

static int ENUM_KERNEL_WENDLANDQUINTICC2_2D = -1

static int ENUM_GRADKERNEL_CUBIC = -1
```

```

static int ENUM_GRADKERNEL_WENDLANDQUINTICC2 = -1
static int ENUM_GRADKERNEL_POLY6 = -1
static int ENUM_GRADKERNEL_SPIKY = -1
static int ENUM_GRADKERNEL_PRECOMPUTED_CUBIC = -1
static int ENUM_GRADKERNEL_CUBIC_2D = -1
static int ENUM_GRADKERNEL_WENDLANDQUINTICC2_2D = -1
static int SIMULATION_METHOD = -1
static int ENUM_CFL_NONE = -1
static int ENUM_CFL_STANDARD = -1
static int ENUM_CFL_ITER = -1
static int ENUM_SIMULATION_WCSPH = -1
static int ENUM_SIMULATION_PCISPH = -1
static int ENUM_SIMULATION_PBF = -1
static int ENUM_SIMULATION_IISPH = -1
static int ENUM_SIMULATION_DFSPH = -1
static int ENUM_SIMULATION_PF = -1
static int ENUM_SIMULATION_ICSPH = -1
static int BOUNDARY_HANDLING_METHOD = -1
static int ENUM_AKINCI2012 = -1
static int ENUM_KOSCHIER2017 = -1
static int ENUM_BENDER2019 = -1

```

### Protected Functions

```

virtual void initParameters()
void registerNonpressureForces()

```

### Protected Attributes

```

std::vector<FluidModel*> m_fluidModels
std::vector<BoundaryModel*> m_boundaryModels
std::vector<FluidInfo> m_fluidInfos
NeighborhoodSearch *m_neighborhoodSearch
AnimationFieldSystem *m_animationFieldSystem
int m_cflMethod
Real m_cflFactor
Real m_cflMinTimeStepSize
Real m_cflMaxTimeStepSize

```

```
int m_kernelMethod
int m_gradKernelMethod
Real m_W_zero
Real (*m_kernelFct)(const Vector3r&)
Vector3r (*m_gradKernelFct)(const Vector3r &r)
SimulationMethods m_simulationMethod
TimeStep *m_timeStep
Vector3r m_gravitation
Real m_particleRadius
Real m_supportRadius
bool m_sim2D
bool m_enableZSort
std::function<void ()> m_simulationMethodChanged
int m_boundaryHandlingMethod
std::string m_cachePath
bool m_useCache
std::vector<NonPressureForceMethod> m_dragMethods
std::vector<NonPressureForceMethod> m_elasticityMethods
std::vector<NonPressureForceMethod> m_surfaceTensionMethods
std::vector<NonPressureForceMethod> m_vorticityMethods
std::vector<NonPressureForceMethod> m_viscoMethods
bool m_simulationIsInitialized
struct FluidInfo
    Fluid object information
```

## Public Functions

```
inline bool hasSameParticleSampling(const FluidInfo &other)
```

## Public Members

```
int type
int numParticles
AlignedBox3r box
std::string id
std::string samplesFile
std::string visMeshFile
Vector3r translation
```

```

Matrix3r rotation
Vector3r scale
Vector3r initialVelocity
Vector3r initialAngularVelocity
unsigned char mode
bool invert
std::array<unsigned int, 3> resolutionSDF
unsigned int emitter_width
unsigned int emitter_height
Real emitter_velocity
Real emitter_emitStartTime
Real emitter_emitEndTime
unsigned int emitter_type
struct NonPressureForceMethod

```

### Public Members

```

std::string m_name
std::function<NonPressureForceBase* (FluidModel*)> m_creator
int m_id

```

## Class SimulationDataDFSPH

- Defined in file \_SPlisHSPlasH\_DFSRH\_SimulationDataDFSPH.h

### Class Documentation

```
class SPH::SimulationDataDFSPH
```

*Simulation* data which is required by the method Divergence-free Smoothed Particle Hydrodynamics introduced by Bender and Koschier [BK15,BK17].

References:

- [BK15] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '15, 147-155. New York, NY, USA, 2015. ACM. URL: <http://doi.acm.org/10.1145/2786784.2786796>
- [BK17] Jan Bender and Dan Koschier. Divergence-free SPH for incompressible and viscous fluids. IEEE Transactions on Visualization and Computer Graphics, 23(3):1193-1206, 2017. URL: <http://dx.doi.org/10.1109/TVCG.2016.2578335>

## Public Functions

```
SimulationDataDFSPH()
virtual ~SimulationDataDFSPH()

virtual void init()
    Initialize the arrays containing the particle data.

virtual void cleanup()
    Release the arrays containing the particle data.

virtual void reset()
    Reset the particle data.

void performNeighborhoodSearchSort()
    Important: First call m_model->performNeighborhoodSearchSort() to call the z_sort of the neighborhood search.

void emittedParticles (FluidModel *model, const unsigned int startIndex)

inline FORCE_INLINE const Real getFactor (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE Real & getFactor (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE void setFactor (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE const Real getKappa (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE Real & getKappa (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE void setKappa (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE const Real getKappaV (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE Real & getKappaV (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE void setKappaV (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE const Real getDensityAdv (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE Real & getDensityAdv (const unsigned int fluidIndex, const unsigned int i)
inline FORCE_INLINE void setDensityAdv (const unsigned int fluidIndex, const unsigned int i)
```

## Protected Attributes

std::vector<std::vector<*Real*>> **m\_factor**  
factor  $\alpha_i$

std::vector<std::vector<*Real*>> **m\_kappa**  
stores  $\kappa$  value of last time step for a warm start of the pressure solver

std::vector<std::vector<*Real*>> **m\_kappaV**  
stores  $\kappa^v$  value of last time step for a warm start of the divergence solver

std::vector<std::vector<*Real*>> **m\_density\_adv**  
adverted density

## Class SimulationDataICSPH

- Defined in file\_SPlisHSPlasH\_ICSPH\_SimulationDataICSPH.h

### Class Documentation

**class SPH::SimulationDataICSPH**

*Simulation* data which is required by the method Implicit Compressible SPH introduced by Gissler et al. [GHB+20].

References:

- [GHB+20] Christoph Gissler, Andreas Henne, Stefan Band, Andreas Peer and Matthias Teschner. An Implicit Compressible SPH Solver for Snow *Simulation*. ACM Transactions on Graphics, 39(4). URL: <https://doi.org/10.1145/3386569.3392431>

### Public Functions

**SimulationDataICSPH()**

**virtual ~SimulationDataICSPH()**

**virtual void init()**

Initialize the arrays containing the particle data.

**virtual void cleanup()**

Release the arrays containing the particle data.

**virtual void reset()**

Reset the particle data.

**void performNeighborhoodSearchSort()**

Important: First call m\_model->*performNeighborhoodSearchSort()* to call the z\_sort of the neighborhood search.

**void emittedParticles (FluidModel \*model, const unsigned int startIndex)**

inline FORCE\_INLINE const Real getAii (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)

inline FORCE\_INLINE Real & getAii (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)

inline FORCE\_INLINE void setAii (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const Real value)

inline FORCE\_INLINE const Real getDensityAdv (const unsigned int fluidIndex, const unsigned int i)

inline FORCE\_INLINE Real & getDensityAdv (const unsigned int fluidIndex, const unsigned int i)

inline FORCE\_INLINE void setDensityAdv (const unsigned int fluidIndex, const unsigned int i, const Real value)

inline FORCE\_INLINE const Real getPressure (const unsigned int fluidIndex, const unsigned int i)

inline FORCE\_INLINE Real & getPressure (const unsigned int fluidIndex, const unsigned int i)

inline FORCE\_INLINE void setPressure (const unsigned int fluidIndex, const unsigned int i, const Real value)

inline FORCE\_INLINE Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int i)

inline FORCE\_INLINE const Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int i)

inline FORCE\_INLINE void setPressureAccel (const unsigned int fluidIndex, const unsigned int i, const Vector3r & value)

inline FORCE\_INLINE Vector3r & getPressureGradient (const unsigned int fluidIndex, const unsigned int i)

```
inline FORCE_INLINE const Vector3r & getPressureGradient (const unsigned int fluidIndex)
inline FORCE_INLINE void setPressureGradient (const unsigned int fluidIndex, const uns
```

### Protected Attributes

```
std::vector<std::vector<Real>> m_aai
std::vector<std::vector<Real>> m_density_adv
std::vector<std::vector<Real>> m_pressure
std::vector<std::vector<Vector3r>> m_pressureGradient
std::vector<std::vector<Vector3r>> m_pressureAccel
```

## Class SimulationDataIISPH

- Defined in file\_SPlisHSPlasH\_IISPH\_SimulationDataIISPH.h

### Class Documentation

```
class SPH::SimulationDataIISPH
```

*Simulation* data which is required by the method Implicit Incompressible SPH introduced by Ihmsen et al. [ICS+14].

References:

- [ICS+14] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible SPH. IEEE Transactions on Visualization and Computer Graphics, 20(3):426-435, March 2014. URL: <http://dx.doi.org/10.1109/TVCG.2013.105>

### Public Functions

```
SimulationDataIISPH()
```

```
virtual ~SimulationDataIISPH()
```

```
virtual void init()
```

Initialize the arrays containing the particle data.

```
virtual void cleanup()
```

Release the arrays containing the particle data.

```
virtual void reset()
```

Reset the particle data.

```
void performNeighborhoodSearchSort()
```

Important: First call m\_model->*performNeighborhoodSearchSort()* to call the z\_sort of the neighborhood search.

```
void emittedParticles (FluidModel *model, const unsigned int startIndex)
```

```
inline FORCE_INLINE const Real getAii (const unsigned int fluidIndex, const unsigned i
```

```
inline FORCE_INLINE Real & getAii (const unsigned int fluidIndex, const unsigned int i
```

```
inline FORCE_INLINE void setAii (const unsigned int fluidIndex, const unsigned int i,
```

```

inline FORCE_INLINE Vector3r & getDii (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE const Vector3r & getDii (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE void setDii (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const Vector3r & v)
inline FORCE_INLINE Vector3r & getDij_pj (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const unsigned int p)
inline FORCE_INLINE const Vector3r & getDij_pj (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const unsigned int p)
inline FORCE_INLINE void setDij_pj (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const unsigned int p, const Vector3r & v)
inline FORCE_INLINE const Real getDensityAdv (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE Real & getDensityAdv (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE void setDensityAdv (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const Real & v)
inline FORCE_INLINE const Real getPressure (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE Real & getPressure (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE void setPressure (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const Real & v)
inline FORCE_INLINE const Real getLastPressure (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE Real & getLastPressure (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE void setLastPressure (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const Real & v)
inline FORCE_INLINE Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE const Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int i, const unsigned int j)
inline FORCE_INLINE void setPressureAccel (const unsigned int fluidIndex, const unsigned int i, const unsigned int j, const Vector3r & v)

```

### Protected Attributes

```

std::vector<std::vector<Real>> m_aii
std::vector<std::vector<Vector3r>> m_dii
std::vector<std::vector<Vector3r>> m_dij_pj
std::vector<std::vector<Real>> m_density_adv
std::vector<std::vector<Real>> m_pressure
std::vector<std::vector<Real>> m_lastPressure
std::vector<std::vector<Vector3r>> m_pressureAccel

```

### Class SimulationDataPBF

- Defined in file\_SPlisHSPlasH\_PBF\_SimulationDataPBF.h

## Class Documentation

```
class SPH::SimulationDataPBF
```

*Simulation* data which is required by the method Position-Based Fluids introduced by Macklin and Mueller [MM13,BMO+14,BMM15].

References:

- [MM13] Miles Macklin and Matthias Müller. Position based fluids. ACM Trans. Graph., 32(4):104:1–104:12, July 2013. URL: <http://doi.acm.org/10.1145/2461912.2461984>
- [BMO+14] Jan Bender, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin. A survey on position-based simulation methods in computer graphics. Computer Graphics Forum, 33(6):228–251, 2014. URL: <http://dx.doi.org/10.1111/cgf.12346>
- [BMM15] Jan Bender, Matthias Müller, and Miles Macklin. Position-based simulation methods in computer graphics. In EUROGRAPHICS 2015 Tutorials. Eurographics Association, 2015. URL: <http://dx.doi.org/10.2312/egt.20151045>

## Public Functions

```
SimulationDataPBF()
```

```
virtual ~SimulationDataPBF()
```

```
virtual void init()
```

Initialize the arrays containing the particle data.

```
virtual void cleanup()
```

Release the arrays containing the particle data.

```
virtual void reset()
```

Reset the particle data.

```
void performNeighborhoodSearchSort()
```

Important: First call m\_model->*performNeighborhoodSearchSort()* to call the z\_sort of the neighborhood search.

```
void emittedParticles (FluidModel *model, const unsigned int startIndex)
```

```
inline FORCE_INLINE const Real & getLambda (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE Real & getLambda (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE void setLambda (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE Vector3r & getDeltaX (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE const Vector3r & getDeltaX (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE void setDeltaX (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE Vector3r & getLastPosition (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE const Vector3r & getLastPosition (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE void setLastPosition (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE Vector3r & getOldPosition (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE const Vector3r & getOldPosition (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE void setOldPosition (const unsigned int fluidIndex, const unsigned int)
```

## Protected Attributes

```
std::vector<std::vector<Real>> m_lambda
std::vector<std::vector<Vector3r>> m_deltaX
std::vector<std::vector<Vector3r>> m_oldX
std::vector<std::vector<Vector3r>> m_lastX
```

## Class SimulationDataPCISPH

- Defined in file\_SPlisHSPlasH\_PCISPH\_SimulationDataPCISPH.h

## Class Documentation

**class SPH::SimulationDataPCISPH**

*Simulation* data which is required by the method Predictive-corrective Incompressible SPH introduced by Solenthaler and Pajarola [SP09].

References:

- [SP09] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. ACM Trans. Graph., 28(3):40:1-40:6, July 2009. URL: <http://doi.acm.org/10.1145/1531326.1531346>

## Public Functions

```
SimulationDataPCISPH()
virtual ~SimulationDataPCISPH()
virtual void init()
    Initialize the arrays containing the particle data.
virtual void cleanup()
    Release the arrays containing the particle data.
virtual void reset()
    Reset the particle data.
void performNeighborhoodSearchSort()
    Important: First call m_model->performNeighborhoodSearchSort() to call the z_sort of the neighborhood search.
inline Real getPCISPH_ScalingFactor(const unsigned int fluidIndex)
void emittedParticles(FluidModel *model, const unsigned int startIndex)
inline FORCE_INLINE Vector3r & getPredictedPosition(const unsigned int fluidIndex, const Real densityAdv)
inline FORCE_INLINE const Vector3r & getPredictedPosition(const unsigned int fluidIndex, const Real densityAdv)
inline FORCE_INLINE void setPredictedPosition(const unsigned int fluidIndex, const Vector3r & predictedPosition)
inline FORCE_INLINE Vector3r & getPredictedVelocity(const unsigned int fluidIndex, const Real densityAdv)
inline FORCE_INLINE const Vector3r & getPredictedVelocity(const unsigned int fluidIndex, const Real densityAdv)
inline FORCE_INLINE void setPredictedVelocity(const unsigned int fluidIndex, const Vector3r & predictedVelocity)
inline FORCE_INLINE const Real getDensityAdv(const unsigned int fluidIndex, const Real densityAdv)
```

```
inline FORCE_INLINE Real & getDensityAdv (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID);
inline FORCE_INLINE void setDensityAdv (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID, Real density);
inline FORCE_INLINE const Real getPressure (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID);
inline FORCE_INLINE Real & getPressure (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID, Real pressure);
inline FORCE_INLINE void setPressure (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID, Real pressure);
inline FORCE_INLINE Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID, Vector3r pressureAccel);
inline FORCE_INLINE const Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID, Vector3r pressureAccel);
inline FORCE_INLINE void setPressureAccel (const unsigned int fluidIndex, const unsigned int particleIndex, const unsigned int particleType, const unsigned int particleID, const unsigned int subParticleID, Vector3r pressureAccel);
```

### Protected Attributes

```
std::vector<Real> m_pcisph_factor;
std::vector<std::vector<Vector3r>> m_predX;
std::vector<std::vector<Vector3r>> m_predV;
std::vector<std::vector<Real>> m_densityAdv;
std::vector<std::vector<Real>> m_pressure;
std::vector<std::vector<Vector3r>> m_pressureAccel;
```

## Class SimulationDataPF

- Defined in file\_SPlisHSPlasH\_PF\_SimulationDataPF.h

### Class Documentation

```
class SPH::SimulationDataPF
```

*Simulation* data which is required by the method Projective Fluids introduced by Weiler, Koschier and Bender [WKB16].

References:

- [WKB16] Marcel Weiler, Dan Koschier, and Jan Bender. Projective fluids. In Proceedings of the 9th International Conference on Motion in Games, MIG '16, 79-84. New York, NY, USA, 2016. ACM. URL: <http://doi.acm.org/10.1145/2994258.2994282>

### Public Functions

```
SimulationDataPF()
virtual ~SimulationDataPF()
virtual void init()
    Initialize the arrays containing the particle data.
virtual void cleanup()
    Release the arrays containing the particle data.
virtual void reset()
    Reset the particle data.
```

```

void performNeighborhoodSearchSort ()
Important: First call m_model->performNeighborhoodSearchSort() to call the z_sort of the neighborhood
search.

void emittedParticles (FluidModel *model, const unsigned int startIndex)
inline FORCE_INLINE const Vector3r getOldPosition (const unsigned int fluidIndex, const
inline FORCE_INLINE Vector3r & getOldPosition (const unsigned int fluidIndex, const un
inline FORCE_INLINE void setOldPosition (const unsigned int fluidIndex, const unsigned
inline FORCE_INLINE const unsigned int getNumFluidNeighbors (const unsigned int fluidIndex
inline FORCE_INLINE unsigned int & getNumFluidNeighbors (const unsigned int fluidIndex
inline FORCE_INLINE void setNumFluidNeighbors (const unsigned int fluidIndex, const un
inline FORCE_INLINE const Vector3r & getS (const unsigned int fluidIndex, const unsigned
inline FORCE_INLINE Vector3r & getS (const unsigned int fluidIndex, const unsigned int
inline FORCE_INLINE void setS (const unsigned int fluidIndex, const unsigned int i, co
inline FORCE_INLINE const Vector3r & getDiag (const unsigned int fluidIndex, const uns
inline FORCE_INLINE Vector3r & getDiag (const unsigned int fluidIndex, const unsigned
inline FORCE_INLINE void setDiag (const unsigned int fluidIndex, const unsigned int i,
inline FORCE_INLINE const unsigned int & getParticleOffset (const unsigned int fluidIndex

```

## Protected Attributes

**std::vector<std::vector<*Vector3r*>> m\_old\_position**  
 particle position from last timestep

**std::vector<std::vector<unsigned int>> m\_num\_fluid\_neighbors**  
 number of neighbors that are fluid particles

**std::vector<std::vector<*Vector3r*>> m\_s**  
 positions predicted from momentum

**std::vector<std::vector<*Vector3r*>> m\_mat\_diag**  
 diagonal of system matrix, used by preconditioner

**std::vector<unsigned int> m\_particleOffset**

## Class SimulationDataWCSPH

- Defined in file\_SPlisHSPlasH\_WCSPH\_SimulationDataWCSPH.h

## Class Documentation

```
class SPH::SimulationDataWCSPH
```

*Simulation* data which is required by the method Weakly Compressible SPH for Free Surface Flows introduced by Becker and Teschner [BT07].

References:

- [BT07] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. In ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '07, 209-217. Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. URL: <http://dl.acm.org/citation.cfm?id=1272690.1272719>

## Public Functions

```
SimulationDataWCSPH()
```

```
virtual ~SimulationDataWCSPH()
```

```
virtual void init()
```

Initialize the arrays containing the particle data.

```
virtual void cleanup()
```

Release the arrays containing the particle data.

```
virtual void reset()
```

Reset the particle data.

```
void performNeighborhoodSearchSort()
```

Important: First call `m_model->performNeighborhoodSearchSort()` to call the z\_sort of the neighborhood search.

```
void emittedParticles (FluidModel *model, const unsigned int startIndex)
```

```
inline FORCE_INLINE const Real getPressure (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE Real & getPressure (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE void setPressure (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE const Vector3r & getPressureAccel (const unsigned int fluidIndex, const unsigned int)
```

```
inline FORCE_INLINE void setPressureAccel (const unsigned int fluidIndex, const unsigned int)
```

## Protected Attributes

```
std::vector<std::vector<Real>> m_pressure
```

```
std::vector<std::vector<Vector3r>> m_pressureAccel
```

## Class SpikyKernel

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

### Class Documentation

```
class SPH::SpikyKernel
    Spiky kernel.
```

#### Public Static Functions

```
    static inline Real getRadius()
    static inline void setRadius(Real val)
    static inline Real W(const Real r)
         $W(r,h) = 15/(\pi \cdot h^6) \cdot (h-r)^3$ 
    static inline Real W(const Vector3r &r)
    static inline Vector3r gradW(const Vector3r &r)
         $\text{grad}(W(r,h)) = -r(45/(\pi \cdot h^6) \cdot (h-r)^2)$ 
    static inline Real W_zero()
```

#### Protected Static Attributes

```
    static Real m_radius
    static Real m_k
    static Real m_l
    static Real m_W_zero
```

## Class StaticRigidBody

- Defined in file\_SPlisHSPlasH\_StaticRigidBody.h

### Inheritance Relationships

#### Base Type

- public SPH::RigidBodyObject (*Class RigidBodyObject*)

## Class Documentation

```
class SPH::StaticRigidBody : public SPH::RigidBodyObject
```

This class stores the information of a static rigid body which is not part of a rigid body simulation.

### Public Functions

```
inline StaticRigidBody()

inline virtual bool isDynamic() const

inline virtual Real const getMass() const

inline virtual Vector3r const &getPosition() const

inline virtual void setPosition(const Vector3r &x)

inline Vector3r const &getPosition0() const

inline void setPosition0(const Vector3r &x)

inline virtual Vector3r getWorldSpacePosition() const

inline virtual Vector3r const &getVelocity() const

inline virtual void setVelocity(const Vector3r &v)

inline virtual Quaternionr const &getRotation() const

inline virtual void setRotation(const Quaternionr &q)

inline Quaternionr const &getRotation0() const

inline void setRotation0(const Quaternionr &q)

inline virtual Matrix3r getWorldSpaceRotation() const

inline virtual Vector3r const &getAngularVelocity() const

inline virtual void setAngularVelocity(const Vector3r &v)

inline virtual void addForce(const Vector3r &f)

inline virtual void addTorque(const Vector3r &t)

inline void animate()

inline virtual const std::vector<Vector3r> &getVertices() const

inline virtual const std::vector<Vector3r> &getVertexNormals() const

inline virtual const std::vector<unsigned int> &getFaces() const

inline void setWorldSpacePosition(const Vector3r &x)

inline void setWorldSpaceRotation(const Matrix3r &r)

inline TriangleMesh &getGeometry()

inline virtual void updateMeshTransformation()

inline void reset()
```

## Protected Attributes

*Vector3r* **m\_x0**  
*Vector3r* **m\_x**  
*Quaternionr* **m\_q**  
*Quaternionr* **m\_q0**  
*Vector3r* **m\_velocity**  
*Vector3r* **m\_angularVelocity**  
*TriangleMesh* **m\_geometry**

## Class SurfaceTension\_Akinci2013

- Defined in file\_SPlisHSPlasH\_SurfaceTension\_SurfaceTension\_Akinci2013.h

## Inheritance Relationships

### Base Type

- public SPH::SurfaceTensionBase (*Class SurfaceTensionBase*)

## Class Documentation

**class** SPH::SurfaceTension\_Akinci2013 : **public** SPH::SurfaceTensionBase  
This class implements the surface tension method introduced by Akinci et al. [ATT13].

References:

- [AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. ACM Trans. Graph., 32(6):182:1-182:8, November 2013. URL: <http://doi.acm.org/10.1145/2508363.2508395>

## Public Functions

```
SurfaceTension_Akinci2013 (FluidModel *model)
virtual ~SurfaceTension_Akinci2013 (void)
virtual void step ()
virtual void reset ()
void computeNormals ()
virtual void performNeighborhoodSearchSort ()
inline FORCE_INLINE Vector3r & getNormal (const unsigned int i)
inline FORCE_INLINE const Vector3r & getNormal (const unsigned int i) const
inline FORCE_INLINE void setNormal (const unsigned int i, const Vector3r &val)
```

## Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)
```

## Protected Attributes

```
std::vector<Vector3r> m_normals
```

## Class SurfaceTension\_Becker2007

- Defined in file\_SPlisHSPlasH\_SurfaceTension\_SurfaceTension\_Becker2007.h

## Inheritance Relationships

### Base Type

- public SPH::SurfaceTensionBase (*Class SurfaceTensionBase*)

## Class Documentation

```
class SPH::SurfaceTension_Becker2007 : public SPH::SurfaceTensionBase
```

This class implements the surface tension method introduced by Becker and Teschner [BT07].

References:

- [BT07] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. In ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '07, 209-217. Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. URL: <http://dl.acm.org/citation.cfm?id=1272690.1272719>

## Public Functions

```
SurfaceTension_Becker2007 (FluidModel *model)  
virtual ~SurfaceTension_Becker2007 (void)  
virtual void step ()  
virtual void reset ()
```

## Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)
```

## Class SurfaceTension\_He2014

- Defined in file\_SPlisHSPlasH\_SurfaceTension\_SurfaceTension\_He2014.h

### Inheritance Relationships

#### Base Type

- public SPH::SurfaceTensionBase (*Class SurfaceTensionBase*)

### Class Documentation

```
class SPH::SurfaceTension_He2014 : public SPH::SurfaceTensionBase
```

This class implements the surface tension method introduced by He et al. [HWZ+14].

References:

- [HWZ+14] Xiaowei He, Huamin Wang, Fengjun Zhang, Hongan Wang, Guoping Wang, and Kun Zhou. Robust simulation of sparsely sampled thin features in SPH-based free surface flows. ACM Trans. Graph., 34(1):7:1-7:9, December 2014. URL: <http://doi.acm.org/10.1145/2682630>

#### Public Functions

```
SurfaceTension_He2014 (FluidModel *model)
virtual ~SurfaceTension_He2014 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
inline FORCE_INLINE const Real getColor (const unsigned int i) const
inline FORCE_INLINE Real & getColor (const unsigned int i)
inline FORCE_INLINE void setColor (const unsigned int i, const Real p)
inline FORCE_INLINE const Real getGradC2 (const unsigned int i) const
inline FORCE_INLINE Real & getGradC2 (const unsigned int i)
inline FORCE_INLINE void setGradC2 (const unsigned int i, const Real p)
```

#### Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
```

## Protected Attributes

```
std::vector<Real> m_color  
std::vector<Real> m_gradC2
```

## Class SurfaceTension\_ZorillaRitter2020

- Defined in file\_SPlisHSPlasH\_SurfaceTension\_SurfaceTension\_ZorillaRitter2020.h

## Inheritance Relationships

### Base Type

- public SPH::SurfaceTensionBase (*Class SurfaceTensionBase*)

## Class Documentation

```
class SPH::SurfaceTension_ZorillaRitter2020 : public SPH::SurfaceTensionBase
```

This class implements the surface tension method introduced by Zorilla, Ritter, Sappl, Rauch, Harders: extended version 2020: <https://doi.org/10.3390/computers9020023> and original version 2019: <https://diglib.eg.org/handle/10.2312/cgvc20191260>

### Public Functions

```
SurfaceTension_ZorillaRitter2020 (FluidModel *model)  
virtual ~SurfaceTension_ZorillaRitter2020 (void)  
virtual void performNeighborhoodSearchSort () override
```

### Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)  
static bool classifySurfaceParticle (double com, int non, double d_offset = 0.0)
```

Linear classifier. Divides into surface or non-surface particle. The function is equivalent to the network classifier. Also, inspect lines 344 to 348 in the cpp file for how to compute the required input.

#### Parameters

- com** – normalized center of mass / number of neighbors
- non** – number of neighbors
- d\_offset** – constant parameter d

**Returns** true if surface, false otherwise

## Class SurfaceTensionBase

- Defined in file\_SPlisHSPlasH\_SurfaceTension\_SurfaceTensionBase.h

### Inheritance Relationships

#### Base Type

- public SPH::NonPressureForceBase (*Class NonPressureForceBase*)

#### Derived Types

- public SPH::SurfaceTension\_Akinci2013 (*Class SurfaceTension\_Akinci2013*)
- public SPH::SurfaceTension\_Becker2007 (*Class SurfaceTension\_Becker2007*)
- public SPH::SurfaceTension\_He2014 (*Class SurfaceTension\_He2014*)
- public SPH::SurfaceTension\_ZorillaRitter2020 (*Class SurfaceTension\_ZorillaRitter2020*)

### Class Documentation

```
class SPH::SurfaceTensionBase : public SPH::NonPressureForceBase
```

Base class for all surface tension methods.

Subclassed by *SPH::SurfaceTension\_Akinci2013*, *SPH::SurfaceTension\_Becker2007*,  
*SPH::SurfaceTension\_He2014*, *SPH::SurfaceTension\_ZorillaRitter2020*

#### Public Functions

```
SurfaceTensionBase (FluidModel *model)  

virtual ~SurfaceTensionBase (void)
```

#### Public Static Attributes

```
static int SURFACE_TENSION = -1  

static int SURFACE_TENSION_BOUNDARY = -1
```

#### Protected Functions

```
virtual void initParameters ()
```

## Protected Attributes

*Real* **m\_surfaceTension**  
*Real* **m\_surfaceTensionBoundary**

## Class TimeIntegration

- Defined in file\_SPlisHSPlasH\_PBF\_TimeIntegration.h

## Class Documentation

**class SPH::TimeIntegration**

Class for the position-based fluids time integration.

### Public Static Functions

**static void semiImplicitEuler (const Real h, const Real mass, Vector3r &position, Vector3r &velocity, const Vector3r &acceleration)**

Perform an integration step for a particle using the semi-implicit Euler (symplectic Euler) method:

$$\begin{aligned}\mathbf{v}(t + h) &= \mathbf{v}(t) + \mathbf{a}(t)h \\ \mathbf{x}(t + h) &= \mathbf{x}(t) + \mathbf{v}(t + h)h\end{aligned}$$

#### Parameters

- h** – time step size
- mass** – mass of the particle
- position** – position of the particle
- velocity** – velocity of the particle
- acceleration** – acceleration of the particle

**static void velocityUpdateFirstOrder (const Real h, const Real mass, const Vector3r &position, const Vector3r &velocity, const Vector3r &oldPosition, Vector3r &velocity)**

Perform a velocity update (first order) for the linear velocity:

$$\mathbf{v}(t + h) = \frac{1}{h}(\mathbf{p}(t + h) - \mathbf{p}(t))$$

#### Parameters

- h** – time step size
- mass** – mass of the particle
- position** – new position  $\mathbf{p}(t + h)$  of the particle
- oldPosition** – position  $\mathbf{p}(t)$  of the particle before the time step
- velocity** – resulting velocity of the particle

---

```
static void velocityUpdateSecondOrder (const Real h, const Real mass, const Vector3r &position, const Vector3r &oldPosition, const Vector3r &positionOfLastStep, Vector3r &velocity)
```

## Class TimeManager

- Defined in file\_SPlisHSPlasH\_TimeManager.h

### Class Documentation

```
class SPH::TimeManager
```

Class to manage the current simulation time and the time step size. This class is a singleton.

#### Public Functions

```
TimeManager ()  

~TimeManager ()  

Real getTime ()  

void setTime (Real t)  

Real getTimeStepSize ()  

void setTimeStepSize (Real tss)  

void saveState (BinaryFileWriter &binWriter)  

void loadState (Binary.FileReader &binReader)
```

#### Public Static Functions

```
static TimeManager *getCurrent ()  

static void setCurrent (TimeManager *tm)  

static bool hasCurrent ()
```

## Class TimeStep

- Defined in file\_SPlisHSPlasH\_TimeStep.h

### Inheritance Relationships

#### Base Type

- public ParameterObject

## Derived Types

- public SPH::TimeStepDFSPH (*Class TimeStepDFSPH*)
- public SPH::TimeStepICSPH (*Class TimeStepICSPH*)
- public SPH::TimeStepIISPH (*Class TimeStepIISPH*)
- public SPH::TimeStepPBF (*Class TimeStepPBF*)
- public SPH::TimeStepPCISPH (*Class TimeStepPCISPH*)
- public SPH::TimeStepPF (*Class TimeStepPF*)
- public SPH::TimeStepWCSPH (*Class TimeStepWCSPH*)

## Class Documentation

```
class SPH::TimeStep : public ParameterObject
```

Base class for the simulation methods.

Subclassed by *SPH::TimeStepDFSPH*, *SPH::TimeStepICSPH*, *SPH::TimeStepIISPH*, *SPH::TimeStepPBF*, *SPH::TimeStepPCISPH*, *SPH::TimeStepPF*, *SPH::TimeStepWCSPH*

### Public Functions

```
TimeStep()
```

```
virtual ~TimeStep(void)
```

```
void computeDensities(const unsigned int fluidModelIndex)
```

Determine densities of all fluid particles.

```
virtual void step() = 0
```

```
virtual void reset()
```

```
virtual void init()
```

```
virtual void resize() = 0
```

```
inline virtual void emittedParticles(FluidModel *model, const unsigned int startIndex)
```

```
inline virtual void saveState(BinaryFileWriter &binWriter)
```

```
inline virtual void loadState(BinaryFileReader &binReader)
```

### Public Static Attributes

```
static int SOLVER_ITERATIONS = -1
```

```
static int MIN_ITERATIONS = -1
```

```
static int MAX_ITERATIONS = -1
```

```
static int MAX_ERROR = -1
```

## Protected Functions

```

void clearAccelerations (const unsigned int fluidModelIndex)
    Clear accelerations and add gravitation.

virtual void initParameters ()

void approximateNormal (Discregrid::DiscreteGrid *map, const Eigen::Vector3d &x,
    Eigen::Vector3d &n, const unsigned int dim)
void computeVolumeAndBoundaryX (const unsigned int fluidModelIndex, const unsigned int i,
    const Vector3r &xi)
void computeVolumeAndBoundaryX ()
void computeDensityAndGradient (const unsigned int fluidModelIndex, const unsigned int i,
    const Vector3r &xi)
void computeDensityAndGradient ()

```

## Protected Attributes

```

unsigned int m_iterations
Real m_maxError
unsigned int m_minIterations
unsigned int m_maxIterations

```

## Class TimeStepDFSPH

- Defined in file \_SPlisHSPlasH\_DFSPh\_TimeStepDFSPH.h

## Inheritance Relationships

### Base Type

- public SPH::TimeStep (*Class TimeStep*)

## Class Documentation

**class** SPH::TimeStepDFSPH : **public** SPH::TimeStep

This class implements the Divergence-free Smoothed Particle Hydrodynamics approach introduced by Bender and Koschier [BK15,BK17,KBST19].

References:

- [BK15] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '15, 147-155. New York, NY, USA, 2015. ACM. URL: <http://doi.acm.org/10.1145/2786784.2786796>
- [BK17] Jan Bender and Dan Koschier. Divergence-free SPH for incompressible and viscous fluids. IEEE Transactions on Visualization and Computer Graphics, 23(3):1193-1206, 2017. URL: <http://dx.doi.org/10.1109/TVCG.2016.2578335>

- [KBST19] Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. Smoothed particle hydrodynamics for physically-based simulation of fluids and solids. In Eurographics 2019 - Tutorials. Eurographics Association, 2019. URL: <https://interactivecomputergraphics.github.io/SPH-Tutorial>

## Public Functions

```
TimeStepDFSPH()
virtual ~TimeStepDFSPH(void)
virtual void step()
virtual void reset()
virtual void resize()
```

## Public Static Attributes

```
static int SOLVER_ITERATIONS_V = -1
static int MAX_ITERATIONS_V = -1
static int MAX_ERROR_V = -1
static int USE_DIVERGENCE_SOLVER = -1
```

## Protected Functions

```
void computeDFSPHFactor(const unsigned int fluidModelIndex)
void pressureSolve()
void pressureSolveIteration(const unsigned int fluidModelIndex, Real &avg_density_err)
void divergenceSolve()
void divergenceSolveIteration(const unsigned int fluidModelIndex, Real &avg_density_err)
void computeDensityAdv(const unsigned int fluidModelIndex, const unsigned int index, const
                      int numParticles, const Real h, const Real density0)
void computeDensityChange(const unsigned int fluidModelIndex, const unsigned int index,
                         const Real h)
void warmstartDivergenceSolve(const unsigned int fluidModelIndex)
void warmstartPressureSolve(const unsigned int fluidModelIndex)
void performNeighborhoodSearch()
    Perform the neighborhood search for all fluid particles.
virtual void emittedParticles(FluidModel *model, const unsigned int startIndex)
virtual void initParameters()
```

## Protected Attributes

```
SimulationDataDFSPH m_simulationData
unsigned int m_counter
const Real m_eps = static_cast<Real>(1.0e-5)
bool m_enableDivergenceSolver
unsigned int m_iterationsV
Real m_maxErrorV
unsigned int m_maxIterationsV
```

## Class TimeStepICSPH

- Defined in file\_SPlisHSPlasH\_ICSPH\_TimeStepICSPH.h

## Inheritance Relationships

### Base Type

- public SPH::TimeStep (*Class TimeStep*)

## Class Documentation

**class** SPH::TimeStepICSPH : public SPH::TimeStep

This class implements the Implicit Compressible SPH approach introduced by Gissler et al. [GHB+20].

References:

- [GHB+20] Christoph Gissler, Andreas Henne, Stefan Band, Andreas Peer and Matthias Teschner. An Implicit Compressible SPH Solver for Snow *Simulation*. ACM Transactions on Graphics, 39(4). URL: <https://doi.org/10.1145/3386569.3392431>

## Public Functions

```
TimeStepICSPH()
virtual ~TimeStepICSPH(void)
virtual void step()
virtual void reset()
virtual void resize()
inline const SimulationDataICSPH &getSimulationData()
```

## Public Static Attributes

```
static int LAMBDA = -1  
static int PRESSURE_CLAMPING = -1
```

## Protected Functions

```
void computeDensityAdv (const unsigned int fluidModelIndex)  
    Compute rho_adv, i ^ (0) (see equation(6) in[GHB + 20]) using the velocities after the non-pressure forces  
    were applied.  
void compute_aii (const unsigned int fluidModelIndex)  
void pressureSolve ()  
void pressureSolveIteration (const unsigned int fluidModelIndex, Real &avg_density_err)  
void integration (const unsigned int fluidModelIndex)  
void computePressureAccels (const unsigned int fluidModelIndex)  
    Determine the pressure accelerations when the pressure is already known.  
void performNeighborhoodSearch ()  
    Perform the neighborhood search for all fluid particles.  
virtual void initParameters ()  
virtual void emittedParticles (FluidModel *model, const unsigned int startIndex)
```

## Protected Attributes

```
SimulationDataICSPH m_simulationData  
Real m_lambda  
bool m_clamping  
const Real m_psi = 1.5  
unsigned int m_counter
```

## Class TimeStepIISPH

- Defined in file\_SPlisHSPlasH\_IISPH\_TimeStepIISPH.h

## Inheritance Relationships

### Base Type

- public SPH::TimeStep (*Class TimeStep*)

## Class Documentation

```
class SPH::TimeStepIISPH : public SPH::TimeStep
```

This class implements the Implicit Incompressible SPH approach introduced by Ihmsen et al. [ICS+14].

References:

- [ICS+14] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible SPH. IEEE Transactions on Visualization and Computer Graphics, 20(3):426-435, March 2014. URL: <http://dx.doi.org/10.1109/TVCG.2013.105>

### Public Functions

```
TimeStepIISPH()  

virtual ~TimeStepIISPH(void)  

virtual void step()  

virtual void reset()  

virtual void resize()  

inline const SimulationDataIISPH &getSimulationData()
```

### Protected Functions

```
void predictAdvection(const unsigned int fluidModelIndex)  

void pressureSolve()  

void pressureSolveIteration(const unsigned int fluidModelIndex, Real &avg_density_err)  

void integration(const unsigned int fluidModelIndex)  

void computePressureAccels(const unsigned int fluidModelIndex)  

    Determine the pressure accelerations when the pressure is already known.  

void performNeighborhoodSearch()  

    Perform the neighborhood search for all fluid particles.
```

```
virtual void emittedParticles(FluidModel *model, const unsigned int startIndex)
```

### Protected Attributes

```
SimulationDataIISPH m_simulationData
```

```
unsigned int m_counter
```

## Class TimeStepPBF

- Defined in file\_SPlisHSPlasH\_PBF\_TimeStepPBF.h

### Inheritance Relationships

#### Base Type

- public SPH::TimeStep (*Class TimeStep*)

### Class Documentation

```
class SPH::TimeStepPBF : public SPH::TimeStep
```

This class implements the position-based fluids approach introduced by Macklin and Mueller [MM13,BMO+14,BMM15].

References:

- [MM13] Miles Macklin and Matthias Müller. Position based fluids. ACM Trans. Graph., 32(4):104:1-104:12, July 2013. URL: <http://doi.acm.org/10.1145/2461912.2461984>
- [BMO+14] Jan Bender, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin. A survey on position-based simulation methods in computer graphics. Computer Graphics Forum, 33(6):228-251, 2014. URL: <http://dx.doi.org/10.1111/cgf.12346>
- [BMM15] Jan Bender, Matthias Müller, and Miles Macklin. Position-based simulation methods in computer graphics. In EUROGRAPHICS 2015 Tutorials. Eurographics Association, 2015. URL: <http://dx.doi.org/10.2312/egt.20151045>

### Public Functions

```
TimeStepPBF()
```

Initialize the simulation data required for this method.

```
virtual ~TimeStepPBF (void)
```

```
virtual void step()
```

Perform a simulation step.

```
virtual void reset()
```

Reset the simulation method.

```
virtual void resize()
```

### Public Static Attributes

```
static int VELOCITY_UPDATE_METHOD = -1
```

```
static int ENUM_PBF_FIRST_ORDER = -1
```

```
static int ENUM_PBF_SECOND_ORDER = -1
```

## Protected Functions

```
void pressureSolve()
    Perform a position-based correction step for the following density constraint: $C(\mathbf{x}) = \left( \frac{\rho_i}{\rho_0} - 1 \right) = 0$ 
void pressureSolveIteration (const unsigned int fluidModelIndex, Real &avg_density_err)
void performNeighborhoodSearch()
    Perform the neighborhood search for all fluid particles.
virtual void emittedParticles (FluidModel *model, const unsigned int startIndex)
virtual void initParameters()
```

## Protected Attributes

```
SimulationDataPBF m_simulationData
unsigned int m_counter
int m_velocityUpdateMethod
```

## Class TimeStepPCISPH

- Defined in file\_SPlisHSPlasH\_PCISPH\_TimeStepPCISPH.h

## Inheritance Relationships

### Base Type

- public SPH::TimeStep (*Class TimeStep*)

## Class Documentation

```
class SPH::TimeStepPCISPH : public SPH::TimeStep
```

This class implements the Predictive-corrective Incompressible SPH approach introduced by Solenthaler and Pajarola [SP09].

### References:

- [SP09] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. ACM Trans. Graph., 28(3):40:1-40:6, July 2009. URL: <http://doi.acm.org/10.1145/1531326.1531346>

## Public Functions

```
TimeStepPCISPH()
virtual ~TimeStepPCISPH (void)
virtual void step ()
virtual void reset ()
virtual void resize ()
```

## Protected Functions

```
void pressureSolve()  
void pressureSolveIteration (const unsigned int fluidModelIndex, Real &avg_density_err)  
void performNeighborhoodSearch()  
    Perform the neighborhood search for all fluid particles.  
virtual void emittedParticles (FluidModel *model, const unsigned int startIndex)
```

## Protected Attributes

```
SimulationDataPCISPH m_simulationData  
unsigned int m_counter
```

## Class TimeStepPF

- Defined in file\_SPlisHSPlasH\_PF\_TimeStepPF.h

## Inheritance Relationships

### Base Type

- public SPH::TimeStep (*Class TimeStep*)

## Class Documentation

```
class SPH::TimeStepPF : public SPH::TimeStep
```

This class implements the Projective Fluids approach introduced by Weiler, Koschier and Bender [WKB16].

References:

- [WKB16] Marcel Weiler, Dan Koschier, and Jan Bender. Projective fluids. In Proceedings of the 9th International Conference on Motion in Games, MIG '16, 79-84. New York, NY, USA, 2016. ACM. URL: <http://doi.acm.org/10.1145/2994258.2994282>

## Public Functions

```
TimeStepPF ()  
virtual ~TimeStepPF (void)  
virtual void step () override  
virtual void reset () override  
virtual void resize () override
```

## Public Static Functions

```
static void matrixVecProd (const Real *vec, Real *result, void *userData)
```

## Public Static Attributes

```
static int STIFFNESS = -1
```

## Protected Types

```
using VectorXr = Eigen::Matrix<Real, -1, 1>
using VectorXrMap = Eigen::Map<VectorXr>
using Solver = Eigen::ConjugateGradient<MatrixReplacement, Eigen::Lower | Eigen::Upper, JacobiPreconditioner3D>
```

## Protected Functions

```
void preparePreconditioner ()
void initialGuessForPositions (const unsigned int fluidModelIndex)
void solvePDConstraints ()
void updatePositionsAndVelocity (const VectorXr &x)
void addAccelerationToVelocity ()
void matrixFreeRHS (const VectorXr &x, VectorXr &result)
    compute the right hand side of the system in a matrix-free fashion and store the result in result
void performNeighborhoodSearch ()
    Perform the neighborhood search for all fluid particles.
virtual void emittedParticles (FluidModel *model, const unsigned int startIndex)
    override
virtual void initParameters () override
```

## Protected Attributes

```
SimulationDataPF m_simulationData
Solver m_solver
Real m_stiffness
unsigned int m_counter
unsigned int m_numActiveParticlesTotal
```

## Protected Static Functions

```
static FORCE_INLINE void diagonalMatrixElement (const unsigned int row, Vector3r &resu
```

## Class TimeStepWCSPH

- Defined in file\_SPlisHSPlasH\_WCSPH\_TimeStepWCSPH.h

## Inheritance Relationships

### Base Type

- public SPH::TimeStep (*Class TimeStep*)

## Class Documentation

```
class SPH::TimeStepWCSPH : public SPH::TimeStep
```

This class implements the Weakly Compressible SPH for Free Surface Flows approach introduced by Becker and Teschner [BT07].

References:

- [BT07] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. In ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '07, 209-217. Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. URL: <http://dl.acm.org/citation.cfm?id=1272690.1272719>

## Public Functions

```
TimeStepWCSPH()
virtual ~TimeStepWCSPH(void)
virtual void step()
virtual void reset()
virtual void resize()
```

## Public Static Attributes

```
static int STIFFNESS = -1
static int EXPONENT = -1
```

## Protected Functions

```
void computePressureAccels (const unsigned int fluidModelIndex)
    Determine the pressure accelerations when the pressure is already known.

void performNeighborhoodSearch ()
    Perform the neighborhood search for all fluid particles.

virtual void emittedParticles (FluidModel *model, const unsigned int startIndex)
virtual void initParameters ()
```

## Protected Attributes

```
Real m_stiffness
Real m_exponent
SimulationDataWCSPH m_simulationData
unsigned int m_counter
```

## Class TriangleMesh

- Defined in file\_SPlisHSPlasH\_TriangleMesh.h

## Class Documentation

```
class SPH::TriangleMesh
```

Data structure for a triangle mesh with normals and vertex normals.

### Public Types

```
typedef std::vector<unsigned int> Faces
typedef std::vector<Vector3r> Normals
typedef std::vector<Vector3r> Vertices
```

### Public Functions

```
TriangleMesh ()
~TriangleMesh ()
void release ()
void initMesh (const unsigned int nPoints, const unsigned int nFaces)
void addFace (const unsigned int *const indices)
    Add a new face.
void addFace (const int *const indices)
    Add a new face.
void addVertex (const Vector3r &vertex)
    Add new vertex.
```

```
inline const Faces &getFaces() const
inline Faces &getFaces()

inline const Normals &getFaceNormals() const
inline Normals &getFaceNormals()

inline const Normals &getVertexNormals() const
inline Normals &getVertexNormals()

inline const Vertices &getVertices() const
inline Vertices &getVertices()

inline const Vertices &getVertices0() const
inline Vertices &getVertices0()

inline unsigned int numVertices() const
inline unsigned int numFaces() const

void updateMeshTransformation(const Vector3r &x, const Matrix3r &R)
void updateNormals()
void updateVertexNormals()
```

### Protected Attributes

*Vertices* **m\_x0**  
*Vertices* **m\_x**  
*Faces* **m\_indices**  
*Normals* **m\_normals**  
*Normals* **m\_vertexNormals**

### Class Viscosity\_Bender2017

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_Bender2017.h

### Inheritance Relationships

#### Base Type

- public SPH::ViscosityBase (*Class ViscosityBase*)

## Class Documentation

```
class SPH::Viscosity_Bender2017 : public SPH::ViscosityBase
```

This class implements the implicit simulation method for viscous fluids introduced by Bender and Koschier [BK17].

References:

- [BK17] Jan Bender and Dan Koschier. Divergence-free SPH for incompressible and viscous fluids. IEEE Transactions on Visualization and Computer Graphics, 23(3):1193-1206, 2017. URL: <http://dx.doi.org/10.1109/TVCG.2016.2578335>

## Public Functions

```
Viscosity_Bender2017 (FluidModel *model)
virtual ~Viscosity_Bender2017 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
void computeTargetStrainRate ()
void computeViscosityFactor ()

inline FORCE_INLINE void viscoGradientMultTransposeRightOpt (const Eigen::Matrix< Real
    Matrix product

inline FORCE_INLINE const Vector6r & getTargetStrainRate (const unsigned int i) const
inline FORCE_INLINE Vector6r & getTargetStrainRate (const unsigned int i)
inline FORCE_INLINE void setTargetStrainRate (const unsigned int i, const Vector6r &val)
inline FORCE_INLINE const Matrix6r & getViscosityFactor (const unsigned int i) const
inline FORCE_INLINE Matrix6r & getViscosityFactor (const unsigned int i)
inline FORCE_INLINE void setViscosityFactor (const unsigned int i, const Matrix6r &val)
inline FORCE_INLINE const Vector6r & getViscosityLambda (const unsigned int i) const
inline FORCE_INLINE Vector6r & getViscosityLambda (const unsigned int i)
inline FORCE_INLINE void setViscosityLambda (const unsigned int i, const Vector6r &val)
```

## Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
```

### Public Static Attributes

```
static int ITERATIONS = -1  
static int MAX_ITERATIONS = -1  
static int MAX_ERROR = -1
```

### Protected Functions

```
virtual void initParameters()
```

### Protected Attributes

```
std::vector<Vector6r> m_targetStrainRate  
std::vector<Matrix6r> m_viscosityFactor  
std::vector<Vector6r> m_viscosityLambda  
unsigned int m_iterations  
unsigned int m_maxIter  
Real m_maxError
```

## Class Viscosity\_Peer2015

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_Peer2015.h

### Inheritance Relationships

#### Base Type

- public SPH::ViscosityBase (*Class ViscosityBase*)

### Class Documentation

```
class SPH::Viscosity_Peer2015 : public SPH::ViscosityBase
```

This class implements the implicit simulation method for viscous fluids introduced by Peer et al. [PICT15].

References:

- [PICT15] A. Peer, M. Ihmsen, J. Cornelis, and M. Teschner. An Implicit Viscosity Formulation for SPH Fluids. ACM Trans. Graph., 34(4):1-10, 2015. URL: <http://doi.acm.org/10.1145/2766925>

## Public Functions

```
Viscosity_Peer2015 (FluidModel *model)
virtual ~Viscosity_Peer2015 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
inline FORCE_INLINE const Matrix3r & getTargetNablaV (const unsigned int i) const
inline FORCE_INLINE Matrix3r & getTargetNablaV (const unsigned int i)
inline FORCE_INLINE void setTargetNablaV (const unsigned int i, const Matrix3r &val)
```

## Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
static void matrixVecProd (const Real *vec, Real *result, void *userData)
static FORCE_INLINE void diagonalMatrixElement (const unsigned int row, Real &result,
```

## Public Static Attributes

```
static int ITERATIONS = -1
static int MAX_ITERATIONS = -1
static int MAX_ERROR = -1
```

## Protected Types

```
typedef Eigen::ConjugateGradient<MatrixReplacement, Eigen::Lower | Eigen::Upper, JacobiPreconditionerID> Solver
```

## Protected Functions

```
virtual void initParameters ()
void computeDensities ()
```

## Protected Attributes

```
std::vector<Real> m_density
std::vector<Matrix3r> m_targetNablaV
Solver m_solver
unsigned int m_iterations
unsigned int m_maxIter
Real m_maxError
```

## Class Viscosity\_Peer2016

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_Peer2016.h

### Inheritance Relationships

#### Base Type

- public SPH::ViscosityBase (*Class ViscosityBase*)

### Class Documentation

```
class SPH::Viscosity_Peer2016 : public SPH::ViscosityBase
```

This class implements the implicit simulation method for viscous fluids introduced by Peer and Teschner [PGBT17].

References:

- [PGBT17] Andreas Peer, Christoph Gissler, Stefan Band, and Matthias Teschner. An implicit SPH formulation for incompressible linearly elastic solids. Computer Graphics Forum, 2017. URL: <http://dx.doi.org/10.1111/cgf.13317>

### Public Functions

```
Viscosity_Peer2016(FluidModel *model)  
virtual ~Viscosity_Peer2016(void)  
virtual void step()  
virtual void reset()  
virtual void performNeighborhoodSearchSort()  
inline FORCE_INLINE const Matrix3r & getTargetNablaV (const unsigned int i) const  
inline FORCE_INLINE Matrix3r & getTargetNablaV (const unsigned int i)  
inline FORCE_INLINE void setTargetNablaV (const unsigned int i, const Matrix3r &val)  
inline FORCE_INLINE const Vector3r & getOmega (const unsigned int i) const  
inline FORCE_INLINE Vector3r & getOmega (const unsigned int i)  
inline FORCE_INLINE void setOmega (const unsigned int i, const Vector3r &val)
```

### Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)  
static void matrixVecProdV(const Real *vec, Real *result, void *userData)  
static FORCE_INLINE void diagonalMatrixElementV (const unsigned int row, Real &result,  
static void matrixVecProdOmega (const Real *vec, Real *result, void *userData)  
static FORCE_INLINE void diagonalMatrixElementOmega (const unsigned int row, Real &res
```

## Public Static Attributes

```
static int ITERATIONS_V = -1
static int ITERATIONS_OMEGA = -1
static int MAX_ITERATIONS_V = -1
static int MAX_ERROR_V = -1
static int MAX_ITERATIONS_OMEGA = -1
static int MAX_ERROR_OMEGA = -1
```

## Protected Types

**typedef** Eigen::ConjugateGradient<*MatrixReplacement*, Eigen::Lower | Eigen::Upper, *JacobiPreconditionerID*> **Solver**

## Protected Functions

```
virtual void initParameters ()
void computeDensities ()
```

## Protected Attributes

```
std::vector<Real> m_density
std::vector<Matrix3r> m_targetNablaV
std::vector<Vector3r> m_omega
Solver m_solverV
Solver m_solverOmega
unsigned int m_iterationsV
unsigned int m_iterationsOmega
unsigned int m_maxIterV
Real m_maxErrorV
unsigned int m_maxIterOmega
Real m_maxErrorOmega
```

## Class Viscosity\_Standard

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_Standard.h

## Inheritance Relationships

### Base Type

- public SPH::ViscosityBase (*Class ViscosityBase*)

## Class Documentation

```
class SPH::Viscosity_Standard : public SPH::ViscosityBase
```

This class implements the standard method for viscosity described e.g. by Ihmsen et al. [IOS+14].

The method evaluates the term  $\nu \nabla^2 \mathbf{v}$  and uses an approximation of the kernel Laplacian to improve the stability. This approximation is given in [IOS+14].

References:

- [IOS+14] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH Fluids in Computer Graphics. In Sylvain Lefebvre and Michela Spagnuolo, editors, Eurographics 2014 - State of the Art Reports. The Eurographics Association, 2014. URL: <http://dx.doi.org/10.2312/egst.20141034>

### Public Functions

```
Viscosity_Standard(FluidModel *model)  
virtual ~Viscosity_Standard(void)  
virtual void step()  
virtual void reset()
```

### Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)
```

### Public Static Attributes

```
static int VISCOSITY_COEFFICIENT_BOUNDARY = -1
```

### Protected Functions

```
virtual void initParameters()
```

## Protected Attributes

*Real* `m_boundaryViscosity`

## Class Viscosity\_Takahashi2015

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_Takahashi2015.h

## Inheritance Relationships

### Base Type

- `public SPH::ViscosityBase (Class ViscosityBase)`

## Class Documentation

```
class SPH::Viscosity_Takahashi2015 : public SPH::ViscosityBase
```

This class implements a variant of the implicit simulation method for viscous fluids introduced by Takahashi et al. [TDF+15]. In the original work of Takahashi et al. the second-ring neighbors are required to create the matrix of the linear system. In contrast we use a meshless conjugate gradient solver which performs the required matrix-vector multiplication in two sequential loops. In this way only the one-ring neighbors are required in each loop which increases the performance significantly.

Thanks to Anreas Peer who helped us with the implementation.

References:

- [TDF+15] T. Takahashi, Y. Dobashi, I. Fujishiro, T. Nishita, and M.C. Lin. Implicit Formulation for SPH-based Viscous Fluids. Computer Graphics Forum, 34(2):493-502, 2015. URL: <http://dx.doi.org/10.1111/cgf.12578>

## Public Functions

```
Viscosity_Takahashi2015 (FluidModel *model)
virtual ~Viscosity_Takahashi2015 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
inline FORCE_INLINE const Matrix3r & getViscousStress (const unsigned int i) const
inline FORCE_INLINE Matrix3r & getViscousStress (const unsigned int i)
inline FORCE_INLINE void setViscousStress (const unsigned int i, const Matrix3r &val)
inline FORCE_INLINE const Vector3r & getAccel (const unsigned int i) const
inline FORCE_INLINE Vector3r & getAccel (const unsigned int i)
inline FORCE_INLINE void setAccel (const unsigned int i, const Vector3r &val)
```

## Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)  
static void matrixVecProd(const Real *vec, Real *result, void *userData)  
static FORCE_INLINE void diagonalMatrixElement (const unsigned int row, Real &result, ...)
```

## Public Static Attributes

```
static int ITERATIONS = -1  
static int MAX_ITERATIONS = -1  
static int MAX_ERROR = -1
```

## Protected Types

```
typedef Eigen::ConjugateGradient<MatrixReplacement, Eigen::Lower | Eigen::Upper, Eigen::IdentityPreconditioner> Solver
```

## Protected Functions

```
virtual void initParameters()
```

## Protected Attributes

```
std::vector<Vector3r> m_accel  
std::vector<Matrix3r> m_viscousStress  
Solver m_solver  
unsigned int m_iterations  
unsigned int m_maxIter  
Real m_maxError
```

## Protected Static Functions

```
static void computeViscosityAcceleration(Viscosity_Takahashi2015 *visco, const Real  
*v)
```

## Class Viscosity\_Weiler2018

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_Weiler2018.h

## Inheritance Relationships

### Base Type

- public SPH::ViscosityBase (*Class ViscosityBase*)

## Class Documentation

**class** SPH::Viscosity\_Weiler2018 : **public** SPH::ViscosityBase

This class implements the implicit Laplace viscosity method introduced by Weiler et al. 2018 [WKBB18].

References:

- [WKBB18] Marcel Weiler, Dan Koschier, Magnus Brand, and Jan Bender. A physically consistent implicit viscosity solver for SPH fluids. Computer Graphics Forum (Eurographics), 2018. URL: <https://doi.org/10.1111/cgf.13349>

### Public Functions

```
Viscosity_Weiler2018 (FluidModel *model)
virtual ~Viscosity_Weiler2018 (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
inline FORCE_INLINE const Vector3r & getVDiff (const unsigned int i) const
inline FORCE_INLINE Vector3r & getVDiff (const unsigned int i)
inline FORCE_INLINE void setVDiff (const unsigned int i, const Vector3r &val)
void computeRHS (VectorXr &b, VectorXr &g)
void applyForces (const VectorXr &x)
```

### Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
static void matrixVecProd (const Real *vec, Real *result, void *userData)
```

### Public Static Attributes

```
static int ITERATIONS = -1
static int MAX_ITERATIONS = -1
static int MAX_ERROR = -1
static int VISCOSITY_COEFFICIENT_BOUNDARY = -1
```

## Protected Types

```
typedef Eigen::ConjugateGradient<MatrixReplacement, Eigen::Lower | Eigen::Upper, BlockJacobiPreconditioner3D> Solver
```

## Protected Functions

```
virtual void initParameters()
```

## Protected Attributes

```
Real m_boundaryViscosity  
unsigned int m_maxIter  
Real m_maxError  
unsigned int m_iterations  
std::vector<Vector3r> m_vDiff  
Real m_tangentialDistanceFactor  
Solver m_solver
```

## Protected Static Functions

```
static FORCE_INLINE void diagonalMatrixElement (const unsigned int row, Matrix3r &resu
```

## Class Viscosity\_XSPH

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_XSPH.h

## Inheritance Relationships

### Base Type

- public SPH::ViscosityBase (*Class ViscosityBase*)

## Class Documentation

```
class SPH::Viscosity_XSPH : public SPH::ViscosityBase
```

This class implements the XSPH method described by Schechter and Bridson [SB12].

References:

- [SB12] Hagit Schechter and Robert Bridson. Ghost sph for animating water. ACM Trans. Graph., 31(4):61:1-61:8, July 2012. URL: <http://doi.acm.org/10.1145/2185520.2185557>

## Public Functions

```
Viscosity_XSPH(FluidModel *model)
virtual ~Viscosity_XSPH(void)
virtual void step()
virtual void reset()
```

## Public Static Functions

```
static inline NonPressureForceBase *creator(FluidModel *model)
```

## Public Static Attributes

```
static int VISCOSITY_COEFFICIENT_BOUNDARY = -1
```

## Protected Functions

```
virtual void initParameters()
```

## Protected Attributes

*Real* **m\_boundaryViscosity**

## Class ViscosityBase

- Defined in file\_SPlisHSPlasH\_Viscosity\_ViscosityBase.h

## Inheritance Relationships

### Base Type

- public SPH::NonPressureForceBase (*Class NonPressureForceBase*)

### Derived Types

- public SPH::Viscosity\_Bender2017 (*Class Viscosity\_Bender2017*)
- public SPH::Viscosity\_Peer2015 (*Class Viscosity\_Peer2015*)
- public SPH::Viscosity\_Peer2016 (*Class Viscosity\_Peer2016*)
- public SPH::Viscosity\_Standard (*Class Viscosity\_Standard*)
- public SPH::Viscosity\_Takahashi2015 (*Class Viscosity\_Takahashi2015*)
- public SPH::Viscosity\_Weiler2018 (*Class Viscosity\_Weiler2018*)
- public SPH::Viscosity\_XSPH (*Class Viscosity\_XSPH*)

## Class Documentation

```
class SPH::ViscosityBase : public SPH::NonPressureForceBase
    Base class for all viscosity methods.
```

Subclassed by *SPH::Viscosity\_Bender2017*, *SPH::Viscosity\_Peer2015*, *SPH::Viscosity\_Peer2016*, *SPH::Viscosity\_Standard*, *SPH::Viscosity\_Takahashi2015*, *SPH::Viscosity\_Weiler2018*, *SPH::Viscosity\_XSPH*

### Public Functions

```
ViscosityBase(FluidModel *model)
virtual ~ViscosityBase(void)
```

### Public Static Attributes

```
static int VISCOSITY_COEFFICIENT = -1
```

### Protected Functions

```
virtual void initParameters()
```

### Protected Attributes

*Real* m\_viscosity

## Class VorticityBase

- Defined in file \_SPlisHSPlasH\_Vorticity\_VorticityBase.h

### Inheritance Relationships

#### Base Type

- public SPH::NonPressureForceBase (*Class NonPressureForceBase*)

#### Derived Types

- public SPH::MicropolarModel\_Bender2017 (*Class MicropolarModel\_Bender2017*)
- public SPH::VorticityConfinement (*Class VorticityConfinement*)

## Class Documentation

```
class SPH::VorticityBase : public SPH::NonPressureForceBase
    Base class for all vorticity methods.

    Subclassed by SPH::MicropolarModel_Bender2017, SPH::VorticityConfinement
```

### Public Functions

```
VorticityBase (FluidModel *model)
virtual ~VorticityBase (void)
```

### Public Static Attributes

```
static int VORTICITY_COEFFICIENT = -1
```

### Protected Functions

```
virtual void initParameters()
```

### Protected Attributes

*Real* **m\_vorticityCoeff**

## Class VorticityConfinement

- Defined in file\_SPlisHSPlasH\_Vorticity\_VorticityConfinement.h

## Inheritance Relationships

### Base Type

- public** SPH::VorticityBase (*Class VorticityBase*)

## Class Documentation

```
class SPH::VorticityConfinement : public SPH::VorticityBase
    This class implements the vorticity confinement method introduced by Macklin and Mueller [MM13].
```

### References:

- [MM13] Miles Macklin and Matthias Müller. Position based fluids. ACM Trans. Graph., 32(4):104:1–104:12, July 2013. URL: <http://doi.acm.org/10.1145/2461912.2461984>

## Public Functions

```
VorticityConfinement (FluidModel *model)
virtual ~VorticityConfinement (void)
virtual void step ()
virtual void reset ()
virtual void performNeighborhoodSearchSort ()
inline FORCE_INLINE const Vector3r & getAngularVelocity (const unsigned int i) const
inline FORCE_INLINE Vector3r & getAngularVelocity (const unsigned int i)
inline FORCE_INLINE void setAngularVelocity (const unsigned int i, const Vector3r &val)
```

## Public Static Functions

```
static inline NonPressureForceBase *creator (FluidModel *model)
```

## Protected Attributes

```
std::vector<Vector3r> m_omega
std::vector<Real> m_normOmega
```

## Class WendlandQuinticC2Kernel

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

## Class Documentation

```
class SPH::WendlandQuinticC2Kernel
quintic Wendland C2 kernel.
```

## Public Static Functions

```
static inline Real getRadius ()
static inline void setRadius (Real val)
static inline Real W (const Real r)
static inline Real W (const Vector3r &r)
static inline Vector3r gradW (const Vector3r &r)
static inline Real W_zero ()
```

### Protected Static Attributes

```
static Real m_radius
static Real m_k
static Real m_l
static Real m_W_zero
```

## Class WendlandQuinticC2Kernel2D

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

### Class Documentation

```
class SPH::WendlandQuinticC2Kernel2D
Wendland Quintic C2 spline kernel (2D).
```

### Public Static Functions

```
static inline Real getRadius ()
static inline void setRadius (Real val)
static inline Real W (const Real r)
static inline Real W (const Vector3r &r)
static inline Vector3r gradW (const Vector3r &r)
static inline Real W_zero ()
```

### Protected Static Attributes

```
static Real m_radius
static Real m_k
static Real m_l
static Real m_W_zero
```

## Class ConsoleSink

- Defined in file\_Utilities\_LOGGER.h

## Inheritance Relationships

### Base Type

- public Utilities::LogSink (*Class LogSink*)

## Class Documentation

```
class Utilities::ConsoleSink : public Utilities::LogSink
```

### Public Functions

```
inline ConsoleSink (const LogLevel minLevel)
inline virtual void write (const LogLevel level, const std::string &str)
```

## Class Counting

- Defined in file\_Utilities\_Counting.h

## Class Documentation

```
class Utilities::Counting
```

### Public Static Functions

```
static inline void reset ()
static inline FORCE_INLINE void increaseCounter (const std::string &name, const Real i)
static inline FORCE_INLINE void printAverageCounts ()
static inline FORCE_INLINE void printCounterSums ()
```

### Public Static Attributes

```
static std::unordered_map<std::string, AverageCount> m_averageCounts
```

## Class FileSink

- Defined in file\_Utilities\_LOGGER.h

## Inheritance Relationships

### Base Type

- public Utilities::LogSink (*Class LogSink*)

## Class Documentation

```
class Utilities::FileSink : public Utilities::LogSink
```

### Public Functions

```
inline FileSink(const LogLevel minLevel, const std::string &fileName)
inline virtual ~FileSink()
inline virtual void write(const LogLevel level, const std::string &str)
```

### Protected Attributes

```
std::ofstream m_file
```

## Class FileSystem

- Defined in file\_Utilities\_FileSystem.h

## Class Documentation

```
class Utilities::FileSystem
```

This class implements different file system functions.

### Public Static Functions

```
static inline std::string getFilePath(const std::string &path)
static inline std::string getFileName(const std::string &path)
static inline std::string getFileNameWithExt(const std::string &path)
static inline std::string getFileExt(const std::string &path)
static inline bool isRelativePath(const std::string &path)
static inline int makeDir(const std::string &path)
static inline int makeDirs(const std::string &path)
    Make all subdirectories.
static inline std::string normalizePath(const std::string &path)
static inline bool fileExists(const std::string &fileName)
static inline std::string getProgramPath()
```

```
static inline bool copyFile (const std::string &source, const std::string &dest)
static inline bool isFile (const std::string &path)
static inline bool isDirectory (const std::string &path)
static inline bool getFilesInDirectory (const std::string &path, std::vector<std::string>
&res)
static inline std::string getFileMD5 (const std::string &filename)
    Compute the MD5 hash of a file.
static inline bool writeMD5File (const std::string &fileName, const std::string &md5File)
    Write the MD5 hash of a file to the md5File.
static inline bool checkMD5 (const std::string &md5Hash, const std::string &md5File)
    Compare an MD5 hash with the hash stored in an MD5 file.
```

## Class IDFactory

- Defined in file\_Utilities\_Timing.h

### Class Documentation

```
class Utilities::IDFactory
    Factory for unique ids.
```

#### Public Static Functions

```
static inline int getId()
```

## Class Logger

- Defined in file\_Utilities\_LOGGER.h

### Class Documentation

```
class Utilities::Logger
```

#### Public Functions

```
inline Logger()
inline ~Logger()
inline void addSink (std::unique_ptr<LogSink> sink)
inline void write (const LogLevel level, const std::string &str)
inline void activate (const bool b)
```

## Protected Attributes

```
std::vector<std::unique_ptr<LogSink>> m_sinks
bool m_active
```

## Class LogSink

- Defined in file\_Utilities\_LOGGER.h

## Inheritance Relationships

### Derived Types

- public Utilities::ConsoleSink (*Class ConsoleSink*)
- public Utilities::FileSink (*Class FileSink*)

## Class Documentation

```
class Utilities::LogSink
Subclassed by Utilities::ConsoleSink, Utilities::FileSink
```

### Public Functions

```
inline LogSink (const LogLevel minLevel)
inline virtual ~LogSink ()
virtual void write (const LogLevel level, const std::string &str) = 0
```

### Protected Attributes

*LogLevel* **m\_minLevel**

## Class LogStream

- Defined in file\_Utilities\_LOGGER.h

## Class Documentation

```
class Utilities::LogStream
```

## Public Functions

```
inline LogStream (Logger *logger, const LogLevel level)
template<typename T>
inline LogStream &operator<< (T const &value)
inline ~LogStream()
```

## Protected Attributes

```
LogLevel m_level
Logger *m_logger
std::ostringstream m_buffer
```

## Class OBJLoader

- Defined in file\_Utilities\_OBJLoader.h

## Class Documentation

```
class Utilities::OBJLoader
Read for OBJ files.
```

### Public Types

```
using Vec3f = std::array<float, 3>
using Vec2f = std::array<float, 2>
```

### Public Static Functions

```
static inline void loadObj (const std::string &filename, std::vector<Vec3f> *x,
                           std::vector<MeshFaceIndices> *faces, std::vector<Vec3f> *normals,
                           std::vector<Vec2f> *texcoords, const Vec3f &scale)
```

This function loads an OBJ file. Only triangulated meshes are supported.

## Class PartioReaderWriter

- Defined in file\_Utilities\_PartioReaderWriter.h

## Class Documentation

```
class Utilities::PartioReaderWriter
```

Class for reading and writing partio files.

### Public Static Functions

```
static bool readParticles (const std::string &fileName, const Vector3r &translation, const Matrix3r &rotation, const Real scale, std::vector<Vector3r> &pos, std::vector<Vector3r> &vel)
static bool readParticles (const std::string &fileName, const Vector3r &translation, const Matrix3r &rotation, const Real scale, std::vector<Vector3r> &positions, std::vector<Vector3r> &velocities, Real &particleRadius)
static bool readParticles (const std::string &fileName, const Vector3r &translation, const Matrix3r &rotation, const Real scale, std::vector<Vector3r> &pos)
static void writeParticles (const std::string &fileName, const unsigned int numParticles, const Vector3r *particlePositions, const Vector3r *particleVelocities, const Real particleRadius)
```

## Class SceneLoader

- Defined in file\_SPlisHSPlasH\_Utilsities\_SceneLoader.h

### Nested Relationships

#### Nested Types

- Struct SceneLoader::AnimationFieldData*
- Struct SceneLoader::BoundaryData*
- Struct SceneLoader::Box*
- Struct SceneLoader::EmitterData*
- Struct SceneLoader::FluidBlock*
- Struct SceneLoader::FluidData*
- Struct SceneLoader::MaterialData*
- Struct SceneLoader::Scene*

## Class Documentation

```
class Utilities::SceneLoader
```

Importer of SPlisHSPlasH scene files.

## Public Functions

```
void readScene (const char *fileName, Scene &scene)  
template<typename T>  
inline bool readValue (const nlohmann::json &j, T &v)  
template<typename T, int size>  
inline bool readVector (const nlohmann::json &j, Eigen::Matrix<T, size, 1, Eigen::DontAlign> &vec)  
template<typename T>  
inline bool readValue (const std::string &section, const std::string &key, T &v)  
inline bool hasValue (const std::string &section, const std::string &key)  
template<typename T, int size>  
inline bool readVector (const std::string &section, const std::string &key, Eigen::Matrix<T, size, 1, Eigen::DontAlign> &vec)  
void readMaterialParameterObject (const std::string &key, GenParam::ParameterObject *paramObj)  
void readParameterObject (const std::string &key, GenParam::ParameterObject *paramObj)  
template<>  
bool readValue (const nlohmann::json &j, bool &v)  
template<>  
bool readValue (const nlohmann::json &j, bool &v)
```

## Protected Functions

```
void readParameterObject (nlohmann::json &config, GenParam::ParameterObject *paramObj)
```

## Protected Attributes

```
nlohmann::json m_jsonData  
struct AnimationFieldData  
    Struct to store an animation field object.
```

## Public Members

```
std::string particleFieldName  
std::string expression[3]  
unsigned int shapeType  
Vector3r x  
Matrix3r rotation  
Vector3r scale  
Real startTime  
Real endTime  
struct BoundaryData  
    Struct to store a boundary object.
```

**Public Members**

```

std::string samplesFile
std::string meshFile
Vector3r translation
Matrix3r rotation
Vector3r scale
Real density
bool dynamic
bool isWall
Eigen::Matrix<float, 4, 1, Eigen::DontAlign> color
void *rigidBody
std::string mapFile
bool mapInvert
Real mapThickness
Eigen::Matrix<unsigned int, 3, 1, Eigen::DontAlign> mapResolution
unsigned int samplingMode
bool isAnimated

struct Box
    Struct for an AABB.

```

**Public Members**

```

Vector3r m_minX
Vector3r m_maxX

struct EmitterData
    Struct to store an emitter object.

```

**Public Members**

```

std::string id
unsigned int width
unsigned int height
Vector3r x
Real velocity
Matrix3r rotation
Real emitStartTime
Real emitEndTime
unsigned int type

```

```
struct FluidBlock
    Struct to store a fluid block.
```

#### Public Members

```
std::string id
std::string visMeshFile
Box box
unsigned char mode
Vector3r initialVelocity
Vector3r initialAngularVelocity
struct FluidData
    Struct to store a fluid object.
```

#### Public Members

```
std::string id
std::string samplesFile
std::string visMeshFile
Vector3r translation
Matrix3r rotation
Vector3r scale
Vector3r initialVelocity
Vector3r initialAngularVelocity
unsigned char mode
bool invert
std::array<unsigned int, 3> resolutionSDF
struct MaterialData
    Struct to store particle coloring information.
```

#### Public Members

```
std::string id
std::string colorField
unsigned int colorMapType
Real minValue
Real maxValue
unsigned int maxEmitterParticles
bool emitterReuseParticles
Vector3r emitterBoxMin
```

```

Vector3r emitterBoxMax

struct Scene
    Struct to store scene information.

Public Members

std::vector<BoundaryData*> boundaryModels
std::vector<FluidData*> fluidModels
std::vector<FluidBlock*> fluidBlocks
std::vector<EmitterData*> emitters
std::vector<AnimationFieldData*> animatedFields
std::vector<MaterialData*> materials

Real particleRadius
bool sim2D
Real timeStepSize
Vector3r camPosition
Vector3r camLookat

```

## Class SDFFunctions

- Defined in file\_SPlisHSPlasH\_Utilsities\_SDFFunctions.h

## Class Documentation

```

class Utilities::SDFFunctions
    Functions for generating and querying an SDF.

```

### Public Static Functions

```

static Discregrid::CubicLagrangeDiscreteGrid *generateSDF (const unsigned int numVertices, const Vector3r *vertices, const unsigned int numFaces, const unsigned int *faces, const AlignedBox3r &bbox, const std::array<unsigned int, 3> &resolution, const bool invert = false)

```

Generate SDF from mesh.

```

static AlignedBox3r computeBoundingBox (const unsigned int numVertices, const Vector3r *vertices)

```

Compute the bounding box of a mesh.

```

static double distance (Discregrid::CubicLagrangeDiscreteGrid *sdf, const Vector3r &x, const Real thickness, Vector3r &normal, Vector3r &nextSurfacePoint)

```

Determine distance of a point x to the surface represented by the SDF and corresponding surface normal and next point on the surface.

```
static double distance (Discregrid::CubicLagrangeDiscreteGrid *sdf, const Vector3r &x, const  
Real thickness)
```

Determine distance of a point x to the surface represented by the SDF.

## Class StringTools

- Defined in file\_Utility\_StringTools.h

### Class Documentation

```
class Utilities::StringTools
```

Tools to handle std::string objects.

#### Public Static Functions

```
static inline void tokenize (const std::string &str, std::vector<std::string> &tokens, const  
std::string &delimiters = " ")  
template<typename T>  
static inline std::string real2String (const T r)  
    converts a double or a float to a string  
static inline std::string to_upper (const std::string &str)
```

## Class SystemInfo

- Defined in file\_Utility\_SystemInfo.h

### Class Documentation

```
class Utilities::SystemInfo
```

#### Public Static Functions

```
static inline std::string getHostName ()
```

## Class Timing

- Defined in file\_Utility\_Timing.h

## Class Documentation

```
class Utilities::Timing
    Class for time measurements.
```

### Public Static Functions

```
static inline void reset()
static inline FORCE_INLINE void startTiming (const std::string &name=std::string(""))
static inline FORCE_INLINE double stopTiming (bool print=true)
static inline FORCE_INLINE double stopTiming (bool print, int &id)
static inline FORCE_INLINE void printAverageTimes ()
static inline FORCE_INLINE void printTimeSums ()
```

### Public Static Attributes

```
static bool m_dontPrintTimes
static unsigned int m_startCounter
static unsigned int m_stopCounter
static std::stack<TimingHelper> m_timingStack
static std::unordered_map<int, AverageTime> m_averageTimes
```

## Class VolumeSampling

- Defined in file\_SPlisHSPlasH\_Utilsies\_VolumeSampling.h

## Class Documentation

```
class Utilities::VolumeSampling
    This class implements a volume sampling of 3D models.
```

### Public Static Functions

```
static void sampleMesh(const unsigned int numVertices, const Vector3r *vertices, const
                     unsigned int numFaces, const unsigned int *faces, const Real radius,
                     const AlignedBox3r *region, const std::array<unsigned int, 3> &resolution, const bool invert, const unsigned int sampleMode,
                     std::vector<Vector3r> &samples)
```

Performs the volume sampling with the respective parameters.

#### Parameters

- numVertices** – number of vertices
- vertices** – vertex data
- numFaces** – number of faces

- **faces** – index list of faces
- **radius** – radius of sampled particles
- **region** – defines a subregion of the mesh to be sampled (nullptr if not used)
- **resolution** – resolution of the used SDF
- **invert** – defines if the mesh should be inverted and the outside is sampled
- **sampleMode** – 0=regular, 1=almost dense, 2=dense
- **samples** – sampled vertices that will be returned

## Class WindingNumbers

- Defined in file\_SPlisHSPlasH\_Utilsities\_WindingNumbers.h

### Class Documentation

```
class Utilities::WindingNumbers
```

#### Public Static Functions

```
static Real computeGeneralizedWindingNumber (const Vector3r &p, const Vector3r  
                                             &a, const Vector3r &b, const Vector3r  
                                             &c)
```

Determine the winding number for a point p and a triangle abc.

```
static Real computeGeneralizedWindingNumber (const Vector3r &p, const  
                                             SPH::TriangleMesh &mesh)
```

Determine the winding number of a point p in a triangle mesh.

## Class Vector3f8

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Class Documentation

```
class Vector3f8
```

#### Public Functions

```
inline Vector3f8 ()
```

```
inline Vector3f8 (const bool)
```

```
inline Vector3f8 (const Scalarf8 &x, const Scalarf8 &y, const Scalarf8 &z)
```

```
inline Vector3f8 (const Scalarf8 &x)
```

```
inline Vector3f8 (const Vector3f &x)
```

```
inline Vector3f8 (const Vector3f &v0, const Vector3f &v1, const Vector3f &v2, const Vec-  
tor3f &v3, const Vector3f &v4, const Vector3f &v5, const Vector3f &v6,  
const Vector3f &v7)
```

```

inline Vector3f8 (Vector3f const *x)
inline void setZero ()
inline Scalarf8 &operator[] (int i)
inline const Scalarf8 &operator[] (int i) const
inline Scalarf8 &x()
inline Scalarf8 &y()
inline Scalarf8 &z()
inline const Scalarf8 &x() const
inline const Scalarf8 &y() const
inline const Scalarf8 &z() const
inline Scalarf8 dot (const Vector3f8 &a) const
inline Scalarf8 operator* (const Vector3f8 &a) const
inline void cross (const Vector3f8 &a, const Vector3f8 &b)
inline const Vector3f8 operator% (const Vector3f8 &a) const
inline Vector3f8 &operator*= (const Scalarf8 &s)
inline const Vector3f8 operator/ (const Scalarf8 &s) const
inline Vector3f8 &operator/= (const Scalarf8 &s)
inline const Vector3f8 operator- () const
inline Scalarf8 squaredNorm () const
inline Scalarf8 norm () const
inline void normalize ()
inline void store (std::vector<Vector3r> &Vf) const
inline void store (Vector3r *Vf) const
inline Vector3r reduce () const

```

## Public Members

*Scalarf8 v[3]*

## Public Static Functions

```

static inline Vector3f8 blend (Scalarf8 const &c, Vector3f8 const &a, Vector3f8 const &b)

```

### 24.1.3 Enums

#### Enum BoundaryHandlingMethods

- Defined in file\_SPlisHSPlasH\_Simulation.h

#### Enum Documentation

```
enum SPH::BoundaryHandlingMethods
Values:
enumerator Akinci2012
enumerator Koschier2017
enumerator Bender2019
enumerator NumSimulationMethods
```

#### Enum FieldType

- Defined in file\_SPlisHSPlasH\_FluidModel.h

#### Enum Documentation

```
enum SPH::FieldType
Values:
enumerator Scalar
enumerator Vector3
enumerator Vector6
enumerator Matrix3
enumerator Matrix6
enumerator UInt
```

#### Enum ParticleState

- Defined in file\_SPlisHSPlasH\_FluidModel.h

#### Enum Documentation

```
enum SPH::ParticleState
Values:
enumerator Active
enumerator AnimatedByEmitter
enumerator Fixed
```

## Enum SimulationMethods

- Defined in file\_SPlisHSPlasH\_Simulation.h

### Enum Documentation

```
enum SPH::SimulationMethods
```

*Values:*

```
enumerator WCSPH  
enumerator PCISPH  
enumerator PBF  
enumerator IISPH  
enumerator DFSPH  
enumerator PF  
enumerator ICSPH  
enumerator NumSimulationMethods
```

## Enum SurfaceSamplingMode

- Defined in file\_SPlisHSPlasH\_Utilsities\_SurfaceSampling.h

### Enum Documentation

```
enum SPH::SurfaceSamplingMode
```

*Values:*

```
enumerator PoissonDisk  
enumerator RegularTriangle  
enumerator Regular2D
```

## Enum LogLevel

- Defined in file\_Utilsities\_LOGGER.h

### Enum Documentation

```
enum Utilities::LogLevel
```

*Values:*

```
enumerator DEBUG  
enumerator INFO  
enumerator WARN  
enumerator ERR
```

## 24.1.4 Functions

### Function abs

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 abs (Scalarf8 const &a)
```

### Function blend

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 blend (Scalarf8 const &c, Scalarf8 const &a, Scalarf8 const &b)
```

### Template Function constant8f

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
template<int i0, int i1, int i2, int i3, int i4, int i5, int i6, int i7>
static inline __m256 constant8f ()
```

### Function convert\_one

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 convert_one (const unsigned int *idx, const Real *x, const unsigned char
count = 8u)
```

### Function convert\_zero(const unsigned int \*, const Real \*, const unsigned char)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 convert_zero (const unsigned int *idx, const Real *x, const unsigned char count = 8u)
```

### Function convert\_zero(const Real, const unsigned char)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 convert_zero (const Real x, const unsigned char count = 8u)
```

### Function convertMat\_zero

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
static inline Matrix3f8 convertMat_zero (const unsigned int *idx, const Matrix3r *v, const unsigned char count = 8u)
```

### Function convertVec\_zero(const unsigned int \*, const Real \*, const unsigned char)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
inline Vector3f8 convertVec_zero (const unsigned int *idx, const Real *v, const unsigned char count = 8u)
```

### Function convertVec\_zero(const unsigned int \*, const Vector3r \*, const unsigned char)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
static inline Vector3f8 convertVec_zero (const unsigned int *idx, const Vector3r *v, const unsigned char count = 8u)
```

## Function dyadicProduct

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
inline void dyadicProduct (const Vector3f8 &a, const Vector3f8 &b, Matrix3f8 &res)
```

## Function getTime

- Defined in file\_SPlisHSPlasH\_AnimationField.cpp

## Function Documentation

```
Real SPH::TimeManager::getTime()
```

## Function max

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 max (Scalarf8 const &a, Scalarf8 const &b)
```

## Function multiplyAndAdd

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 multiplyAndAdd (const Scalarf8 &a, const Scalarf8 &b, const Scalarf8 &c)
```

## Function multiplyAndSubtract

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 multiplyAndSubtract (const Scalarf8 &a, const Scalarf8 &b, const Scalarf8 &c)
```

## Function operator!=

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 operator!=(Scalarf8 const &a, Scalarf8 const &b)
```

## Function operator\*(Scalarf8 const&, Scalarf8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 operator* (Scalarf8 const &a, Scalarf8 const &b)
```

## Function operator\*(Vector3f8 const&, const Scalarf8&)

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

## Function Documentation

```
inline Vector3f8 operator* (Vector3f8 const &a, const Scalarf8 &s)
```

## Function operator\*=

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 &operator*=(Scalarf8 &a, Scalarf8 const &b)
```

## Function operator+(Scalarf8 const&, Scalarf8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsies\_AVX\_math.h

## Function Documentation

```
static inline Scalarf8 operator+ (Scalarf8 const &a, Scalarf8 const &b)
```

### Function operator+(Vector3f8 const&, Vector3f8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
inline Vector3f8 operator+ (Vector3f8 const &a, Vector3f8 const &b)
```

### Function operator+=(Scalarf8&, Scalarf8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
static inline Scalarf8 &operator+= (Scalarf8 &a, Scalarf8 const &b)
```

### Function operator+=(Vector3f8&, Vector3f8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
inline Vector3f8 &operator+= (Vector3f8 &a, Vector3f8 const &b)
```

### Function operator-(Scalarf8&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
inline Scalarf8 operator- (Scalarf8 &a)
```

### Function operator-(Scalarf8 const&, Scalarf8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
static inline Scalarf8 operator- (Scalarf8 const &a, Scalarf8 const &b)
```

### Function operator-(Vector3f8 const&, Vector3f8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
inline Vector3f8 operator- (Vector3f8 const &a, Vector3f8 const &b)
```

### Function operator-=(Scalarmf8&, Scalarmf8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
static inline Scalarmf8 &operator-= (Scalarmf8 &a, Scalarmf8 const &b)
```

### Function operator-=(Vector3f8&, Vector3f8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
inline Vector3f8 &operator-= (Vector3f8 &a, Vector3f8 const &b)
```

### Function operator-=(Matrix3f8&, Matrix3f8 const&)

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
inline Matrix3f8 &operator-= (Matrix3f8 &a, Matrix3f8 const &b)
```

### Function operator/

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

#### Function Documentation

```
static inline Scalarmf8 operator/ (Scalarmf8 const &a, Scalarmf8 const &b)
```

### Function operator/=

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 &operator/=(Scalarf8 &a, Scalarf8 const &b)
```

### Function operator<

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 operator<(Scalarf8 const &a, Scalarf8 const &b)
```

### Function operator<=

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 operator<=(Scalarf8 const &a, Scalarf8 const &b)
```

### Function operator==

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 operator==(Scalarf8 const &a, Scalarf8 const &b)
```

### Function operator>

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

### Function Documentation

```
static inline Scalarf8 operator>(Scalarf8 const &a, Scalarf8 const &b)
```

**Function operator>=**

- Defined in file\_SPlisHSPlasH\_Utilsities\_AVX\_math.h

**Function Documentation**

```
static inline Scalarf8 operator>=(Scalarf8 const &a, Scalarf8 const &b)
```

**24.1.5 Variables****Variable haltonVec323**

- Defined in file\_SPlisHSPlasH\_SurfaceTension\_SurfaceTension\_ZorillaRitter2020\_haltonVec323.h

**Variable Documentation**

```
std::vector<float> haltonVec323
```

**Variable SPH::gaussian\_abscissae\_1**

- Defined in file\_SPlisHSPlasH\_Utilsities\_GaussQuadrature.cpp

**Variable Documentation**

```
double const SPH::gaussian_abscissae_1[101][51]
```

**Variable SPH::gaussian\_n\_1**

- Defined in file\_SPlisHSPlasH\_Utilsities\_GaussQuadrature.cpp

**Variable Documentation**

```
unsigned int const SPH::gaussian_n_1[101]
```

**Variable SPH::gaussian\_weights\_1**

- Defined in file\_SPlisHSPlasH\_Utilsities\_GaussQuadrature.cpp

## Variable Documentation

double **const SPH::gaussian\_weights\_1**[101][51]

## Variable Utilities::logger

- Defined in file\_Utility(Logger.h)

## Variable Documentation

Utilities::*Logger* Utilities::logger

## 24.1.6 Defines

### Define \_USE\_MATH\_DEFINES

- Defined in file\_SPlisHSPlasH\_Drag\_DragForce\_Gissler2017.cpp

### Define Documentation

\_USE\_MATH\_DEFINES

### Define \_USE\_MATH\_DEFINES

- Defined in file\_SPlisHSPlasH\_SPHKernels.h

### Define Documentation

\_USE\_MATH\_DEFINES

### Define \_USE\_MATH\_DEFINES

- Defined in file\_SPlisHSPlasH\_Utils\_PoissonDiskSampling.cpp

### Define Documentation

\_USE\_MATH\_DEFINES

## Define \_USE\_MATH\_DEFINES

- Defined in file\_SPlisHSPlasH\_Utilsities\_WindingNumbers.cpp

## Define Documentation

`_USE_MATH_DEFINES`

## Define compute\_Vj

- Defined in file\_SPlisHSPlasH\_FluidModel.h

## Define Documentation

`compute_Vj (fm_neighbor)`

## Define compute\_Vj\_gradW

- Defined in file\_SPlisHSPlasH\_FluidModel.h

## Define Documentation

`compute_Vj_gradW ()`

## Define compute\_Vj\_gradW\_samephase

- Defined in file\_SPlisHSPlasH\_FluidModel.h

## Define Documentation

`compute_Vj_gradW_samephase ()`

## Define compute\_xj

- Defined in file\_SPlisHSPlasH\_FluidModel.h

## Define Documentation

`compute_xj (fm_neighbor, pid)`

### Define forall\_boundary\_neighbors

- Defined in file\_SPlisHSPlasH\_Simulation.h

### Define Documentation

**forall\_boundary\_neighbors (code)**

Loop over the boundary neighbors of all fluid phases. Simulation \*sim and unsigned int fluidModelIndex must be defined.

### Define forall\_density\_maps

- Defined in file\_SPlisHSPlasH\_Simulation.h

### Define Documentation

**forall\_density\_maps (code)**

Loop over the boundary density maps. Simulation \*sim, unsigned int nBoundaries and unsigned int fluidModelIndex must be defined.

### Define forall\_fluid\_neighbors

- Defined in file\_SPlisHSPlasH\_Simulation.h

### Define Documentation

**forall\_fluid\_neighbors (code)**

Loop over the fluid neighbors of all fluid phases. Simulation \*sim and unsigned int fluidModelIndex must be defined.

### Define forall\_fluid\_neighbors\_in\_same\_phase

- Defined in file\_SPlisHSPlasH\_Simulation.h

### Define Documentation

**forall\_fluid\_neighbors\_in\_same\_phase (code)**

Loop over the fluid neighbors of the same fluid phase. Simulation sim, unsigned int fluidModelIndex and Fluid-Model model must be defined.

## Define forall\_volume\_maps

- Defined in file\_SPlisHSPlasH\_Simulation.h

## Define Documentation

**forall\_volume\_maps** (*code*)

Loop over the boundary volume maps. Simulation \*sim, unsigned int nBoundaries and unsigned int fluidModelIndex must be defined.

## Define FORCE\_INLINE

- Defined in file\_SPlisHSPlasH\_Common.h

## Define Documentation

**FORCE\_INLINE**

## Define INCREASE\_COUNTER

- Defined in file\_Utils\_Counting.h

## Define Documentation

**INCREASE\_COUNTER** (*counterName*, *increaseBy*)

## Define INIT\_COUNTING

- Defined in file\_Utils\_Counting.h

## Define Documentation

**INIT\_COUNTING**

## Define INIT\_LOGGING

- Defined in file\_Utils\_Logger.h

## Define Documentation

`INIT_LOGGING`

### Define INIT\_TIMING

- Defined in file\_Utility\_Timing.h

## Define Documentation

`INIT_TIMING`

### Define LOG\_DEBUG

- Defined in file\_Utility\_Logger.h

## Define Documentation

`LOG_DEBUG`

### Define LOG\_ERR

- Defined in file\_Utility\_Logger.h

## Define Documentation

`LOG_ERR`

### Define LOG\_INFO

- Defined in file\_Utility\_Logger.h

## Define Documentation

`LOG_INFO`

### Define LOG\_WARN

- Defined in file\_Utility\_Logger.h

**Define Documentation****LOG\_WARN****Define PD\_USE\_DIAGONAL\_PRECONDITIONER**

- Defined in file\_SPlisHSPlasH\_PF\_TimeStepPF.h

**Define Documentation****PD\_USE\_DIAGONAL\_PRECONDITIONER****Define REAL\_MAX**

- Defined in file\_SPlisHSPlasH\_Common.h

**Define Documentation****REAL\_MAX****Define REAL\_MIN**

- Defined in file\_SPlisHSPlasH\_Common.h

**Define Documentation****REAL\_MIN****Define RealParameter**

- Defined in file\_SPlisHSPlasH\_Common.h

**Define Documentation****RealParameter****Define RealParameterType**

- Defined in file\_SPlisHSPlasH\_Common.h

## Define Documentation

**RealParameterType**

### Define RealVectorParameter

- Defined in file\_SPlisHSPlasH\_Common.h

## Define Documentation

**RealVectorParameter**

### Define RealVectorParameterType

- Defined in file\_SPlisHSPlasH\_Common.h

## Define Documentation

**RealVectorParameterType**

### Define REPORT\_MEMORY\_LEAKS

- Defined in file\_SPlisHSPlasH\_Common.h

## Define Documentation

**REPORT\_MEMORY\_LEAKS**

### Define S\_ISDIR

- Defined in file\_Utils\_FileSystem.h

## Define Documentation

**S\_ISDIR (*mode*)**

### Define S\_ISREG

- Defined in file\_Utils\_FileSystem.h

## Define Documentation

**S\_ISREG** (*mode*)

## Define START\_TIMING

- Defined in file\_Utility\_Timing.h

## Define Documentation

**START\_TIMING** (*timerName*)

## Define STOP\_TIMING

- Defined in file\_Utility\_Timing.h

## Define Documentation

**STOP\_TIMING**

## Define STOP\_TIMING\_AVG

- Defined in file\_Utility\_Timing.h

## Define Documentation

**STOP\_TIMING\_AVG**

## Define STOP\_TIMING\_AVG\_PRINT

- Defined in file\_Utility\_Timing.h

## Define Documentation

**STOP\_TIMING\_AVG\_PRINT**

## Define STOP\_TIMING\_PRINT

- Defined in file\_Utility\_Timing.h

**Define Documentation**

**STOP\_TIMING\_PRINT**

**Define USE\_BLOCKDIAGONAL\_PRECONDITIONER**

- Defined in file\_SPlisHSPlasH\_Viscosity\_Viscosity\_Weiler2018.h

**Define Documentation**

**USE\_BLOCKDIAGONAL\_PRECONDITIONER**

**Define USE\_WARMSTART**

- Defined in file\_SPlisHSPlasH\_DFSPH\_TimeStepDFSPH.h

**Define Documentation**

**USE\_WARMSTART**

**Define USE\_WARMSTART\_V**

- Defined in file\_SPlisHSPlasH\_DFSPH\_TimeStepDFSPH.h

**Define Documentation**

**USE\_WARMSTART\_V**

**Define Vec3Block**

- Defined in file\_SPlisHSPlasH\_PF\_TimeStepPF.cpp

**Define Documentation**

**Warning:** doxygen define: Cannot find define “Vec3Block” in doxygen xml output for project “SPlisHSPlasH” from directory: ./doxyoutput/xml

## 24.1.7 Typedefs

**Typedef AlignedBox2r**

- Defined in file\_SPlisHSPlasH\_Common.h

## Typedef Documentation

```
using AlignedBox2r = Eigen::AlignedBox<Real, 2>
```

## Typedef AlignedBox3r

- Defined in file\_SPlisHSPlasH\_Common.h

## Typedef Documentation

```
using AlignedBox3r = Eigen::AlignedBox<Real, 3>
```

## Typedef AngleAxisr

- Defined in file\_SPlisHSPlasH\_Common.h

## Typedef Documentation

```
using AngleAxisr = Eigen::AngleAxis<Real>
```

## Typedef AtomicRealVec

- Defined in file\_SPlisHSPlasH\_PF\_TimeStepPF.cpp

## Typedef Documentation

**Warning:** doxygen typedef: Cannot find typedef “AtomicRealVec” in doxygen xml output for project “SPlisHSPlasH” from directory: ./doxygenoutput/xml

## Typedef Matrix2r

- Defined in file\_SPlisHSPlasH\_Common.h

## Typedef Documentation

```
using Matrix2r = Eigen::Matrix<Real, 2, 2, Eigen::DontAlign>
```

### **Typedef Matrix3f**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Matrix3f = Eigen::Matrix<float, 3, 3, Eigen::DontAlign>
```

### **Typedef Matrix3r**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Matrix3r = Eigen::Matrix<Real, 3, 3, Eigen::DontAlign>
```

### **Typedef Matrix4r**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Matrix4r = Eigen::Matrix<Real, 4, 4, Eigen::DontAlign>
```

### **Typedef Matrix5r**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Matrix5r = Eigen::Matrix<Real, 5, 5, Eigen::DontAlign>
```

### **Typedef Matrix6r**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Matrix6r = Eigen::Matrix<Real, 6, 6, Eigen::DontAlign>
```

## Typedef MatrixXr

- Defined in file\_SPlisHSPlasH\_Common.h

### Typedef Documentation

```
using MatrixXr = Eigen::Matrix<Real, -1, -1, 0, -1, -1>
```

## Typedef NeighborhoodSearch

- Defined in file\_SPlisHSPlasH\_NeighborhoodSearch.h

### Typedef Documentation

```
typedef CompactNSearch::NeighborhoodSearch NeighborhoodSearch
```

## Typedef Quaternionr

- Defined in file\_SPlisHSPlasH\_Common.h

### Typedef Documentation

```
using Quaternionr = Eigen::Quaternion<Real, Eigen::DontAlign>
```

## Typedef Real

- Defined in file\_SPlisHSPlasH\_Common.h

### Typedef Documentation

```
typedef float Real
```

## Typedef SystemMatrixType

- Defined in file\_SPlisHSPlasH\_Utils\_MatrixFreeSolver.h

### Typedef Documentation

```
using SystemMatrixType = Eigen::SparseMatrix<Real>
```

### **Typedef Vector2i**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Vector2i = Eigen::Matrix<int, 2, 1, Eigen::DontAlign>
```

### **Typedef Vector2r**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Vector2r = Eigen::Matrix<Real, 2, 1, Eigen::DontAlign>
```

### **Typedef Vector3f**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Vector3f = Eigen::Matrix<float, 3, 1, Eigen::DontAlign>
```

### **Typedef Vector3r**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Vector3r = Eigen::Matrix<Real, 3, 1, Eigen::DontAlign>
```

### **Typedef Vector4f**

- Defined in file\_SPlisHSPlasH\_Common.h

#### **Typedef Documentation**

```
using Vector4f = Eigen::Matrix<float, 4, 1, Eigen::DontAlign>
```

**Typedef Vector4r**

- Defined in file\_SPlisHSPlasH\_Common.h

**Typedef Documentation**

```
using Vector4r = Eigen::Matrix<Real, 4, 1, Eigen::DontAlign>
```

**Typedef Vector5r**

- Defined in file\_SPlisHSPlasH\_Common.h

**Typedef Documentation**

```
using Vector5r = Eigen::Matrix<Real, 5, 1, Eigen::DontAlign>
```

**Typedef Vector6r**

- Defined in file\_SPlisHSPlasH\_Common.h

**Typedef Documentation**

```
using Vector6r = Eigen::Matrix<Real, 6, 1, Eigen::DontAlign>
```

**Typedef VectorXr**

- Defined in file\_SPlisHSPlasH\_Common.h

**Typedef Documentation**

```
using SPH::TimeStepPF::VectorXr = Eigen::Matrix<Real, -1, 1>
```



---

CHAPTER  
**TWENTYFIVE**

---

**REFERENCES**



---

CHAPTER  
**TWENTYSIX**

---

## **INDICES AND TABLES**

- genindex
- search



## BIBLIOGRAPHY

- [AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. *ACM Trans. Graph.*, 32(6):182:1–182:8, November 2013. URL: <http://doi.acm.org/10.1145/2508363.2508395>, doi:10.1145/2508363.2508395.
- [AIA+12] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible sph. *ACM Trans. Graph.*, 31(4):62:1–62:8, July 2012. URL: <http://doi.acm.org/10.1145/2185520.2185558>, doi:10.1145/2185520.2185558.
- [BIT09] Markus Becker, Markus Ihmsen, and Matthias Teschner. Corotated SPH for deformable solids. In *Proceedings of Eurographics Conference on Natural Phenomena*, 27–34. 2009. URL: <http://dx.doi.org/10.2312EG/DL/conf/EG2009/nph/027-034>, doi:10.2312EG/DL/conf/EG2009/nph/027-034.
- [BT07] Markus Becker and Matthias Teschner. Weakly compressible sph for free surface flows. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA ‘07, 209–217. Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. URL: <http://dl.acm.org/citation.cfm?id=1272690.1272719>.
- [BK15] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA ‘15, 147–155. New York, NY, USA, 2015. ACM. URL: <http://doi.acm.org/10.1145/2786784.2786796>, doi:10.1145/2786784.2786796.
- [BK17] Jan Bender and Dan Koschier. Divergence-free sph for incompressible and viscous fluids. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1193–1206, 2017. URL: <http://dx.doi.org/10.1109/TVCG.2016.2578335>, doi:10.1109/TVCG.2016.2578335.
- [BKKW17] Jan Bender, Dan Koschier, Tassilo Kugelstadt, and Marcel Weiler. A micropolar material model for turbulent sph fluids. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA ‘17. ACM, 2017. URL: <http://doi.acm.org/10.1145/3099564.3099578>, doi:10.1145/3099564.3099578.
- [BWK19] Jan Bender, Tassilo Kugelstadt, Marcel Weiler, and Dan Koschier. Volume maps: an implicit boundary representation for sph. In *Proceedings of ACM SIGGRAPH Conference on Motion, Interaction and Games*, MIG ‘19. ACM, 2019. URL: <https://dl.acm.org/doi/10.1145/3359566.3360077>, doi:10.1145/3359566.3360077.
- [BMullerM15] Jan Bender, Matthias Müller, and Miles Macklin. Position-based simulation methods in computer graphics. In *EUROGRAPHICS 2015 Tutorials*. Eurographics Association, 2015. URL: <http://dx.doi.org/10.2312/egt.20151045>, doi:10.2312/egt.20151045.
- [BMullerO+14] Jan Bender, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin. A survey on position-based simulation methods in computer graphics. *Computer Graphics Forum*, 33(6):228–251, 2014. URL: <http://dx.doi.org/10.1111/cgf.12346>, doi:10.1111/cgf.12346.
- [DCB14] Crispin Deul, Patrick Charrier, and Jan Bender. Position-based rigid body dynamics. *Computer Animation and Virtual Worlds*, 2014. URL: <http://dx.doi.org/10.1002/cav.1614>, doi:10.1002/cav.1614.

- [GBP+17] Christoph Gissler, Stefan Band, Andreas Peer, Markus Ihmsen, and Matthias Teschner. Approximate air-fluid interactions for sph. In *Virtual Reality Interactions and Physical Simulations*, 1–10. April 2017. URL: <http://dx.doi.org/10.2312/vrphys.20171081>, doi:10.2312/vrphys.20171081.
- [HWZ+14] Xiaowei He, Huamin Wang, Fengjun Zhang, Hongan Wang, Guoping Wang, and Kun Zhou. Robust simulation of sparsely sampled thin features in sph-based free surface flows. *ACM Trans. Graph.*, 34(1):7:1–7:9, December 2014. URL: <http://doi.acm.org/10.1145/2682630>, doi:10.1145/2682630.
- [ICS+14] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible sph. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):426–435, March 2014. URL: <http://dx.doi.org/10.1109/TVCG.2013.105>, doi:10.1109/TVCG.2013.105.
- [IOS+14] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH Fluids in Computer Graphics. In Sylvain Lefebvre and Michela Spagnuolo, editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014. URL: <http://dx.doi.org/10.2312/egst.20141034>, doi:10.2312/egst.20141034.
- [JZW+15] Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. Blue noise sampling using an sph-based method. *ACM Transactions on Graphics (TOG)*, 34(6):1–11, 2015.
- [KB17] Dan Koschier and Jan Bender. Density maps for improved sph boundary handling. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 1–10. July 2017. URL: <http://dx.doi.org/10.1145/3099564.3099565>, doi:10.1145/3099564.3099565.
- [KBST19] Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. Smoothed particle hydrodynamics for physically-based simulation of fluids and solids. In *Eurographics 2019 - Tutorials*. Eurographics Association, 2019. URL: <https://interactivecomputergraphics.github.io/SPH-Tutorial>, doi:10.2312/egt.20191035.
- [KBFernandezFernandez+21] Tassilo Kugelstadt, Jan Bender, José Antonio Fernández-Fernández, Stefan Rhys Jeske, Fabian Löschner, and Andreas Longva. Fast corotated elastic sph solids with implicit zero-energy mode control. *Proc. ACM Comput. Graph. Interact. Tech.*, 2021.
- [MMuller13] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32(4):104:1–104:12, July 2013. URL: <http://doi.acm.org/10.1145/2461912.2461984>, doi:10.1145/2461912.2461984.
- [MMullerCK14] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified Particle Physics for Real-Time Applications. *ACM Trans. Graph.*, 33(4):1–12, 2014. URL: <http://doi.acm.org/10.1145/2601097.2601152>, doi:10.1145/2601097.2601152.
- [PICT15] A. Peer, M. Ihmsen, J. Cornelis, and M. Teschner. An Implicit Viscosity Formulation for SPH Fluids. *ACM Trans. Graph.*, 34(4):1–10, 2015. URL: <http://doi.acm.org/10.1145/2766925>, doi:10.1145/2766925.
- [PGBT17] Andreas Peer, Christoph Gissler, Stefan Band, and Matthias Teschner. An implicit sph formulation for incompressible linearly elastic solids. *Computer Graphics Forum*, 2017. URL: <http://dx.doi.org/10.1111/cgf.13317>, doi:10.1111/cgf.13317.
- [PT16] Andreas Peer and Matthias Teschner. Prescribed velocity gradients for highly viscous SPH fluids with vorticity diffusion. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–9, 2016. URL: <http://dx.doi.org/10.1109/tvcg.2016.2636144>, doi:10.1109/tvcg.2016.2636144.
- [SB12] Hagit Schechter and Robert Bridson. Ghost sph for animating water. *ACM Trans. Graph.*, 31(4):61:1–61:8, July 2012. URL: <http://doi.acm.org/10.1145/2185520.2185557>, doi:10.1145/2185520.2185557.
- [SP09] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible sph. *ACM Trans. Graph.*, 28(3):40:1–40:6, July 2009. URL: <http://doi.acm.org/10.1145/1531326.1531346>, doi:10.1145/1531326.1531346.

- [TDF+15] T. Takahashi, Y. Dobashi, I. Fujishiro, T. Nishita, and M.C. Lin. Implicit Formulation for SPH-based Viscous Fluids. *Computer Graphics Forum*, 34(2):493–502, 2015. URL: <http://dx.doi.org/10.1111/cgf.12578>, doi:10.1111/cgf.12578.
- [WKB16] Marcel Weiler, Dan Koschier, and Jan Bender. Projective fluids. In *Proceedings of the 9th International Conference on Motion in Games*, MIG ‘16, 79–84. New York, NY, USA, 2016. ACM. URL: <http://doi.acm.org/10.1145/2994258.2994282>, doi:10.1145/2994258.2994282.
- [WKBB18] Marcel Weiler, Dan Koschier, Magnus Brand, and Jan Bender. A physically consistent implicit viscosity solver for sph fluids. *Computer Graphics Forum (Eurographics)*, 2018. URL: <https://doi.org/10.1111/cgf.13349>, doi:10.1111/cgf.13349.



# INDEX

## Symbols

\_USE\_MATH\_DEFINES (*C macro*), 222, 223

### A

abs (*C++ function*), 214

AlignedBox2r (*C++ type*), 231

AlignedBox3r (*C++ type*), 231

AlignmentAllocator (*C++ class*), 94

AlignmentAllocator::~AlignmentAllocator  
(*C++ function*), 94

AlignmentAllocator::adress (*C++ function*),  
94

AlignmentAllocator::AlignmentAllocator  
(*C++ function*), 94

AlignmentAllocator::allocate (*C++ func-  
tion*), 94

AlignmentAllocator::const\_pointer (*C++  
type*), 94

AlignmentAllocator::const\_reference  
(*C++ type*), 94

AlignmentAllocator::construct (*C++ func-  
tion*), 94

AlignmentAllocator::deallocate (*C++ func-  
tion*), 94

AlignmentAllocator::destroy (*C++ function*),  
94

AlignmentAllocator::difference\_type  
(*C++ type*), 94

AlignmentAllocator::max\_size (*C++ func-  
tion*), 94

AlignmentAllocator::operator!= (*C++ func-  
tion*), 94

AlignmentAllocator::operator== (*C++ func-  
tion*), 94

AlignmentAllocator::pointer (*C++ type*), 94

AlignmentAllocator::rebind (*C++ struct*), 81,  
94

AlignmentAllocator::rebind::other (*C++  
type*), 81, 95

AlignmentAllocator::reference (*C++ type*),  
94

AlignmentAllocator::size\_type (*C++ type*),  
94

AlignmentAllocator::value\_type (*C++ type*),  
94

AngleAxisr (*C++ type*), 231

### B

blend (*C++ function*), 214

### C

compute\_Vj (*C macro*), 223

compute\_Vj\_gradW (*C macro*), 223

compute\_Vj\_gradW\_samephase (*C macro*), 223

compute\_xj (*C macro*), 223

constant8f (*C++ function*), 214

convert\_one (*C++ function*), 214

convert\_zero (*C++ function*), 215

convertMat\_zero (*C++ function*), 215

convertVec\_zero (*C++ function*), 215

### D

dyadicProduct (*C++ function*), 216

### E

Eigen::internal::generic\_product\_impl<MatrixReplace-  
Rhs, SparseShape, DenseShape,  
GemvProduct> (*C++ struct*), 81

Eigen::internal::generic\_product\_impl<MatrixReplace-  
Rhs, SparseShape, DenseShape,  
GemvProduct>::Scalar (*C++ type*), 81

Eigen::internal::generic\_product\_impl<MatrixReplace-  
Rhs, SparseShape, DenseShape,  
GemvProduct>::scaleAndAddTo (*C++  
function*), 81

Eigen::internal::traits<SPH::MatrixReplacement>  
(*C++ struct*), 82

### F

forall\_boundary\_neighbors (*C macro*), 224

forall\_density\_maps (*C macro*), 224

forall\_fluid\_neighbors (*C macro*), 224

forall\_fluid\_neighbors\_in\_same\_phase (*C macro*), 224  
forall\_volume\_maps (*C macro*), 225  
FORCE\_INLINE (*C macro*), 225

**H**  
haltonVec323 (*C++ member*), 221

**I**  
INCREASE\_COUNTER (*C macro*), 225  
INIT\_COUNTING (*C macro*), 225  
INIT\_LOGGING (*C macro*), 226  
INIT\_TIMING (*C macro*), 226

**L**  
LOG\_DEBUG (*C macro*), 226  
LOG\_ERR (*C macro*), 226  
LOG\_INFO (*C macro*), 226  
LOG\_WARN (*C macro*), 227

**M**  
Matrix2r (*C++ type*), 231  
Matrix3f (*C++ type*), 232  
Matrix3f8 (*C++ class*), 95  
Matrix3f8::determinant (*C++ function*), 95  
Matrix3f8::m (*C++ member*), 95  
Matrix3f8::Matrix3f8 (*C++ function*), 95  
Matrix3f8::operator () (*C++ function*), 95  
Matrix3f8::operator\* (*C++ function*), 95  
Matrix3f8::operator+= (*C++ function*), 95  
Matrix3f8::reduce (*C++ function*), 95  
Matrix3f8::setCol (*C++ function*), 95  
Matrix3f8::setZero (*C++ function*), 95  
Matrix3f8::store (*C++ function*), 95  
Matrix3f8::transpose (*C++ function*), 95  
Matrix3r (*C++ type*), 232  
Matrix4r (*C++ type*), 232  
Matrix5r (*C++ type*), 232  
Matrix6r (*C++ type*), 232  
MatrixXr (*C++ type*), 233  
max (*C++ function*), 216  
multiplyAndAdd (*C++ function*), 216  
multiplyAndSubtract (*C++ function*), 216

**N**  
NeighborhoodSearch (*C++ type*), 233

**O**  
operator!= (*C++ function*), 217  
operator\* (*C++ function*), 217  
operator\*= (*C++ function*), 217  
operator+ (*C++ function*), 217, 218  
operator+= (*C++ function*), 218

operator/ (*C++ function*), 219  
operator/= (*C++ function*), 220  
operator== (*C++ function*), 220  
operator- (*C++ function*), 218, 219  
operator-= (*C++ function*), 219  
operator> (*C++ function*), 220  
operator>= (*C++ function*), 221  
operator< (*C++ function*), 220  
operator<= (*C++ function*), 220

**P**  
PD\_USE\_DIAGONAL\_PRECONDITIONER (*C macro*), 227

**Q**  
Quaternion8f (*C++ class*), 96  
Quaternion8f::operator\* (*C++ function*), 96  
Quaternion8f::operator[] (*C++ function*), 96  
Quaternion8f::q (*C++ member*), 96  
Quaternion8f::Quaternion8f (*C++ function*), 96  
Quaternion8f::set (*C++ function*), 96  
Quaternion8f::store (*C++ function*), 96  
Quaternion8f::toRotationMatrix (*C++ function*), 96  
Quaternion8f::w (*C++ function*), 96  
Quaternion8f::x (*C++ function*), 96  
Quaternion8f::y (*C++ function*), 96  
Quaternion8f::z (*C++ function*), 96  
Quaternionr (*C++ type*), 233

**R**  
Real (*C++ type*), 233  
REAL\_MAX (*C macro*), 227  
REAL\_MIN (*C macro*), 227  
RealParameter (*C macro*), 227  
RealParameterType (*C macro*), 228  
RealVectorParameter (*C macro*), 228  
RealVectorParameterType (*C macro*), 228  
REPORT\_MEMORY\_LEAKS (*C macro*), 228

**S**  
S\_ISDIR (*C macro*), 228  
S\_ISREG (*C macro*), 229  
Scalarf8 (*C++ class*), 97  
Scalarf8::load (*C++ function*), 97  
Scalarf8::operator= (*C++ function*), 97  
Scalarf8::reduce (*C++ function*), 97  
Scalarf8::rsqrt (*C++ function*), 97  
Scalarf8::Scalf8 (*C++ function*), 97  
Scalarf8::setZero (*C++ function*), 97  
Scalarf8::sqrt (*C++ function*), 97  
Scalarf8::store (*C++ function*), 97  
Scalarf8::v (*C++ member*), 97

SPH::AdhesionKernel (*C++ class*), 97  
 SPH::AdhesionKernel::getRadius (*C++ function*), 98  
 SPH::AdhesionKernel::m\_k (*C++ member*), 98  
 SPH::AdhesionKernel::m\_radius (*C++ member*), 98  
 SPH::AdhesionKernel::m\_W\_zero (*C++ member*), 98  
 SPH::AdhesionKernel::setRadius (*C++ function*), 98  
 SPH::AdhesionKernel::W (*C++ function*), 98  
 SPH::AdhesionKernel::W\_zero (*C++ function*), 98  
 SPH::AnimationField (*C++ class*), 98  
 SPH::AnimationField::~AnimationField (*C++ function*), 98  
 SPH::AnimationField::AnimationField (*C++ function*), 98  
 SPH::AnimationField::m\_endTime (*C++ member*), 99  
 SPH::AnimationField::m\_expression (*C++ member*), 99  
 SPH::AnimationField::m\_particleFieldName (*C++ member*), 99  
 SPH::AnimationField::m\_rotation (*C++ member*), 99  
 SPH::AnimationField::m\_scale (*C++ member*), 99  
 SPH::AnimationField::m\_startTime (*C++ member*), 99  
 SPH::AnimationField::m\_type (*C++ member*), 99  
 SPH::AnimationField::m\_x (*C++ member*), 99  
 SPH::AnimationField::reset (*C++ function*), 98  
 SPH::AnimationField::setEndTime (*C++ function*), 98  
 SPH::AnimationField::setStartTime (*C++ function*), 98  
 SPH::AnimationField::step (*C++ function*), 98  
 SPH::AnimationFieldSystem (*C++ class*), 99  
 SPH::AnimationFieldSystem::~AnimationFieldSystem (*C++ function*), 99  
 SPH::AnimationFieldSystem::addAnimationField (BlockJacobiPreconditioner3D::*\_solve\_impl* (*C++ function*), 102)  
 SPH::AnimationFieldSystem::AnimationFieldSystem (BlockJacobiPreconditioner3D::*analyzePattern* (*C++ function*), 102)  
 SPH::AnimationFieldSystem::getAnimationFields (BlockJacobiPreconditioner3D::*BlockJacobiPrecon* (*C++ function*), 102)  
 SPH::AnimationFieldSystem::m\_fields (*C++ member*), 99  
 SPH::AnimationFieldSystem::numAnimationFields (BlockJacobiPreconditioner3D::*compute* (*C++ function*), 102)  
 SPH::AnimationFieldSystem::reset (*C++ function*), 102  
 SPH::BlockJacobiPreconditioner3D::DiagonalMatrixElement (BlockJacobiPreconditioner3D::*DiagonalMatrixElement* (*C++ function*), 102)

```

(C++ type), 101
SPH::BlockJacobiPreconditioner3D::factorize (C++ member), 103
(C++ function), 102
SPH::BlockJacobiPreconditioner3D::info (C++ function), 102
SPH::BlockJacobiPreconditioner3D::init (C++ function), 102
SPH::BlockJacobiPreconditioner3D::m_diagonalElement (C++ member), 103
SPH::BlockJacobiPreconditioner3D::m_dim (C++ member), 102
SPH::BlockJacobiPreconditioner3D::m_invDiag (C++ member), 102
SPH::BlockJacobiPreconditioner3D::m_userData (C++ member), 102
SPH::BlockJacobiPreconditioner3D::rows (C++ function), 102
SPH::BlockJacobiPreconditioner3D::solve (C++ function), 102
SPH::BlockJacobiPreconditioner3D::StorageIndex (C++ function), 104
(C++ type), 101
SPH::BlockJacobiPreconditioner3D::[anonymous] (C++ function), 104
(C++ enum), 101
SPH::BlockJacobiPreconditioner3D::[anonymous]::(C++ function), 104
(C++ enumerator), 101
SPH::BoundaryHandlingMethods (C++ enum), 212
SPH::BoundaryHandlingMethods::Akinci2012 (C++ enumerator), 212
SPH::BoundaryHandlingMethods::Bender2019 (C++ enumerator), 212
SPH::BoundaryHandlingMethods::Koschier2017 (C++ enumerator), 212
SPH::BoundaryHandlingMethods::NumSimulationMeth (C++ member), 105
(C++ enumerator), 212
SPH::BoundaryModel (C++ class), 103
SPH::BoundaryModel::~BoundaryModel (C++ function), 103
SPH::BoundaryModel::BoundaryModel (C++ function), 103
SPH::BoundaryModel::clearForceAndTorque (C++ function), 103
SPH::BoundaryModel::getForceAndTorque (C++ function), 103
SPH::BoundaryModel::getRigidBodyObject (C++ function), 103
SPH::BoundaryModel::loadState (C++ function), 103
SPH::BoundaryModel::m_forcePerThread (C++ member), 103
SPH::BoundaryModel::m_rigidBody (C++ member), 103
SPH::BoundaryModel::m_torquePerThread (C++ member), 103
SPH::BoundaryModel::performNeighborhoodSearchSort (C++ function), 103
SPH::BoundaryModel::reset (C++ function), 103
SPH::BoundaryModel::saveState (C++ function), 103
SPH::BoundaryModel_Akinci2012 (C++ class), 104
SPH::BoundaryModel_Akinci2012::~BoundaryModel_Akinci2012 (C++ function), 104
SPH::BoundaryModel_Akinci2012::BoundaryModel_Akinci2012 (C++ function), 104
SPH::BoundaryModel_Akinci2012::computeBoundaryVolume (C++ function), 104
SPH::BoundaryModel_Akinci2012::getPointSetIndex (C++ function), 104
SPH::BoundaryModel_Akinci2012::initModel (C++ function), 104
SPH::BoundaryModel_Akinci2012::isSorted (C++ function), 104
SPH::BoundaryModel_Akinci2012::loadState (C++ function), 104
SPH::BoundaryModel_Akinci2012::m_pointSetIndex (C++ function), 104
SPH::BoundaryModel_Akinci2012::m_sorted (C++ member), 105
SPH::BoundaryModel_Akinci2012::m_v (C++ member), 105
SPH::BoundaryModel_Akinci2012::m_x (C++ member), 105
SPH::BoundaryModel_Akinci2012::m_x0 (C++ member), 105
SPH::BoundaryModel_Akinci2012::numberOfParticles (C++ function), 104
SPH::BoundaryModel_Akinci2012::performNeighborhoodSearch (C++ function), 104
SPH::BoundaryModel_Akinci2012::reset (C++ function), 104
SPH::BoundaryModel_Akinci2012::resize (C++ function), 104
SPH::BoundaryModel_Akinci2012::saveState (C++ function), 104
SPH::BoundaryModel_Bender2019 (C++ class), 105
SPH::BoundaryModel_Bender2019::~BoundaryModel_Bender2019 (C++ function), 105
SPH::BoundaryModel_Bender2019::BoundaryModel_Bender2019 (C++ function), 105
SPH::BoundaryModel_Bender2019::getMap (C++ function), 105

```

```

SPH::BoundaryModel_Bender2019::getMaxDis$SPH::BoundaryModel_Koschier2017::setMaxDist
    (C++ function), 105                                     (C++ function), 107
SPH::BoundaryModel_Bender2019::getMaxVel$SPH::BoundaryModel_Koschier2017::setMaxVel
    (C++ function), 105                                     (C++ function), 107
SPH::BoundaryModel_Bender2019::initModel$SPH::CholeskyAVXSolver (C++ class), 107
    (C++ function), 105                                     SPH::CholeskyAVXSolver::~CholeskyAVXSolver
SPH::BoundaryModel_Bender2019::m_boundaryVolume (C++ function), 107
    (C++ member), 106                                     SPH::CholeskyAVXSolver::CholeskyAVXSolver
SPH::BoundaryModel_Bender2019::m_boundaryXj   (C++ function), 107
    (C++ member), 106                                     SPH::CholeskyAVXSolver::L (C++ member),
SPH::BoundaryModel_Bender2019::m_map          108
    (C++ member), 106                                     SPH::CholeskyAVXSolver::load (C++ func-
SPH::BoundaryModel_Bender2019::m_maxDist      tion), 107
    (C++ member), 106                                     SPH::CholeskyAVXSolver::LT (C++ member),
SPH::BoundaryModel_Bender2019::m_maxVel       108
    (C++ member), 106                                     SPH::CholeskyAVXSolver::n_rhs_lines
SPH::BoundaryModel_Bender2019::reset          (C++ function), 105
    (C++ member), 108                                     (C++ member), 108
SPH::BoundaryModel_Bender2019::setMap         (C++ function), 105
    (C++ function), 105                                     SPH::CholeskyAVXSolver::perm (C++ mem-
SPH::BoundaryModel_Bender2019::setMaxDist     ber), 108
    (C++ function), 105                                     SPH::CholeskyAVXSolver::perm_inv (C++ mem-
SPH::BoundaryModel_Bender2019::setMaxVel      ber), 108
    (C++ function), 105                                     SPH::CholeskyAVXSolver::save (C++ func-
SPH::BoundaryModel_Koschier2017   (C++ class), 106
    (C++ function), 107                                     SPH::CholeskyAVXSolver::solve (C++ func-
SPH::BoundaryModel_Koschier2017::~BoundaryModel 107
    (C++ function), 106                                     SPH::CholeskySparseMatrixfAVX (C++ class),
SPH::BoundaryModel_Koschier2017::BoundaryModel_108
    (C++ function), 106                                     SPH::CholeskySparseMatrixfAVX::CholeskySparseMatri-
SPH::BoundaryModel_Koschier2017::getMap        (C++ function), 106
    (C++ function), 106                                     SPH::CholeskySparseMatrixfAVX::cols
SPH::BoundaryModel_Koschier2017::getMaxDist   (C++ function), 106
    (C++ function), 108                                     SPH::CholeskySparseMatrixfAVX::diagonal_inv
SPH::BoundaryModel_Koschier2017::getMaxVel    (C++ function), 107
    (C++ function), 108                                     SPH::CholeskySparseMatrixfAVX::load
SPH::BoundaryModel_Koschier2017::initModel    (C++ function), 106
    (C++ function), 106                                     SPH::CholeskySparseMatrixfAVX::n_rhs_lines
SPH::BoundaryModel_Koschier2017::m_boundaryDens (C++ member), 108
    (C++ member), 107                                     SPH::CholeskySparseMatrixfAVX::rows_offset
SPH::BoundaryModel_Koschier2017::m_boundaryDens (C++ member), 108
    (C++ member), 107                                     SPH::CholeskySparseMatrixfAVX::save
SPH::BoundaryModel_Koschier2017::m_boundaryXj  (C++ function), 108
    (C++ member), 107                                     SPH::CholeskySparseMatrixfAVX::vals
SPH::BoundaryModel_Koschier2017::m_map         (C++ function), 107
    (C++ member), 108                                     SPH::CohesionKernel (C++ class), 109
SPH::BoundaryModel_Koschier2017::m_maxDis$SPH::CohesionKernel::getRadius (C++ func-
    (C++ member), 107                                     tion), 109
SPH::BoundaryModel_Koschier2017::m_maxVe$SPH::CohesionKernel::m_c (C++ member), 109
    (C++ member), 107                                     SPH::CohesionKernel::m_k (C++ member), 109
SPH::BoundaryModel_Koschier2017::reset         SPH::CohesionKernel::m_radius (C++ mem-
    (C++ function), 106                                     ber), 109
SPH::BoundaryModel_Koschier2017::setMap        SPH::CohesionKernel::m_W_zero (C++ mem-
    (C++ function), 106                                     ber), 109

```

SPH::CohesionKernel::setRadius (*C++ function*), 109  
SPH::CohesionKernel::W (*C++ function*), 109  
SPH::CohesionKernel::W\_zero (*C++ function*), 109  
SPH::CubicKernel (*C++ class*), 109  
SPH::CubicKernel2D (*C++ class*), 110  
SPH::CubicKernel2D::getRadius (*C++ function*), 110  
SPH::CubicKernel2D::gradW (*C++ function*), 110  
SPH::CubicKernel2D::m\_k (*C++ member*), 111  
SPH::CubicKernel2D::m\_l (*C++ member*), 111  
SPH::CubicKernel2D::m\_radius (*C++ member*), 111  
SPH::CubicKernel2D::m\_W\_zero (*C++ member*), 111  
SPH::CubicKernel2D::setRadius (*C++ function*), 110  
SPH::CubicKernel2D::W (*C++ function*), 110  
SPH::CubicKernel2D::W\_zero (*C++ function*), 110  
SPH::CubicKernel::getRadius (*C++ function*), 110  
SPH::CubicKernel::gradW (*C++ function*), 110  
SPH::CubicKernel::m\_k (*C++ member*), 110  
SPH::CubicKernel::m\_l (*C++ member*), 110  
SPH::CubicKernel::m\_radius (*C++ member*), 110  
SPH::CubicKernel::m\_W\_zero (*C++ member*), 110  
SPH::CubicKernel::setRadius (*C++ function*), 110  
SPH::CubicKernel::W (*C++ function*), 110  
SPH::CubicKernel::W\_zero (*C++ function*), 110  
SPH::DebugTools (*C++ class*), 111  
SPH::DebugTools::~DebugTools (*C++ function*), 111  
SPH::DebugTools::cleanup (*C++ function*), 111  
SPH::DebugTools::DebugTools (*C++ function*), 111  
SPH::DebugTools::DETERMINE\_NUM\_NEIGHBORSSPH::DragForce\_Gissler2017::creator (*C++ member*), 111  
SPH::DebugTools::DETERMINE\_THREAD\_IDS (*C++ member*), 111  
SPH::DebugTools::DETERMINE\_VELOCITY\_CHANGESPH::DragForce\_Gissler2017::mu\_a (*C++ member*), 111  
SPH::DebugTools::determineNumNeighbors (*C++ function*), 112  
SPH::DebugTools::determineThreadIds (*C++ function*), 112  
SPH::DebugTools::determineVelocityChangeSPH::DragForce\_Gissler2017::rho\_a (*C++ member*), 112  
SPH::DebugTools::emittedParticles (*C++ function*), 111  
SPH::DebugTools::init (*C++ function*), 111  
SPH::DebugTools::initParameters (*C++ function*), 112  
SPH::DebugTools::m\_determineNumNeighbors (*C++ member*), 112  
SPH::DebugTools::m\_determineThreadIds (*C++ member*), 112  
SPH::DebugTools::m\_determineVelocityChanges (*C++ member*), 112  
SPH::DebugTools::m\_numNeighbors (*C++ member*), 112  
SPH::DebugTools::m\_threadIds (*C++ member*), 112  
SPH::DebugTools::m\_velocityChanges (*C++ member*), 112  
SPH::DebugTools::m\_vOld (*C++ member*), 112  
SPH::DebugTools::performNeighborhoodSearchSort (*C++ function*), 111  
SPH::DebugTools::reset (*C++ function*), 111  
SPH::DebugTools::step (*C++ function*), 111  
SPH::DragBase (*C++ class*), 112  
SPH::DragBase::~DragBase (*C++ function*), 113  
SPH::DragBase::DRAG\_COEFFICIENT (*C++ member*), 113  
SPH::DragBase::DragBase (*C++ function*), 113  
SPH::DragBase::initParameters (*C++ function*), 113  
SPH::DragBase::m\_dragCoefficient (*C++ member*), 113  
SPH::DragForce\_Gissler2017 (*C++ class*), 113  
SPH::DragForce\_Gissler2017::~DragForce\_Gissler2017 (*C++ function*), 113  
SPH::DragForce\_Gissler2017::c\_b (*C++ member*), 114  
SPH::DragForce\_Gissler2017::c\_d (*C++ member*), 114  
SPH::DragForce\_Gissler2017::c\_F (*C++ member*), 114  
SPH::DragForce\_Gissler2017::c\_k (*C++ member*), 114  
SPH::DragForce\_Gissler2017::dragForce\_Gissler2017::creator (*C++ function*), 114  
SPH::DragForce\_Gissler2017::dragForce\_Gissler2017 (*C++ function*), 113  
SPH::DragForce\_Gissler2017::mu\_a (*C++ member*), 114  
SPH::DragForce\_Gissler2017::mu\_l (*C++ member*), 114  
SPH::DragForce\_Gissler2017::reset (*C++ function*), 113  
SPH::DragForce\_Gissler2017::rho\_a (*C++ member*), 114  
SPH::DragForce\_Gissler2017::sigma (*C++ member*), 114

```

    member), 114
SPH::DragForce_Gissler2017::step (C++ function), 113
SPH::DragForce_Macklin2014 (C++ class), 114
SPH::DragForce_Macklin2014::~DragForce_Macklin (C++ function), 115
SPH::DragForce_Macklin2014::creator (C++ function), 115
SPH::DragForce_Macklin2014::reset (C++ function), 114
SPH::DragForce_Macklin2014::step (C++ function), 114
SPH::Elasticity_Becker2009 (C++ class), 115
SPH::Elasticity_Becker2009::~Elasticity_Becker2009 (C++ function), 115
SPH::Elasticity_Becker2009::ALPHA (C++ member), 116
SPH::Elasticity_Becker2009::computeForce (C++ function), 116
SPH::Elasticity_Becker2009::computeForce (C++ function), 116
SPH::Elasticity_Becker2009::computeRotations (C++ function), 116
SPH::Elasticity_Becker2009::computeStress (C++ function), 116
SPH::Elasticity_Becker2009::creator (C++ function), 115
SPH::Elasticity_Becker2009::initParameters (C++ function), 116
SPH::Elasticity_Becker2009::initValues (C++ function), 116
SPH::Elasticity_Becker2009::loadState (C++ function), 115
SPH::Elasticity_Becker2009::m_alpha (C++ member), 116
SPH::Elasticity_Becker2009::m_current_toSPHit (C++ member), 116
SPH::Elasticity_Becker2009::m_F (C++ member), 116
SPH::Elasticity_Becker2009::m_initial_toSPHit (C++ member), 116
SPH::Elasticity_Becker2009::m_initialNeigh (C++ member), 116
SPH::Elasticity_Becker2009::m_restVolume (C++ member), 116
SPH::Elasticity_Becker2009::m_rotations (C++ member), 116
SPH::Elasticity_Becker2009::m_stress (C++ member), 116
SPH::Elasticity_Becker2009::performNeigh (C++ function), 115
SPH::Elasticity_Becker2009::reset (C++ function), 115
SPH::Elasticity_Becker2009::saveState (C++ function), 115
SPH::Elasticity_Becker2009::step (C++ function), 115
SPH::Elasticity_Kugelstadt2021 (C++ class), 117
SPH::Elasticity_Kugelstadt2021::~Elasticity_Kugelstadt2021 (C++ function), 117
SPH::Elasticity_Kugelstadt2021::APD_Newton_AVX (C++ function), 118
SPH::Elasticity_Kugelstadt2021::computeF (C++ function), 120
SPH::Elasticity_Kugelstadt2021::computeMatrixL (C++ function), 118
SPH::Elasticity_Kugelstadt2021::computeMD5 (C++ function), 118
SPH::Elasticity_Kugelstadt2021::computeRHS (C++ function), 118
SPH::Elasticity_Kugelstadt2021::computeRotations (C++ function), 117
SPH::Elasticity_Kugelstadt2021::creator (C++ function), 117
SPH::Elasticity_Kugelstadt2021::deferredInit (C++ function), 119
SPH::Elasticity_Kugelstadt2021::ElasticObject (C++ struct), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::~ElasticObject (C++ function), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::ElasticObject (C++ function), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_f (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_fa (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_fn (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_n (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_pa (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_p (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_q (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_R (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_rh (C++ member), 82, 120
SPH::Elasticity_Kugelstadt2021::ElasticObject::m_s (C++ member), 82, 120

```

(*C++ member*), 82, 120  
 SPH::Elasticity\_Kugelstadt2021::Elasticity\_Kugelstadt2021::m\_maxErrorV  
     (*C++ member*), 82, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_maxIterV  
     (*C++ struct*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_maxNeighbors  
     (*C++ function*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_mu  
     (*C++ function*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_objects  
     (*C++ member*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_precomp\_RL\_gradW  
     (*C++ member*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_precomp\_RLj\_gradW  
     (*C++ member*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_precomputed\_index  
     (*C++ member*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_restVolumes  
     (*C++ member*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_RL  
     (*C++ member*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::FactorizationElasticity\_Kugelstadt2021::m\_rotations  
     (*C++ member*), 83, 120  
 SPH::Elasticity\_Kugelstadt2021::findObject::Elasticity\_Kugelstadt2021::m\_solver  
     (*C++ function*), 118  
 SPH::Elasticity\_Kugelstadt2021::initFact::Elasticity\_Kugelstadt2021::m\_stiffnessDamping  
     (*C++ function*), 118  
 SPH::Elasticity\_Kugelstadt2021::initParam::Elasticity\_Kugelstadt2021::m\_stress  
     (*C++ function*), 119  
 SPH::Elasticity\_Kugelstadt2021::initSystem::Elasticity\_Kugelstadt2021::m\_totalNeighbors  
     (*C++ function*), 118  
 SPH::Elasticity\_Kugelstadt2021::initValue::Elasticity\_Kugelstadt2021::m\_vDiff  
     (*C++ function*), 118  
 SPH::Elasticity\_Kugelstadt2021::ITERATION::Elasticity\_Kugelstadt2021::MASS\_DAMPING\_COEFF  
     (*C++ member*), 118  
 SPH::Elasticity\_Kugelstadt2021::loadStat::Elasticity\_Kugelstadt2021::matrixVecProd  
     (*C++ function*), 117  
 SPH::Elasticity\_Kugelstadt2021::m\_alpha::SPH::Elasticity\_Kugelstadt2021::MAX\_ERROR\_V  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_currentState::SPH::Elasticity\_Kugelstadt2021::MAX\_ITERATIONS\_V  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_F::SPH::Elasticity\_Kugelstadt2021::MAX\_NEIGHBORS  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_initia\$PH::Elasticity\_Kugelstadt2021::performNeighborhood  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_initia\$RH::Elasticity\_Kugelstadt2021::precomputeValues  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_iterat\$BH\$VElasticity\_Kugelstadt2021::reset  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_L::SPH::Elasticity\_Kugelstadt2021::rotationMatricesToA  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_lambdaSPH::Elasticity\_Kugelstadt2021::saveState  
     (*C++ member*), 119  
 SPH::Elasticity\_Kugelstadt2021::m\_massDampingEffectivity\_Kugelstadt2021::Solver

(*C++ type*), 118  
SPH::Elasticity\_Kugelstadt2021::step (*C++ function*), 117  
SPH::Elasticity\_Kugelstadt2021::stepElasticity (*C++ member*), 122  
    (*C++ function*), 118  
SPH::Elasticity\_Kugelstadt2021::stepVolumeSolver (*C++ member*), 122  
    (*C++ function*), 118  
SPH::Elasticity\_Kugelstadt2021::STIFFNESS\_DAMPING (*C++ member*), 122  
    (*C++ function*), 118  
SPH::Elasticity\_Peer2018 (*C++ class*), 121  
SPH::Elasticity\_Peer2018::~Elasticity\_Peering, 121  
    (*C++ function*), 121  
SPH::Elasticity\_Peer2018::ALPHA (*C++ member*), 121  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::computeMatrixLSPH::Elasticity\_Peer2018::performNeighborhoodSearch  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::computeRHS SPH::Elasticity\_Peer2018::reset (*C++ function*), 121  
SPH::Elasticity\_Peer2018::computeRotationsBH::Elasticity\_Peer2018::saveState  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::creator (*C++ function*), 121  
    (*C++ type*), 122  
SPH::Elasticity\_Peer2018::deferredInit SPH::Elasticity\_Peer2018::step (*C++ function*), 121  
SPH::Elasticity\_Peer2018::Elasticity\_Peer2018ElasticityBase (*C++ class*), 123  
    (*C++ function*), 121  
SPH::Elasticity\_Peer2018::initParameters SPH::ElasticityBase::~ElasticityBase  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::initValues SPH::ElasticityBase::determineFixedParticles  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::ITERATIONS SPH::ElasticityBase::ElasticityBase  
    (*C++ member*), 121  
    (*C++ function*), 123  
SPH::Elasticity\_Peer2018::loadState SPH::ElasticityBase::FIXED\_BOX\_MAX (*C++ member*), 123  
    (*C++ function*), 121  
SPH::Elasticity\_Peer2018::m\_alpha (*C++ member*), 122  
SPH::Elasticity\_Peer2018::m\_current\_to\_initial (*C++ function*), 123  
    (*C++ member*), 122  
SPH::Elasticity\_Peer2018::m\_F (*C++ member*), 122  
SPH::Elasticity\_Peer2018::m\_initial\_to\_current (*C++ member*), 124  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::m\_initialNeighbors SPH::ElasticityBase::m\_poissonRatio  
    (*C++ member*), 124  
SPH::Elasticity\_Peer2018::m\_iterations SPH::ElasticityBase::m\_youngsModulus  
    (*C++ member*), 122  
SPH::Elasticity\_Peer2018::m\_L (*C++ member*), 122  
SPH::Elasticity\_Peer2018::m\_maxError SPH::ElasticityBase::POISSON\_RATIO (*C++ member*), 123  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::m\_maxIter SPH::ElasticityBase::YOUNGS\_MODULUS  
    (*C++ member*), 124  
SPH::Elasticity\_Peer2018::m\_restVolumes SPH::Emitter (*C++ class*), 124  
    (*C++ function*), 122  
SPH::Elasticity\_Peer2018::m\_solver SPH::Emitter::~Emitter  
    (*C++ function*), 124  
SPH::Elasticity\_Peer2018::m\_stress SPH::Emitter::emitParticles  
    (*C++ function*), 124  
SPH::Elasticity\_Peer2018::matrixVecProd SPH::Emitter::emitParticlesCircle  
    (*C++ function*), 124

function), 124  
SPH::Emitter::Emitter (C++ function), 124  
SPH::Emitter::getNextEmitTime (C++ function), 124  
SPH::Emitter::getObjetId (C++ function), 124  
SPH::Emitter::getPosition (C++ function), 124  
SPH::Emitter::getRotation (C++ function), 124  
SPH::Emitter::getSize (C++ function), 125  
SPH::Emitter::getVelocity (C++ function), 124  
SPH::Emitter::loadState (C++ function), 124  
SPH::Emitter::m\_emitCounter (C++ member), 125  
SPH::Emitter::m\_emitEndTime (C++ member), 125  
SPH::Emitter::m\_emitStartTime (C++ member), 125  
SPH::Emitter::m\_height (C++ member), 125  
SPH::Emitter::m\_model (C++ member), 125  
SPH::Emitter::m\_nextEmitTime (C++ member), 125  
SPH::Emitter::m\_objectId (C++ member), 125  
SPH::Emitter::m\_rotation (C++ member), 125  
SPH::Emitter::m\_type (C++ member), 125  
SPH::Emitter::m\_velocity (C++ member), 125  
SPH::Emitter::m\_width (C++ member), 125  
SPH::Emitter::m\_x (C++ member), 125  
SPH::Emitter::reset (C++ function), 124  
SPH::Emitter::saveState (C++ function), 124  
SPH::Emitter::setEmitEndTime (C++ function), 124  
SPH::Emitter::setEmitStartTime (C++ function), 124  
SPH::Emitter::setNextEmitTime (C++ function), 124  
SPH::Emitter::setObjectId (C++ function), 124  
SPH::Emitter::setPosition (C++ function), 124  
SPH::Emitter::setRotation (C++ function), 124  
SPH::Emitter::setVelocity (C++ function), 124  
SPH::Emitter::step (C++ function), 124  
SPH::EmitterSystem (C++ class), 125  
SPH::EmitterSystem::~EmitterSystem (C++ function), 125  
SPH::EmitterSystem::addEmitter (C++ function), 125  
SPH::EmitterSystem::disableReuseParticles (C++ function), 125  
SPH::EmitterSystem::EmitterSystem (C++ function), 125  
SPH::EmitterSystem::enableReuseParticles (C++ function), 125  
SPH::EmitterSystem::getEmitters (C++ function), 125  
SPH::EmitterSystem::loadState (C++ function), 126  
SPH::EmitterSystem::m\_boxMax (C++ member), 126  
SPH::EmitterSystem::m\_boxMin (C++ member), 126  
SPH::EmitterSystem::m\_emitters (C++ member), 126  
SPH::EmitterSystem::m\_maxParticlesToReusePerStep (C++ member), 126  
SPH::EmitterSystem::m\_model (C++ member), 126  
SPH::EmitterSystem::m\_numberOfEmittedParticles (C++ member), 126  
SPH::EmitterSystem::m\_numReusedParticles (C++ member), 126  
SPH::EmitterSystem::m\_reusedParticles (C++ member), 126  
SPH::EmitterSystem::m\_reuseParticles (C++ member), 126  
SPH::EmitterSystem::numEmittedParticles (C++ function), 126  
SPH::EmitterSystem::numEmitters (C++ function), 125  
SPH::EmitterSystem::numReusedParticles (C++ function), 126  
SPH::EmitterSystem::reset (C++ function), 126  
SPH::EmitterSystem::reuseParticles (C++ function), 126  
SPH::EmitterSystem::saveState (C++ function), 126  
SPH::EmitterSystem::step (C++ function), 126  
SPH::FieldDescription (C++ struct), 83  
SPH::FieldDescription::FieldDescription (C++ function), 83  
SPH::FieldDescription::getFct (C++ member), 84  
SPH::FieldDescription::name (C++ member), 84  
SPH::FieldDescription::storeData (C++ member), 84  
SPH::FieldDescription::type (C++ member), 84  
SPH::FieldType (C++ enum), 212  
SPH::FieldType::Matrix3 (C++ enumerator), 212  
SPH::FieldType::Matrix6 (C++ enumerator), 212

212  
 SPH::FieldType::Scalar (*C++ enumerator*), 212  
 SPH::FieldType::UInt (*C++ enumerator*), 212  
 SPH::FieldType::Vector3 (*C++ enumerator*),  
     212  
 SPH::FieldType::Vector6 (*C++ enumerator*),  
     212  
 SPH::FluidModel (*C++ class*), 127  
 SPH::FluidModel::~FluidModel (*C++ function*), 127  
 SPH::FluidModel::addField (*C++ function*),  
     127  
 SPH::FluidModel::computeDragForce (*C++ function*), 128  
 SPH::FluidModel::computeElasticity (*C++ function*), 128  
 SPH::FluidModel::computeSurfaceTension  
     (*C++ function*), 128  
 SPH::FluidModel::computeViscosity (*C++ function*), 128  
 SPH::FluidModel::computeVorticity (*C++ function*), 128  
 SPH::FluidModel::deferredInit (*C++ function*), 127  
 SPH::FluidModel::DENSITY0 (*C++ member*),  
     130  
 SPH::FluidModel::DRAG\_METHOD (*C++ member*),  
     130  
 SPH::FluidModel::ELASTICITY\_METHOD (*C++ member*), 130  
 SPH::FluidModel::emittedParticles (*C++ function*), 127  
 SPH::FluidModel::FluidModel (*C++ function*),  
     127  
 SPH::FluidModel::getDragBase (*C++ function*), 128  
 SPH::FluidModel::getDragMethod (*C++ function*), 128  
 SPH::FluidModel::getElasticityBase (*C++ function*), 128  
 SPH::FluidModel::getElasticityMethod  
     (*C++ function*), 128  
 SPH::FluidModel::getEmitterSystem (*C++ function*), 127  
 SPH::FluidModel::getField (*C++ function*),  
     127  
 SPH::FluidModel::getFields (*C++ function*),  
     127  
 SPH::FluidModel::getId (*C++ function*), 127  
 SPH::FluidModel::getNumActiveParticles0  
     (*C++ function*), 127  
 SPH::FluidModel::getPointSetIndex (*C++ function*), 127  
 SPH::FluidModel::getSurfaceTensionBase  
     (*C++ function*), 128  
 SPH::FluidModel::getSurfaceTensionMethod  
     (*C++ function*), 127  
 SPH::FluidModel::getViscosityBase (*C++ function*), 128  
 SPH::FluidModel::getViscosityMethod  
     (*C++ function*), 128  
 SPH::FluidModel::getVorticityBase (*C++ function*), 128  
 SPH::FluidModel::getVorticityMethod  
     (*C++ function*), 128  
 SPH::FluidModel::init (*C++ function*), 127  
 SPH::FluidModel::initMasses (*C++ function*),  
     130  
 SPH::FluidModel::initModel (*C++ function*),  
     127  
 SPH::FluidModel::initParameters  
     (*C++ function*), 130  
 SPH::FluidModel::loadState (*C++ function*),  
     128  
 SPH::FluidModel::m\_a (*C++ member*), 130  
 SPH::FluidModel::m\_density (*C++ member*),  
     130  
 SPH::FluidModel::m\_density0 (*C++ member*),  
     131  
 SPH::FluidModel::m\_drag (*C++ member*), 131  
 SPH::FluidModel::m\_dragMethod (*C++ member*), 131  
 SPH::FluidModel::m\_dragMethodChanged  
     (*C++ member*), 131  
 SPH::FluidModel::m\_elasticity (*C++ member*), 131  
 SPH::FluidModel::m\_elasticityMethod  
     (*C++ member*), 131  
 SPH::FluidModel::m\_elasticityMethodChanged  
     (*C++ member*), 131  
 SPH::FluidModel::m\_emitterSystem (*C++ member*), 130  
 SPH::FluidModel::m\_fields (*C++ member*),  
     131  
 SPH::FluidModel::m\_id (*C++ member*), 130  
 SPH::FluidModel::m\_masses (*C++ member*),  
     130  
 SPH::FluidModel::m\_numActiveParticles  
     (*C++ member*), 131  
 SPH::FluidModel::m\_numActiveParticles0  
     (*C++ member*), 131  
 SPH::FluidModel::m\_objectId (*C++ member*),  
     130  
 SPH::FluidModel::m\_objectId0 (*C++ member*), 130  
 SPH::FluidModel::m\_particleId (*C++ member*), 130  
 SPH::FluidModel::m\_particleState (*C++*

member), 130  
SPH::FluidModel::m\_pointSetIndex (C++ member), 131  
SPH::FluidModel::m\_surfaceTension (C++ member), 130  
SPH::FluidModel::m\_surfaceTensionMethod (C++ member), 130  
SPH::FluidModel::m\_surfaceTensionMethodChanged (C++ member), 131  
SPH::FluidModel::m\_V (C++ member), 130  
SPH::FluidModel::m\_v (C++ member), 130  
SPH::FluidModel::m\_v0 (C++ member), 130  
SPH::FluidModel::m\_viscosity (C++ member), 130  
SPH::FluidModel::m\_viscosityMethod (C++ member), 130  
SPH::FluidModel::m\_viscosityMethodChanged (C++ member), 131  
SPH::FluidModel::m\_vorticity (C++ member), 131  
SPH::FluidModel::m\_vorticityMethod (C++ member), 131  
SPH::FluidModel::m\_vorticityMethodChanged (C++ member), 131  
SPH::FluidModel::m\_x (C++ member), 130  
SPH::FluidModel::m\_x0 (C++ member), 130  
SPH::FluidModel::NUM\_PARTICLES (C++ member), 130  
SPH::FluidModel::NUM\_REUSE\_PARTICLES (C++ member), 130  
SPH::FluidModel::numActiveParticles (C++ function), 127  
SPH::FluidModel::numberOfFields (C++ function), 127  
SPH::FluidModel::numberOfParticles (C++ function), 127  
SPH::FluidModel::numParticles (C++ function), 127  
SPH::FluidModel::operator= (C++ function), 127  
SPH::FluidModel::performNeighborhoodSearch (C++ function), 127  
SPH::FluidModel::releaseFluidParticles (C++ function), 130  
SPH::FluidModel::removeFieldByName (C++ function), 127  
SPH::FluidModel::reset (C++ function), 127  
SPH::FluidModel::resizeFluidParticles (C++ function), 130  
SPH::FluidModel::saveState (C++ function), 128  
SPH::FluidModel::setDensity0 (C++ function), 127  
SPH::FluidModel::setDragMethod (C++ func-  
tion), 128  
SPH::FluidModel::setDragMethodChangedCallback (C++ function), 128  
SPH::FluidModel::setElasticityMethod (C++ function), 128  
SPH::FluidModel::setElasticityMethodChangedCallback (C++ function), 128  
SPH::FluidModel::setNumActiveParticles (C++ function), 127  
SPH::FluidModel::setNumActiveParticles0 (C++ function), 127  
SPH::FluidModel::setSurfaceMethodChangedCallback (C++ function), 128  
SPH::FluidModel::setSurfaceTensionMethod (C++ function), 128  
SPH::FluidModel::setViscosityMethod (C++ function), 128  
SPH::FluidModel::setViscosityMethodChangedCallback (C++ function), 128  
SPH::FluidModel::setVorticityMethod (C++ function), 128  
SPH::FluidModel::setVorticityMethodChangedCallback (C++ function), 128  
SPH::FluidModel::SURFACE\_TENSION\_METHOD (C++ member), 130  
SPH::FluidModel::VISCOSITY\_METHOD (C++ member), 130  
SPH::FluidModel::VORTICITY\_METHOD (C++ member), 130  
SPH::gaussian\_abscissae\_1 (C++ member), 221  
SPH::gaussian\_n\_1 (C++ member), 221  
SPH::gaussian\_weights\_1 (C++ member), 222  
SPH::GaussQuadrature (C++ class), 131  
SPH::GaussQuadrature::Domain (C++ type), 131  
SPH::GaussQuadrature::exportSamples (C++ function), 131  
SPH::GaussQuadrature::Integrand (C++ type), 131  
SPH::JacobiPreconditioner1D (C++ class), 132  
SPH::JacobiPreconditioner1D::\_solve\_impl (C++ function), 132  
SPH::JacobiPreconditioner1D::analyzePattern (C++ function), 132  
SPH::JacobiPreconditioner1D::cols (C++ function), 132  
SPH::JacobiPreconditioner1D::compute (C++ function), 132  
SPH::JacobiPreconditioner1D::DiagonalMatrixElement (C++ type), 132

```

SPH::JacobiPreconditioner1D::factorize   SPH::JacobiPreconditioner3D::m(userData
    (C++ function), 132                  (C++ member), 134
SPH::JacobiPreconditioner1D::info (C++   SPH::JacobiPreconditioner3D::rows (C++
    function), 132                      function), 133
SPH::JacobiPreconditioner1D::init (C++   SPH::JacobiPreconditioner3D::solve (C++
    function), 132                      function), 133
SPH::JacobiPreconditioner1D::JacobiPrecoSPH::MatrixReplacement::StorageIndex
    (C++ function), 132                  (C++ type), 133
SPH::JacobiPreconditioner1D::m_diagonale$BMenFctbbPreconditioner3D::[anonymous]
    (C++ member), 133                  (C++ enum), 133
SPH::JacobiPreconditioner1D::m_dim (C++   SPH::JacobiPreconditioner3D::[anonymous]::ColsAtCom
    member), 133                      (C++ enumerator), 133
SPH::JacobiPreconditioner1D::m_invDiag   SPH::JacobiPreconditioner3D::[anonymous]::MaxColsAt
    (C++ member), 133                  (C++ enumerator), 133
SPH::JacobiPreconditioner1D::mUserData   SPH::MathFunctions (C++ class), 134
    (C++ member), 133                  SPH::MathFunctions::eigenDecomposition
SPH::JacobiPreconditioner1D::rows (C++   (C++ function), 134
    function), 132                      SPH::MathFunctions::extractRotation
SPH::JacobiPreconditioner1D::solve (C++   (C++ function), 134
    function), 132                      SPH::MathFunctions::getOrthogonalVectors
SPH::JacobiPreconditioner1D::StorageIndex   SPH::MathFunctions::jacobiRotate (C++
    (C++ type), 132                  function), 134
SPH::JacobiPreconditioner1D::[anonymous]   SPH::MathFunctions::pseudoInverse (C++
    (C++ enum), 132                  function), 134
SPH::JacobiPreconditioner1D::[anonymous]::ColsAtTime
    (C++ enumerator), 132              SPH::MathFunctions::svdWithInversionHandling
SPH::JacobiPreconditioner1D::[anonymous]::MaxCol(CatJump), 134
    (C++ enumerator), 132              SPH::MatrixReplacement (C++ class), 135
SPH::JacobiPreconditioner3D (C++ class),   SPH::MatrixReplacement::cols (C++ func-
    133                                tion), 135
SPH::JacobiPreconditioner3D::_solve_implSPH::MatrixReplacement::getMatrixVecProdFct
    (C++ function), 133                  (C++ function), 135
SPH::JacobiPreconditioner3D::analyzePattSPH::MatrixReplacement::getUserData
    (C++ function), 133                  (C++ function), 135
SPH::JacobiPreconditioner3D::cols (C++   SPH::MatrixReplacement::m_dim (C++ mem-
    function), 133                      ber), 136
SPH::JacobiPreconditioner3D::compute   SPH::MatrixReplacement::m_matrixVecProdFct
    (C++ function), 133                  (C++ member), 136
SPH::JacobiPreconditioner3D::DiagonalMatSPH::MatrixReplacement::m(userData (C++
    (C++ type), 133                  member), 136
SPH::JacobiPreconditioner3D::factorize   SPH::MatrixReplacement::MatrixReplacement
    (C++ function), 133                  (C++ function), 135
SPH::JacobiPreconditioner3D::info (C++   SPH::MatrixReplacement::MatrixVecProdFct
    function), 133                      (C++ type), 135
SPH::JacobiPreconditioner3D::init (C++   SPH::MatrixReplacement::operator* (C++
    function), 133                      function), 135
SPH::JacobiPreconditioner3D::JacobiPrecoSPH::MatrixReplacement::RealScalar (C++
    (C++ function), 133                  type), 135
SPH::JacobiPreconditioner3D::m_diagonale$BMenMatrixReplacement::rows (C++ func-
    (C++ member), 134                  tion), 135
SPH::JacobiPreconditioner3D::m_dim (C++   SPH::MatrixReplacement::Scalar (C++ type),
    member), 134                      135
SPH::JacobiPreconditioner3D::m_invDiag   SPH::MatrixReplacement::StorageIndex
    (C++ member), 134                  (C++ type), 135

```

```

SPH::MatrixReplacement::[anonymous] (C++ function), 138
SPH::MatrixReplacement::[anonymous]::ColsAtComp (C++ function), 138
SPH::MatrixReplacement::[anonymous]::IsRowMajor (C++ function), 138
SPH::MatrixReplacement::[anonymous]::MaxColsAtComp (C++ function), 138
SPH::MicropolarModel_Bender2017 (C++ class), 136
SPH::MicropolarModel_Bender2017::~MicropolarModel_Bender2017 Active (C++ enumerator), 212
SPH::MicropolarModel_Bender2017::creatorSPH::ParticleState::AnimatedByEmitter (C++ function), 212
SPH::MicropolarModel_Bender2017::INERTIASPNVER&Elt;icleState::Fixed (C++ enumerator), 212
SPH::MicropolarModel_Bender2017::initParSPH::PoissonDiskSampling (C++ class), 139
SPH::MicropolarModel_Bender2017::m_angularAccel (C++ struct), 84
SPH::MicropolarModel_Bender2017::m_inertiaInver (C++ function), 84
SPH::MicropolarModel_Bender2017::m_omega (C++ struct), 84, 139
SPH::MicropolarModel_Bender2017::m_viscosityOmega (C++ function), 84, 139
SPH::MicropolarModel_Bender2017::MicropolarMode (C++ member), 135, 139
SPH::MicropolarModel_Bender2017::performNeighbo (C++ member), 139
SPH::MicropolarModel_Bender2017::reset (C++ function), 136
SPH::MicropolarModel_Bender2017::step (C++ function), 136
SPH::MicropolarModel_Bender2017::VISCOSITY_OMEGA (C++ member), 85, 140
SPH::NonPressureForceBase (C++ class), 138
SPH::NonPressureForceBase::~NonPressureFSPH::PoissonDiskSampling::PoissonDiskSampling (C++ function), 138
SPH::NonPressureForceBase::deferredInit (C++ function), 138
SPH::NonPressureForceBase::emittedPartic$BH::Poly6Kernel (C++ class), 140
SPH::NonPressureForceBase::getModel (C++ function), 138
SPH::NonPressureForceBase::init (C++ function), 138
SPH::NonPressureForceBase::loadState (C++ function), 138
SPH::NonPressureForceBase::m_model (C++ member), 138
SPH::NonPressureForceBase::NonPressureForceBase (C++ function), 138
SPH::NonPressureForceBase::operator= (C++ function), 140
SPH::NonPressureForceBase::performNeighborhoodSearch (C++ function), 138
SPH::NonPressureForceBase::reset (C++ function), 138
SPH::NonPressureForceBase::saveState (C++ function), 138
SPH::NonPressureForceBase::step (C++ function), 138
SPH::ParticleState (C++ enum), 212
SPH::PoissonDiskSampling::CellPosHasher (C++ struct), 84
SPH::PoissonDiskSampling::HashEntry (C++ struct), 84, 139
SPH::PoissonDiskSampling::HashEntry::operator() (C++ function), 84
SPH::PoissonDiskSampling::HashEntry::samples (C++ member), 85, 139
SPH::PoissonDiskSampling::HashEntry::startIndex (C++ member), 85, 139
SPH::PoissonDiskSampling::InitialPointInfo (C++ struct), 85, 139
SPH::PoissonDiskSampling::InitialPointInfo::cP (C++ member), 85, 140
SPH::PoissonDiskSampling::InitialPointInfo::ID (C++ member), 85, 140
SPH::PoissonDiskSampling::InitialPointInfo::pos (C++ member), 85, 140
SPH::PoissonDiskSampling::sampleMesh (C++ function), 139
SPH::Poly6Kernel (C++ class), 140
SPH::Poly6Kernel::getRadius (C++ function), 140
SPH::Poly6Kernel::gradW (C++ function), 140
SPH::Poly6Kernel::laplacianW (C++ function), 140
SPH::Poly6Kernel::m_k (C++ member), 140
SPH::Poly6Kernel::m_l (C++ member), 140
SPH::Poly6Kernel::m_m (C++ member), 140
SPH::Poly6Kernel::m_radius (C++ member), 140
SPH::Poly6Kernel::m_W_zero (C++ member), 140

```

SPH::Poly6Kernel::setRadius (*C++ function*), 140  
 SPH::Poly6Kernel::W (*C++ function*), 140  
 SPH::Poly6Kernel::W\_zero (*C++ function*), 140  
 SPH::PrecomputedKernel (*C++ class*), 141  
 SPH::PrecomputedKernel::getRadius (*C++ function*), 141  
 SPH::PrecomputedKernel::gradW (*C++ function*), 141  
 SPH::PrecomputedKernel::m\_gradW (*C++ member*), 141  
 SPH::PrecomputedKernel::m\_invStepSize (*C++ member*), 141  
 SPH::PrecomputedKernel::m\_radius (*C++ member*), 141  
 SPH::PrecomputedKernel::m\_radius2 (*C++ member*), 141  
 SPH::PrecomputedKernel::m\_W (*C++ member*), 141  
 SPH::PrecomputedKernel::m\_W\_zero (*C++ member*), 141  
 SPH::PrecomputedKernel::setRadius (*C++ function*), 141  
 SPH::PrecomputedKernel::W (*C++ function*), 141  
 SPH::PrecomputedKernel::W\_zero (*C++ function*), 141  
 SPH::RegularSampling2D (*C++ class*), 142  
 SPH::RegularSampling2D::RegularSampling2D (*C++ function*), 142  
 SPH::RegularSampling2D::sampleMesh (*C++ function*), 142  
 SPH::RegularTriangleSampling (*C++ class*), 142  
 SPH::RegularTriangleSampling::RegularTriangleSampling (*C++ function*), 143  
 SPH::RegularTriangleSampling::sampleMesh (*C++ function*), 143  
 SPH::RigidBodyObject (*C++ class*), 143  
 SPH::RigidBodyObject::~RigidBodyObject (*C++ function*), 143  
 SPH::RigidBodyObject::addForce (*C++ function*), 144  
 SPH::RigidBodyObject::addTorque (*C++ function*), 144  
 SPH::RigidBodyObject::getAngularVelocity (*C++ function*), 144  
 SPH::RigidBodyObject::getFaces (*C++ function*), 144  
 SPH::RigidBodyObject::getMass (*C++ function*), 143  
 SPH::RigidBodyObject::getPosition (*C++ function*), 143  
 SPH::RigidBodyObject::getRotation (*C++ function*), 144  
 SPH::RigidBodyObject::getVelocity (*C++ function*), 144  
 SPH::RigidBodyObject::getVertexNormals (*C++ function*), 144  
 SPH::RigidBodyObject::getVertices (*C++ function*), 144  
 SPH::RigidBodyObject::getWorldSpacePosition (*C++ function*), 144  
 SPH::RigidBodyObject::getWorldSpaceRotation (*C++ function*), 144  
 SPH::RigidBodyObject::isAnimated (*C++ function*), 143  
 SPH::RigidBodyObject::isDynamic (*C++ function*), 143  
 SPH::RigidBodyObject::m\_isAnimated (*C++ member*), 144  
 SPH::RigidBodyObject::RigidBodyObject (*C++ function*), 143  
 SPH::RigidBodyObject::setAngularVelocity (*C++ function*), 144  
 SPH::RigidBodyObject::setIsAnimated (*C++ function*), 143  
 SPH::RigidBodyObject::setPosition (*C++ function*), 144  
 SPH::RigidBodyObject::setRotation (*C++ function*), 144  
 SPH::RigidBodyObject::setVelocity (*C++ function*), 144  
 SPH::RigidBodyObject::updateMeshTransformation (*C++ function*), 144  
 SPH::SimpleQuadrature (*C++ class*), 144  
 SPH::SimpleQuadrature::determineSamplePointsInCircle (*C++ function*), 145  
 SPH::SimpleQuadrature::determineSamplePointsInSphere (*C++ function*), 145  
 SPH::SimpleQuadrature::Domain (*C++ type*), 144  
 SPH::SimpleQuadrature::Integrand (*C++ type*), 144  
 SPH::SimpleQuadrature::integrate (*C++ function*), 145  
 SPH::SimpleQuadrature::m\_samplePoints (*C++ member*), 145  
 SPH::SimpleQuadrature::m\_volume (*C++ member*), 145  
 SPH::Simulation (*C++ class*), 145  
 SPH::Simulation::~Simulation (*C++ function*), 146  
 SPH::Simulation::addBoundaryModel (*C++ function*), 146  
 SPH::Simulation::addDragMethod (*C++ function*), 147  
 SPH::Simulation::addElasticityMethod

(*C++ function*), 147  
SPH::Simulation::addFluidInfo (*C++ function*), 146  
SPH::Simulation::addFluidModel (*C++ function*), 146  
SPH::Simulation::addSurfaceTensionMethod (*C++ function*), 147  
SPH::Simulation::addViscosityMethod (*C++ function*), 148  
SPH::Simulation::addVorticityMethod (*C++ function*), 148  
SPH::Simulation::animateParticles (*C++ function*), 147  
SPH::Simulation::BOUNDARY\_HANDLING\_METHOD (*C++ member*), 149  
SPH::Simulation::CFL\_FACTOR (*C++ member*), 148  
SPH::Simulation::CFL\_MAX\_TIMESTEP\_SIZE (*C++ member*), 148  
SPH::Simulation::CFL\_METHOD (*C++ member*), 148  
SPH::Simulation::CFL\_MIN\_TIMESTEP\_SIZE (*C++ member*), 148  
SPH::Simulation::computeNonPressureForce (*C++ function*), 147  
SPH::Simulation::deferredInit (*C++ function*), 146  
SPH::Simulation::emitParticles (*C++ function*), 147  
SPH::Simulation::emittedParticles (*C++ function*), 147  
SPH::Simulation::ENABLE\_Z\_SORT (*C++ member*), 148  
SPH::Simulation::ENUM\_AKINCI2012 (*C++ member*), 149  
SPH::Simulation::ENUM\_BENDER2019 (*C++ member*), 149  
SPH::Simulation::ENUM\_CFL\_ITER (*C++ member*), 149  
SPH::Simulation::ENUM\_CFL\_NONE (*C++ member*), 149  
SPH::Simulation::ENUM\_CFL\_STANDARD (*C++ member*), 149  
SPH::Simulation::ENUM\_GRADKERNEL\_CUBIC (*C++ member*), 148  
SPH::Simulation::ENUM\_GRADKERNEL\_CUBIC\_2D (*C++ member*), 149  
SPH::Simulation::ENUM\_GRADKERNEL\_POLY6 (*C++ member*), 149  
SPH::Simulation::ENUM\_GRADKERNEL\_PRECOMPUTED\_CUBIC (*C++ member*), 149  
SPH::Simulation::ENUM\_GRADKERNEL\_SPIKY (*C++ member*), 148  
SPH::Simulation::ENUM\_KERNEL\_WENDLANDQUINTICC2 (*C++ member*), 149  
SPH::Simulation::ENUM\_KERNEL\_WENDLANDQUINTICC2\_2D (*C++ member*), 148  
SPH::Simulation::ENUM\_KERNEL\_POLY6 (*C++ member*), 148  
SPH::Simulation::ENUM\_KERNEL\_PRECOMPUTED\_CUBIC (*C++ member*), 148  
SPH::Simulation::ENUM\_KERNEL\_SPIKY (*C++ member*), 148  
SPH::Simulation::ENUM\_KERNEL\_WENDLANDQUINTICC2 (*C++ member*), 148  
SPH::Simulation::ENUM\_KERNEL\_WENDLANDQUINTICC2\_2D (*C++ member*), 148  
SPH::Simulation::ENUM\_KOSCHIER2017 (*C++ member*), 149  
SPH::Simulation::ENUM\_SIMULATION\_DFSPH (*C++ member*), 149  
SPH::Simulation::ENUM\_SIMULATION\_ICSPH (*C++ member*), 149  
SPH::Simulation::ENUM\_SIMULATION\_IISPH (*C++ member*), 149  
SPH::Simulation::ENUM\_SIMULATION\_PBF (*C++ member*), 149  
SPH::Simulation::ENUM\_SIMULATION\_PCISPH (*C++ member*), 149  
SPH::Simulation::ENUM\_SIMULATION\_PF (*C++ member*), 149  
SPH::Simulation::ENUM\_SIMULATION\_WCSPH (*C++ member*), 149  
SPH::FluidInfo (*C++ struct*), 85, 150  
SPH::FluidInfo::box (*C++ member*), 86, 150  
SPH::FluidInfo::emitter\_emitEndTime (*C++ member*), 86, 151  
SPH::FluidInfo::emitter\_emitStartTime (*C++ member*), 86, 151  
SPH::FluidInfo::emitter\_height (*C++ member*), 86, 151  
SPH::FluidInfo::emitter\_type (*C++ member*), 86, 151  
SPH::FluidInfo::emitter\_velocity (*C++ member*), 86, 151  
SPH::FluidInfo::width (*C++ member*), 86, 151  
SPH::FluidInfo::hasSameParticleSampling (*C++ function*), 86, 150  
SPH::FluidInfo::id (*C++ member*), 86, 150  
SPH::FluidInfo::initialAngularVelocity (*C++ function*), 86, 150

```

(C++ member), 86, 151
SPH::Simulation::FluidInfo::initialVelocity (C++ member), 86, 151
SPH::Simulation::FluidInfo::invert (C++ member), 86, 151
SPH::Simulation::FluidInfo::mode (C++ member), 86, 151
SPH::Simulation::FluidInfo::numParticles (C++ member), 86, 150
SPH::Simulation::FluidInfo::resolutionSD (C++ member), 86, 151
SPH::Simulation::FluidInfo::rotation (C++ member), 86, 150
SPH::Simulation::FluidInfo::samplesFile (C++ member), 86, 150
SPH::Simulation::FluidInfo::scale (C++ member), 86, 151
SPH::Simulation::FluidInfo::translation (C++ member), 86, 150
SPH::Simulation::FluidInfo::type (C++ member), 86, 150
SPH::Simulation::FluidInfo::visMeshFile (C++ member), 86, 150
SPH::Simulation::getAnimationFieldSystem (C++ function), 146
SPH::Simulation::getBoundaryHandlingMethod (C++ function), 146
SPH::Simulation::getBoundaryModel (C++ function), 146
SPH::Simulation::getBoundaryModelFromPointSet (C++ function), 146
SPH::Simulation::getCachePath (C++ function), 147
SPH::Simulation::getCurrent (C++ function), 148
SPH::Simulation::getDragMethods (C++ function), 147
SPH::Simulation::getElasticityMethods (C++ function), 147
SPH::Simulation::getFluidInfo (C++ function), 146
SPH::Simulation::getFluidInfos (C++ function), 146
SPH::Simulation::getFluidModel (C++ function), 146
SPH::Simulation::getFluidModelFromPointSet (C++ function), 146
SPH::Simulation::getGradKernel (C++ function), 146
SPH::Simulation::getKernel (C++ function), 146
SPH::Simulation::getNeighborhoodSearch (C++ function), 147
SPH::Simulation::getParticleRadius (C++ function), 147
SPH::Simulation::getSimulationMethod (C++ function), 146
SPH::Simulation::getSupportRadius (C++ function), 147
SPH::Simulation::getSurfaceTensionMethods (C++ function), 147
SPH::Simulation::getTimeStep (C++ function), 147
SPH::Simulation::getUseCache (C++ function), 147
SPH::Simulation::getViscosityMethods (C++ function), 148
SPH::Simulation::getVorticityMethods (C++ function), 148
SPH::Simulation::GRAD_KERNEL_METHOD (C++ member), 148
SPH::Simulation::GRAVITATION (C++ member), 148
SPH::Simulation::hasCurrent (C++ function), 148
SPH::Simulation::init (C++ function), 146
SPH::Simulation::initKernels (C++ function), 147
SPH::Simulation::initParameters (C++ function), 147
SPH::Simulation::is2DSimulation (C++ function), 147
SPH::Simulation::isSimulationInitialized (C++ function), 146
SPH::Simulation::KERNEL_METHOD (C++ member), 148
SPH::Simulation::loadState (C++ function), 147
SPH::Simulation::m_animationFieldSystem (C++ member), 149
SPH::Simulation::m_boundaryHandlingMethod (C++ member), 150
SPH::Simulation::m_boundaryModels (C++ member), 149
SPH::Simulation::m_cachePath (C++ member), 150
SPH::Simulation::m_cflFactor (C++ member), 149
SPH::Simulation::m_cflMaxTimeStepSize (C++ member), 149
SPH::Simulation::m_cflMethod (C++ member), 149
SPH::Simulation::m_cflMinTimeStepSize (C++ member), 149
SPH::Simulation::m_dragMethods (C++ member), 150
SPH::Simulation::m_elasticityMethods (C++ member), 150

```

SPH::Simulation::m\_enableZSort (*C++ member*), 150  
SPH::Simulation::m\_fluidInfos (*C++ member*), 149  
SPH::Simulation::m\_fluidModels (*C++ member*), 149  
SPH::Simulation::m\_gradKernelFct (*C++ member*), 150  
SPH::Simulation::m\_gradKernelMethod (*C++ member*), 150  
SPH::Simulation::m\_gravitation (*C++ member*), 150  
SPH::Simulation::m\_kernelFct (*C++ member*), 150  
SPH::Simulation::m\_kernelMethod (*C++ member*), 149  
SPH::Simulation::m\_neighborhoodSearch (*C++ member*), 149  
SPH::Simulation::m\_particleRadius (*C++ member*), 150  
SPH::Simulation::m\_sim2D (*C++ member*), 150  
SPH::Simulation::m\_simulationIsInitialized (*C++ member*), 150  
SPH::Simulation::m\_simulationMethod (*C++ member*), 150  
SPH::Simulation::m\_simulationMethodChanged (*C++ member*), 150  
SPH::Simulation::m\_supportRadius (*C++ member*), 150  
SPH::Simulation::m\_surfaceTensionMethods (*C++ member*), 150  
SPH::Simulation::m\_timeStep (*C++ member*), 150  
SPH::Simulation::m\_useCache (*C++ member*), 150  
SPH::Simulation::m\_viscoMethods (*C++ member*), 150  
SPH::Simulation::m\_vorticityMethods (*C++ member*), 150  
SPH::Simulation::m\_W\_zero (*C++ member*), 150  
SPH::Simulation::NonPressureForceMethod (*C++ struct*), 87, 151  
SPH::Simulation::NonPressureForceMethod::SPHcrsSimulation::updateTimeStepSizeCFL (*C++ member*), 87, 151  
SPH::Simulation::NonPressureForceMethod::SPHhidSimulation::zSortEnabled (*C++ function*), 87, 151  
SPH::Simulation::NonPressureForceMethod::SPHnsimulationDataDFSPH (*C++ class*), 151  
SPH::Simulation::numberOfBoundaryModels (*C++ function*), 146  
SPH::Simulation::numberOfFluidModels (*C++ function*), 146  
SPH::Simulation::operator= (*C++ function*), 146  
SPH::Simulation::PARTICLE\_RADIUS (*C++ member*), 148  
SPH::Simulation::performNeighborhoodSearch (*C++ function*), 147  
SPH::Simulation::performNeighborhoodSearchSort (*C++ function*), 147  
SPH::Simulation::PrecomputedCubicKernel (*C++ type*), 145  
SPH::Simulation::registerNonpressureForces (*C++ function*), 149  
SPH::Simulation::reset (*C++ function*), 146  
SPH::Simulation::saveState (*C++ function*), 147  
SPH::Simulation::setBoundaryHandlingMethod (*C++ function*), 146  
SPH::Simulation::setCachePath (*C++ function*), 147  
SPH::Simulation::setCurrent (*C++ function*), 148  
SPH::Simulation::setGradKernel (*C++ function*), 146  
SPH::Simulation::setKernel (*C++ function*), 146  
SPH::Simulation::setParticleRadius (*C++ function*), 147  
SPH::Simulation::setSimulationInitialized (*C++ function*), 146  
SPH::Simulation::setSimulationMethod (*C++ function*), 147  
SPH::Simulation::setSimulationMethodChangedCallback (*C++ function*), 147  
SPH::Simulation::setUseCache (*C++ function*), 147  
SPH::Simulation::SIM\_2D (*C++ member*), 148  
SPH::Simulation::Simulation (*C++ function*), 146  
SPH::Simulation::SIMULATION\_METHOD (*C++ member*), 149  
SPH::Simulation::updateBoundaryVolume (*C++ function*), 146  
SPH::Simulation::updateTimeStepSize (*C++ function*), 147  
SPH::Simulation::updateTimeStepSizeCFL (*C++ function*), 147  
SPH::SimulationDataDFSPH::~SimulationDataDFSPH (*C++ function*), 152  
SPH::SimulationDataDFSPH::cleanup (*C++ function*), 152  
SPH::SimulationDataDFSPH::emittedParticles (*C++ function*), 152

```

SPH::SimulationDataDFSPH::init (C++ function), 152
SPH::SimulationDataDFSPH::m_density_adv (C++ member), 152
SPH::SimulationDataDFSPH::m_factor (C++ member), 152
SPH::SimulationDataDFSPH::m_kappa (C++ member), 152
SPH::SimulationDataDFSPH::m_kappaV (C++ member), 152
SPH::SimulationDataDFSPH::performNeighborhoodSearch (C++ function), 152
SPH::SimulationDataDFSPH::reset (C++ function), 152
SPH::SimulationDataDFSPH::SimulationDataDFSPH (C++ function), 152
SPH::SimulationDataICSPH (C++ class), 153
SPH::SimulationDataICSPH::~SimulationDataICSPH (C++ function), 153
SPH::SimulationDataICSPH::cleanup (C++ function), 153
SPH::SimulationDataICSPH::emittedParticles (C++ function), 153
SPH::SimulationDataICSPH::init (C++ function), 153
SPH::SimulationDataICSPH::m_aii (C++ member), 154
SPH::SimulationDataICSPH::m_density_adv (C++ member), 154
SPH::SimulationDataICSPH::m_pressure (C++ member), 154
SPH::SimulationDataICSPH::m_pressureAccel (C++ member), 154
SPH::SimulationDataICSPH::m_pressureGrad (C++ member), 154
SPH::SimulationDataICSPH::performNeighborhoodSearch (C++ function), 153
SPH::SimulationDataICSPH::reset (C++ function), 153
SPH::SimulationDataICSPH::SimulationDataICSPH (C++ class), 153
SPH::SimulationDataIISPH (C++ class), 154
SPH::SimulationDataIISPH::~SimulationDataIISPH (C++ function), 154
SPH::SimulationDataIISPH::cleanup (C++ function), 154
SPH::SimulationDataIISPH::emittedParticles (C++ function), 154
SPH::SimulationDataIISPH::init (C++ function), 154
SPH::SimulationDataIISPH::m_aii (C++ member), 155
SPH::SimulationDataIISPH::m_density_adv (C++ member), 155
SPH::SimulationDataIISPH::m_dii (C++ member), 155
SPH::SimulationDataIISPH::m_dij_pj (C++ member), 155
SPH::SimulationDataIISPH::m_lastPressure (C++ member), 155
SPH::SimulationDataIISPH::m_pressure (C++ member), 155
SPH::SimulationDataIISPH::m_pressureAccel (C++ member), 155
SPH::SimulationDataIISPH::performNeighborhoodSearch (C++ function), 154
SPH::SimulationDataIISPH::reset (C++ function), 154
SPH::SimulationDataIISPH::SimulationDataIISPH (C++ function), 154
SPH::SimulationDataPBF (C++ class), 156
SPH::SimulationDataPBF::~SimulationDataPBF (C++ function), 156
SPH::SimulationDataPBF::cleanup (C++ function), 156
SPH::SimulationDataPBF::emittedParticles (C++ function), 156
SPH::SimulationDataPBF::init (C++ function), 156
SPH::SimulationDataPBF::m_deltaX (C++ member), 157
SPH::SimulationDataPBF::m_lambda (C++ member), 157
SPH::SimulationDataPBF::m_lastX (C++ member), 157
SPH::SimulationDataPBF::m_oldX (C++ member), 157
SPH::SimulationDataPBF::performNeighborhoodSearch (C++ function), 156
SPH::SimulationDataPBF::reset (C++ function), 156
SPH::SimulationDataPBF::SimulationDataPBF (C++ function), 156
SPH::SimulationDataPCISPH (C++ class), 157
SPH::SimulationDataPCISPH::~SimulationDataPCISPH (C++ function), 157
SPH::SimulationDataPCISPH::cleanup (C++ function), 157
SPH::SimulationDataPCISPH::emittedParticles (C++ function), 157
SPH::SimulationDataPCISPH::getPCISPH_ScalingFactor (C++ function), 157
SPH::SimulationDataPCISPH::init (C++ function), 157
SPH::SimulationDataPCISPH::m_densityAdv (C++ member), 158
SPH::SimulationDataPCISPH::m_pcisph_factor (C++ member), 158

```

SPH::SimulationDataPCISPH::m\_predV (*C++ member*), 158      SPH::SimulationDataWCSPH::reset (*C++ function*), 160  
SPH::SimulationDataPCISPH::m\_predX (*C++ member*), 158      SPH::SimulationDataWCSPH::SimulationDataWCSPH (*C++ function*), 160  
SPH::SimulationDataPCISPH::m\_pressure (*C++ member*), 158      SPH::SimulationMethods (*C++ enum*), 213  
SPH::SimulationDataPCISPH::m\_pressureAccel (*C++ member*), 158      SPH::SimulationMethods::DFSPH (*C++ enum*), 213  
SPH::SimulationDataPCISPH::performNeighborhoodSearch (*C++ function*), 157      SPH::SimulationMethods::ICSPH (*C++ enum*), 213  
SPH::SimulationDataPCISPH::reset (*C++ function*), 157      SPH::SimulationMethods::IISPH (*C++ enum*), 213  
SPH::SimulationDataPCISPH::SimulationDataPCISPH (*C++ enumerator*), 213      SPH::SimulationMethods::NumSimulationMethods  
SPH::SimulationDataPF (*C++ class*), 158      SPH::SimulationMethods::PBF (*C++ enumerator*), 213  
SPH::SimulationDataPF::~SimulationDataPFSPH::SimulationMethods::PCISPH (*C++ enumerator*), 213  
SPH::SimulationDataPF::cleanup (*C++ function*), 158      SPH::SimulationMethods::PF (*C++ enumerator*), 213  
SPH::SimulationDataPF::emittedParticles (*C++ function*), 159      SPH::SimulationMethods::WCSPH (*C++ enumerator*), 213  
SPH::SimulationDataPF::init (*C++ function*), 158      SPH::SpikyKernel (*C++ class*), 161  
SPH::SimulationDataPF::m\_mat\_diag (*C++ member*), 159      SPH::SpikyKernel::getRadius (*C++ function*), 161  
SPH::SimulationDataPF::m\_num\_fluid\_neighs (*C++ member*), 159      SPH::SpikyKernel::gradW (*C++ function*), 161  
SPH::SimulationDataPF::m\_old\_position (*C++ member*), 159      SPH::SpikyKernel::m\_k (*C++ member*), 161  
SPH::SimulationDataPF::m\_s (*C++ member*), 159      SPH::SpikyKernel::m\_l (*C++ member*), 161  
SPH::SimulationDataPF::performNeighborhoodSearch (*C++ function*), 158      SPH::SpikyKernel::m\_radius (*C++ member*), 161  
SPH::SimulationDataPF::reset (*C++ function*), 158      SPH::SpikyKernel::W (*C++ function*), 161  
SPH::SimulationDataPF::SimulationDataPF (*C++ function*), 158      SPH::SpikyKernel::W\_zero (*C++ member*), 161  
SPH::SimulationDataWCSPH (*C++ class*), 160      SPH::SpikyKernel::setRadius (*C++ function*), 161  
SPH::SimulationDataWCSPH::~SimulationDataWCSPH (*C++ function*), 160      SPH::StaticRigidBody (*C++ class*), 162  
SPH::SimulationDataWCSPH::cleanup (*C++ function*), 160      SPH::StaticRigidBody::addForce (*C++ function*), 162  
SPH::SimulationDataWCSPH::emittedParticles (*C++ function*), 160      SPH::StaticRigidBody::addTorque (*C++ function*), 162  
SPH::SimulationDataWCSPH::init (*C++ function*), 160      SPH::StaticRigidBody::getAngularVelocity (*C++ function*), 162  
SPH::SimulationDataWCSPH::m\_pressure (*C++ member*), 160      SPH::StaticRigidBody::getFaces (*C++ function*), 162  
SPH::SimulationDataWCSPH::m\_pressureAccel (*C++ member*), 160      SPH::StaticRigidBody::getGeometry (*C++ function*), 162  
SPH::SimulationDataWCSPH::performNeighborhoodSearch (*C++ function*), 160      SPH::StaticRigidBody::getMass (*C++ function*), 162  
SPH::StaticRigidBody::getPosition (*C++ function*), 162  
SPH::StaticRigidBody::getPosition0 (*C++ function*), 162

SPH::StaticRigidBody::getRotation (*C++ function*), 162  
 SPH::StaticRigidBody::getRotation0 (*C++ function*), 162  
 SPH::StaticRigidBody::getVelocity (*C++ function*), 162  
 SPH::StaticRigidBody::getVertexNormals (*C++ function*), 162  
 SPH::StaticRigidBody::getVertices (*C++ function*), 162  
 SPH::StaticRigidBody::getWorldSpacePosition (*C++ function*), 162  
 SPH::StaticRigidBody::getWorldSpaceRotation (*C++ function*), 162  
 SPH::StaticRigidBody::isDynamic (*C++ function*), 162  
 SPH::StaticRigidBody::m-angularVelocity (*C++ member*), 163  
 SPH::StaticRigidBody::m-geometry (*C++ member*), 163  
 SPH::StaticRigidBody::m-q (*C++ member*), 163  
 SPH::StaticRigidBody::m-q0 (*C++ member*), 163  
 SPH::StaticRigidBody::m-velocity (*C++ member*), 163  
 SPH::StaticRigidBody::m-x (*C++ member*), 163  
 SPH::StaticRigidBody::m-x0 (*C++ member*), 163  
 SPH::StaticRigidBody::reset (*C++ function*), 162  
 SPH::StaticRigidBody::setAngularVelocity (*C++ function*), 162  
 SPH::StaticRigidBody::setPosition (*C++ function*), 162  
 SPH::StaticRigidBody::setPosition0 (*C++ function*), 162  
 SPH::StaticRigidBody::setRotation (*C++ function*), 162  
 SPH::StaticRigidBody::setRotation0 (*C++ function*), 162  
 SPH::StaticRigidBody::setVelocity (*C++ function*), 162  
 SPH::StaticRigidBody::setWorldSpacePosition (*C++ function*), 162  
 SPH::StaticRigidBody::setWorldSpaceRotation (*C++ function*), 162  
 SPH::StaticRigidBody::StaticRigidBody (*C++ function*), 162  
 SPH::StaticRigidBody::updateMeshTransform (*C++ function*), 162  
 SPH::SurfaceSamplingMode (*C++ enum*), 213  
 SPH::SurfaceSamplingMode::PoissonDisk  
 SPH::SurfaceSamplingMode::Regular2D  
 (C++ enumerator), 213  
 SPH::SurfaceSamplingMode::RegularTriangle  
 (C++ enumerator), 213  
 SPH::SurfaceTension\_Akinci2013 (*C++ class*), 163  
 SPH::SurfaceTension\_Akinci2013::~SurfaceTension\_Akinci2013  
 (C++ function), 163  
 SPH::SurfaceTension\_Akinci2013::computeNormals  
 (C++ function), 163  
 SPH::SurfaceTension\_Akinci2013::creator  
 (C++ function), 164  
 SPH::SurfaceTension\_Akinci2013::m\_normals  
 (C++ member), 164  
 SPH::SurfaceTension\_Akinci2013::performNeighborhoodSearch  
 (C++ function), 163  
 SPH::SurfaceTension\_Akinci2013::reset  
 (C++ function), 163  
 SPH::SurfaceTension\_Akinci2013::step  
 (C++ function), 163  
 SPH::SurfaceTension\_Akinci2013::SurfaceTension\_Akinci2013  
 (C++ function), 163  
 SPH::SurfaceTension\_Becker2007 (*C++ class*), 164  
 SPH::SurfaceTension\_Becker2007::~SurfaceTension\_Becker2007  
 (C++ function), 164  
 SPH::SurfaceTension\_Becker2007::creator  
 (C++ function), 164  
 SPH::SurfaceTension\_Becker2007::reset  
 (C++ function), 164  
 SPH::SurfaceTension\_Becker2007::step  
 (C++ function), 164  
 SPH::SurfaceTension\_Becker2007::SurfaceTension\_Becker2007  
 (C++ function), 164  
 SPH::SurfaceTension\_He2014 (*C++ class*), 165  
 SPH::SurfaceTension\_He2014::~SurfaceTension\_He2014  
 (C++ function), 165  
 SPH::SurfaceTension\_He2014::creator  
 (C++ function), 165  
 SPH::SurfaceTension\_He2014::m\_color  
 (C++ member), 166  
 SPH::SurfaceTension\_He2014::m\_gradC2  
 (C++ member), 166  
 SPH::SurfaceTension\_He2014::performNeighborhoodSearch  
 (C++ function), 165  
 SPH::SurfaceTension\_He2014::reset (*C++ function*), 165  
 SPH::SurfaceTension\_He2014::step (*C++ function*), 165  
 SPH::SurfaceTension\_He2014::SurfaceTension\_He2014  
 (C++ function), 165  
 SPH::SurfaceTension\_ZorillaRitter2020  
 (C++ class), 166

```

SPH::SurfaceTension_ZorillaRitter2020::~SurfaceTension_ZorillaRitter2020
    (C++ function), 166
SPH::SurfaceTension_ZorillaRitter2020::computeShapeFunction (C++ function), 170
    (C++ function), 166
SPH::SurfaceTension_ZorillaRitter2020::create (C++ function), 171
    (C++ function), 166
SPH::SurfaceTension_ZorillaRitter2020::clearAccelerations (C++ function), 170
    (C++ function), 166
SPH::SurfaceTension_ZorillaRitter2020::computeDensities (C++ function), 170
    (C++ function), 166
SPH::SurfaceTension_ZorillaRitter2020::computeDensityAndGradient (C++ function), 170
    (C++ function), 166
SPH::SurfaceTensionBase (C++ class), 167
SPH::SurfaceTensionBase::~SurfaceTensionBase (C++ function), 170
    (C++ function), 167
SPH::SurfaceTensionBase::initParameters (C++ function), 170
    (C++ function), 167
SPH::SurfaceTensionBase::m_surfaceTension (C++ member), 168
    (C++ member), 168
SPH::SurfaceTensionBase::m_surfaceTensionBoundaryStep (C++ member), 171
    (C++ member), 168
SPH::SurfaceTensionBase::SURFACE_TENSION (C++ member), 171
    (C++ member), 167
SPH::SurfaceTensionBase::SURFACE_TENSION_BOUNDARYStep::m_maxError (C++ member), 171
    (C++ member), 167
SPH::SurfaceTensionBase::SurfaceTensionBase (C++ function), 167
    (C++ function), 167
SPH::TimeIntegration (C++ class), 168
SPH::TimeIntegration::semiImplicitEuler (C++ function), 168
    (C++ function), 168
SPH::TimeIntegration::velocityUpdateFirstOrder (C++ function), 168
    (C++ function), 168
SPH::TimeIntegration::velocityUpdateSecondOrder (C++ function), 169
    (C++ function), 169
SPH::TimeManager (C++ class), 169
SPH::TimeManager::~TimeManager (C++ function), 169
SPH::TimeManager::getCurrent (C++ function), 169
SPH::TimeManager::getTime (C++ function), 169, 216
SPH::TimeManager::getTimeStepSize (C++ function), 169
SPH::TimeManager::hasCurrent (C++ function), 169
SPH::TimeManager::loadState (C++ function), 169
SPH::TimeManager::saveState (C++ function), 169
SPH::TimeManager::setCurrent (C++ function), 169
SPH::TimeManager:: setTime (C++ function), 169
SPH::TimeManager::setTimeStepSize (C++ function), 169
SPH::TimeManager::TimeManager (C++ function), 169
SPH::TimeStep (C++ class), 170
SPH::TimeStep::approximateNormal (C++ function), 170
    (C++ function), 166
SPH::TimeStep::clearAccelerations (C++ function), 170
    (C++ function), 166
SPH::TimeStep::computeDensities (C++ function), 170
    (C++ function), 166
SPH::TimeStep::computeDensityAndGradient (C++ function), 170
    (C++ function), 166
SPH::TimeStep::emittedParticles (C++ function), 170
    (C++ function), 167
SPH::TimeStep::init (C++ function), 170
    (C++ function), 168
SPH::TimeStep::initParameters (C++ function), 170
    (C++ function), 168
SPH::TimeStep::loadState (C++ function), 170
    (C++ function), 168
SPH::TimeStep::m_iterations (C++ member), 171
    (C++ member), 167
SPH::TimeStep::m_maxIterations (C++ member), 171
    (C++ member), 167
SPH::TimeStep::m_minIterations (C++ member), 171
    (C++ member), 167
SPH::TimeStep::MAX_ERROR (C++ member), 170
SPH::TimeStep::MAX_ITERATIONS (C++ member), 170
    (C++ function), 168
SPH::TimeStep::MIN_ITERATIONS (C++ member), 170
    (C++ function), 169
SPH::TimeStep::reset (C++ function), 170
SPH::TimeStep::resize (C++ function), 170
SPH::TimeStep::saveState (C++ function), 170
SPH::TimeStep::SOLVER_ITERATIONS (C++ member), 170
    (C++ function), 169
SPH::TimeStep::step (C++ function), 170
SPH::TimeStep::TimeStep (C++ function), 170
SPH::TimeStepDFSPH (C++ class), 171
SPH::TimeStepDFSPH::~TimeStepDFSPH (C++ function), 172
SPH::TimeStepDFSPH::computeDensityAdv (C++ function), 172
SPH::TimeStepDFSPH::computeDensityChange (C++ function), 172
SPH::TimeStepDFSPH::computeDFSPHFactor (C++ function), 172
SPH::TimeStepDFSPH::divergenceSolve (C++ function), 172
SPH::TimeStepDFSPH::divergenceSolveIteration (C++ function), 172
SPH::TimeStepDFSPH::emittedParticles (C++ function), 172

```

```

SPH::TimeStepDFSPH::initParameters (C++ function), 172
SPH::TimeStepDFSPH::m_counter (C++ member), 173
SPH::TimeStepDFSPH::m_enableDivergenceSo$PMr:TimeStepICSPH::LAMBDA (C++ member),
(C++ member), 173
SPH::TimeStepDFSPH::m_eps (C++ member), 173
SPH::TimeStepDFSPH::m_iterationsV (C++ member), 173
SPH::TimeStepDFSPH::m_maxErrorV (C++ member), 173
SPH::TimeStepDFSPH::m_maxIterationsV (C++ member), 173
SPH::TimeStepDFSPH::m_simulationData (C++ member), 173
SPH::TimeStepDFSPH::MAX_ERROR_V (C++ member), 172
SPH::TimeStepDFSPH::MAX_ITERATIONS_V (C++ member), 172
SPH::TimeStepDFSPH::performNeighborhoodSearch (C++ function), 172
SPH::TimeStepDFSPH::pressureSolve (C++ function), 172
SPH::TimeStepDFSPH::pressureSolveIteration (C++ function), 174
SPH::TimeStepDFSPH::pressureSolveIterati$SH:TimeStepICSPH::reset (C++ function),
(C++ function), 172
SPH::TimeStepDFSPH::reset (C++ function), 172
SPH::TimeStepDFSPH::resize (C++ function), 172
SPH::TimeStepDFSPH::SOLVER_ITERATIONS_V (C++ member), 172
SPH::TimeStepDFSPH::step (C++ function), 172
SPH::TimeStepDFSPH::TimeStepDFSPH (C++ function), 172
SPH::TimeStepDFSPH::USE_DIVERGENCE_SOLVER (C++ member), 172
SPH::TimeStepDFSPH::warmstartDivergenceSolve (C++ function), 172
SPH::TimeStepDFSPH::warmstartPressureSolve (C++ function), 172
SPH::TimeStepICSPH (C++ class), 173
SPH::TimeStepICSPH::~TimeStepICSPH (C++ function), 173
SPH::TimeStepICSPH::compute_aii (C++ function), 174
SPH::TimeStepICSPH::computeDensityAdv (C++ function), 174
SPH::TimeStepICSPH::computePressureAccel$PH:TimeStepIISPH::predictAdvection
(C++ function), 174
SPH::TimeStepICSPH::emittedParticles (C++ function), 174
SPH::TimeStepICSPH::getSimulationData (C++ function), 173
SPH::TimeStepICSPH::initParameters (C++ function), 174
SPH::TimeStepICSPH::integration (C++ function), 174
SPH::TimeStepICSPH::m_clamping (C++ member), 174
SPH::TimeStepICSPH::m_counter (C++ member), 174
SPH::TimeStepICSPH::m_lambda (C++ member), 174
SPH::TimeStepICSPH::m_psi (C++ member), 174
SPH::TimeStepICSPH::m_simulationData (C++ member), 174
SPH::TimeStepICSPH::performNeighborhoodSearch (C++ function), 174
SPH::TimeStepICSPH::PRESSURE_CLAMPING (C++ member), 174
SPH::TimeStepICSPH::pressureSolve (C++ function), 174
SPH::TimeStepICSPH::pressureSolveIteration (C++ function), 174
SPH::TimeStepICSPH::reset (C++ function), 173
SPH::TimeStepICSPH::resize (C++ function), 173
SPH::TimeStepICSPH::step (C++ function), 173
SPH::TimeStepICSPH::TimeStepICSPH (C++ function), 173
SPH::TimeStepIISPH (C++ class), 175
SPH::TimeStepIISPH::~TimeStepIISPH (C++ function), 175
SPH::TimeStepIISPH::computePressureAccels (C++ function), 175
SPH::TimeStepIISPH::emittedParticles (C++ function), 175
SPH::TimeStepIISPH::getSimulationData (C++ function), 175
SPH::TimeStepIISPH::integrate (C++ function), 175
SPH::TimeStepIISPH::m_counter (C++ member), 175
SPH::TimeStepIISPH::m_simulationData (C++ member), 175
SPH::TimeStepIISPH::performNeighborhoodSearch (C++ function), 175
SPH::TimeStepIISPH::predictAdvection (C++ function), 175
SPH::TimeStepIISPH::pressureSolve (C++ function), 175
SPH::TimeStepIISPH::pressureSolveIteration (C++ function), 175

```

SPH::TimeStepIISPH::reset (*C++ function*), 175  
SPH::TimeStepIISPH::resize (*C++ function*), 175  
SPH::TimeStepIISPH::step (*C++ function*), 175  
SPH::TimeStepIISPH::TimeStepIISPH (*C++ function*), 175  
SPH::TimeStepPBF (*C++ class*), 176  
SPH::TimeStepPBF::~TimeStepPBF (*C++ function*), 176  
SPH::TimeStepPBF::emittedParticles (*C++ function*), 177  
SPH::TimeStepPBF::ENUM\_PBF\_FIRST\_ORDER (*C++ member*), 176  
SPH::TimeStepPBF::ENUM\_PBF\_SECOND\_ORDER (*C++ member*), 176  
SPH::TimeStepPBF::initParameters (*C++ function*), 177  
SPH::TimeStepPBF::m\_counter (*C++ member*), 177  
SPH::TimeStepPBF::m\_simulationData (*C++ member*), 177  
SPH::TimeStepPBF::m\_velocityUpdateMethod (*C++ member*), 177  
SPH::TimeStepPBF::performNeighborhoodSearch (*C++ function*), 177  
SPH::TimeStepPBF::pressureSolve (*C++ function*), 177  
SPH::TimeStepPBF::pressureSolveIteration (*C++ function*), 177  
SPH::TimeStepPBF::reset (*C++ function*), 176  
SPH::TimeStepPBF::resize (*C++ function*), 176  
SPH::TimeStepPBF::step (*C++ function*), 176  
SPH::TimeStepPBF::TimeStepPBF (*C++ function*), 176  
SPH::TimeStepPBF::VELOCITY\_UPDATE\_METHODSPH::TimeStepPF::reset (*C++ function*), 178  
SPH::TimeStepPBF::resize (*C++ function*), 178  
SPH::TimeStepPBF::solvePDConstraints (*C++ function*), 179  
SPH::TimeStepPBF::Solver (*C++ type*), 179  
SPH::TimeStepPBF::step (*C++ function*), 178  
SPH::TimeStepPBF::STIFFNESS (*C++ member*), 179  
SPH::TimeStepPBF::TimeStepPF (*C++ function*), 178  
SPH::TimeStepPBF::updatePositionsAndVelocity (*C++ function*), 179  
SPH::TimeStepPBF::VectorXr (*C++ type*), 179, 235  
SPH::TimeStepPBF::VectorXrMap (*C++ type*), 179  
SPH::TimeStepWCSPH (*C++ class*), 180  
SPH::TimeStepWCSPH::~TimeStepWCSPH (*C++ function*), 180

```

SPH::TimeStepWCSPH::computePressureAccel SPH::TriangleMesh::m_x0 (C++ member), 182
    (C++ function), 181
SPH::TimeStepWCSPH::emittedParticles     SPH::TriangleMesh::Normals (C++ type), 181
    (C++ function), 181
SPH::TimeStepWCSPH::EXPONENT (C++ member), 180
SPH::TimeStepWCSPH::initParameters (C++ function), 181
SPH::TimeStepWCSPH::m_counter (C++ member), 181
SPH::TimeStepWCSPH::m_exponent (C++ member), 181
SPH::TimeStepWCSPH::m_simulationData   SPH::TriangleMesh::numFaces (C++ function),
    (C++ member), 181                                         182
SPH::TimeStepWCSPH::m_stiffness        SPH::TriangleMesh::numVertices (C++ function),
    (C++ member), 181                                         182
SPH::TimeStepWCSPH::performNeighborhoodSearch SPH::TriangleMesh::release (C++ function),
    (C++ function), 181
SPH::TimeStepWCSPH::reset   (C++ function), 180
SPH::TimeStepWCSPH::resize  (C++ function), 180
SPH::TimeStepWCSPH::step    (C++ function), 180
SPH::TimeStepWCSPH::STIFFNESS (C++ member), 180
SPH::TimeStepWCSPH::TimeStepWCSPH (C++ function), 180
SPH::TriangleMesh (C++ class), 181
SPH::TriangleMesh::~TriangleMesh   (C++ function), 181
SPH::TriangleMesh::addFace  (C++ function), 181
SPH::TriangleMesh::addVertex (C++ function), 181
SPH::TriangleMesh::Faces (C++ type), 181
SPH::TriangleMesh::getFaceNormals (C++ function), 182
SPH::TriangleMesh::getFaces   (C++ function), 181, 182
SPH::TriangleMesh::getVertexNormals (C++ function), 182
SPH::TriangleMesh::getVertices (C++ function), 182
SPH::TriangleMesh::getVertices0  (C++ function), 182
SPH::TriangleMesh::initMesh  (C++ function), 181
SPH::TriangleMesh::m_indices  (C++ member), 182
SPH::TriangleMesh::m_normals  (C++ member), 182
SPH::TriangleMesh::m_vertexNormals (C++ member), 182
SPH::TriangleMesh::m_x (C++ member), 182
SPH::TriangleMesh::Vertices (C++ type), 181
SPH::Viscosity_Bender2017 (C++ class), 183
SPH::Viscosity_Bender2017::~Viscosity_Bender2017
    (C++ function), 183
SPH::Viscosity_Bender2017::computeTargetStrainRate
    (C++ function), 183
SPH::Viscosity_Bender2017::computeViscosityFactor
    (C++ function), 183
SPH::Viscosity_Bender2017::creator (C++ function), 183
SPH::Viscosity_Bender2017::initParameters
    (C++ function), 184
SPH::Viscosity_Bender2017::ITERATIONS
    (C++ member), 184
SPH::Viscosity_Bender2017::m_iterations
    (C++ member), 184
SPH::Viscosity_Bender2017::m_maxError
    (C++ member), 184
SPH::Viscosity_Bender2017::m_maxIter
    (C++ member), 184
SPH::Viscosity_Bender2017::m_targetStrainRate
    (C++ member), 184
SPH::Viscosity_Bender2017::m_viscosityFactor
    (C++ member), 184
SPH::Viscosity_Bender2017::m_viscosityLambda
    (C++ member), 184
SPH::Viscosity_Bender2017::MAX_ERROR
    (C++ member), 184
SPH::Viscosity_Bender2017::MAX_ITERATIONS
    (C++ member), 184
SPH::Viscosity_Bender2017::performNeighborhoodSearch
    (C++ function), 183
SPH::Viscosity_Bender2017::reset   (C++ function), 183
SPH::Viscosity_Bender2017::step    (C++ function), 183
SPH::Viscosity_Bender2017::Viscosity_Bender2017
    (C++ function), 183

```

SPH::Viscosity\_Peer2015 (*C++ class*), 184  
SPH::Viscosity\_Peer2015::~Viscosity\_Peer2015 (*C++ function*), 185  
SPH::Viscosity\_Peer2015::computeDensities (*C++ function*), 185  
SPH::Viscosity\_Peer2015::creator (*C++ function*), 185  
SPH::Viscosity\_Peer2015::initParameters (*C++ function*), 185  
SPH::Viscosity\_Peer2015::ITERATIONS (*C++ member*), 185  
SPH::Viscosity\_Peer2015::m\_density (*C++ member*), 185  
SPH::Viscosity\_Peer2015::m\_iterations (*C++ member*), 185  
SPH::Viscosity\_Peer2015::m\_maxError (*C++ member*), 185  
SPH::Viscosity\_Peer2015::m\_maxIter (*C++ member*), 185  
SPH::Viscosity\_Peer2015::m\_solver (*C++ member*), 185  
SPH::Viscosity\_Peer2015::m\_targetNablaV (*C++ member*), 185  
SPH::Viscosity\_Peer2015::matrixVecProd (*C++ function*), 185  
SPH::Viscosity\_Peer2015::MAX\_ERROR (*C++ member*), 185  
SPH::Viscosity\_Peer2015::MAX\_ITERATIONS (*C++ member*), 185  
SPH::Viscosity\_Peer2015::performNeighborhoodSearch (*C++ function*), 185  
SPH::Viscosity\_Peer2015::reset (*C++ function*), 185  
SPH::Viscosity\_Peer2015::Solver (*C++ type*), 185  
SPH::Viscosity\_Peer2015::step (*C++ function*), 185  
SPH::Viscosity\_Peer2015::Viscosity\_Peer2015 (*C++ function*), 185  
SPH::Viscosity\_Peer2016 (*C++ class*), 186  
SPH::Viscosity\_Peer2016::~Viscosity\_Peer2016:Viscosity\_Peer2016::Viscosity\_Peer2016 (*C++ function*), 186  
SPH::Viscosity\_Peer2016::computeDensities (*C++ function*), 187  
SPH::Viscosity\_Peer2016::creator (*C++ function*), 186  
SPH::Viscosity\_Peer2016::initParameters (*C++ function*), 187  
SPH::Viscosity\_Peer2016::ITERATIONS\_OMEGA (*C++ member*), 187  
SPH::Viscosity\_Peer2016::ITERATIONS\_V (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_density (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_iterationsOmega (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_iterationsV (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_maxErrorOmega (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_maxErrorV (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_maxIterOmega (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_maxIterV (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_omega (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_solverOmega (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_solverV (*C++ member*), 187  
SPH::Viscosity\_Peer2016::m\_targetNablaV (*C++ member*), 187  
SPH::Viscosity\_Peer2016::matrixVecProdOmega (*C++ function*), 186  
SPH::Viscosity\_Peer2016::matrixVecProdV (*C++ function*), 186  
SPH::Viscosity\_Peer2016::MAX\_ERROR\_OMEGA (*C++ member*), 187  
SPH::Viscosity\_Peer2016::MAX\_ERROR\_V (*C++ member*), 187  
SPH::Viscosity\_Peer2016::MAX\_ITERATIONS\_OMEGA (*C++ member*), 187  
SPH::Viscosity\_Peer2016::MAX\_ITERATIONS\_V (*C++ member*), 187  
SPH::Viscosity\_Peer2016::performNeighborhoodSearch (*C++ function*), 186  
SPH::Viscosity\_Peer2016::reset (*C++ function*), 186  
SPH::Viscosity\_Peer2016::Solver (*C++ type*), 187  
SPH::Viscosity\_Peer2016::step (*C++ function*), 186  
SPH::Viscosity\_Standard (*C++ class*), 188  
SPH::Viscosity\_Standard::~Viscosity\_Standard (*C++ function*), 188  
SPH::Viscosity\_Standard::creator (*C++ function*), 188  
SPH::Viscosity\_Standard::initParameters (*C++ function*), 188  
SPH::Viscosity\_Standard::m\_boundaryViscosity (*C++ member*), 189  
SPH::Viscosity\_Standard::reset (*C++ function*), 188  
SPH::Viscosity\_Standard::step (*C++ function*), 188

```

tion), 188
SPH::Viscosity_Standard::VISCOSITY_COEFFICIENT_BOUNDARY, 192
(C++ member), 188
SPH::Viscosity_Standard::Viscosity_Standard (C++ member), 191
(C++ function), 188
SPH::Viscosity_Takahashi2015 (C++ class), (C++ member), 192
189
SPH::Viscosity_Takahashi2015::~Viscosity_Takahashi2015 (C++ member), 192
(C++ function), 189
SPH::Viscosity_Takahashi2015::computeViscosity (C++ member), 192
(C++ function), 190
SPH::Viscosity_Takahashi2015::creator (C++ member), 192
(C++ function), 190
SPH::Viscosity_Takahashi2015::initParameters (C++ member), 192
(C++ function), 190
SPH::Viscosity_Takahashi2015::ITERATIONS (C++ member), 192
(C++ member), 190
SPH::Viscosity_Takahashi2015::m_accel (C++ member), 190
SPH::Viscosity_Takahashi2015::m_iterations (C++ member), 190
SPH::Viscosity_Takahashi2015::m_maxError (C++ member), 190
SPH::Viscosity_Takahashi2015::m_maxIter (C++ member), 190
SPH::Viscosity_Takahashi2015::m_solver (C++ member), 190
SPH::Viscosity_Takahashi2015::reset (C++ function), 191
SPH::Viscosity_Takahashi2015::m_viscousStress (C++ member), 190
SPH::Viscosity_Takahashi2015::matrixVecProd (C++ function), 190
SPH::Viscosity_Takahashi2015::MAX_ERROR (C++ member), 190
SPH::Viscosity_Takahashi2015::MAX_ITERATIONS (C++ member), 191
SPH::Viscosity_Takahashi2015::performNeighborhoodSearch (C++ function), 191
SPH::Viscosity_Takahashi2015::Solver (C++ type), 192
SPH::Viscosity_Takahashi2015::step (C++ function), 191
SPH::Viscosity_Takahashi2015::Viscosity_XSPH (C++ class), 192
(C++ function), 189
SPH::Viscosity_XSPH (C++ class), 192
(C++ function), 193
SPH::Viscosity_XSPH::~Viscosity_XSPH (C++ function), 193
SPH::Viscosity_XSPH::reset (C++ function), 193
SPH::Viscosity_XSPH::step (C++ function), 193
SPH::Viscosity_XSPH::VISCOSITY_COEFFICIENT_BOUNDARY (C++ member), 193
SPH::Viscosity_XSPH::Viscosity_XSPH (C++ function), 193
SPH::Viscosity_XSPH::ViscosityBase (C++ class), 194

```

SPH::ViscosityBase::~ViscosityBase (*C++ function*), 194  
 SPH::ViscosityBase::initParameters (*C++ function*), 194  
 SPH::ViscosityBase::m\_viscosity (*C++ member*), 194  
 SPH::ViscosityBase::VISCOSITY\_COEFFICIENT (*C++ member*), 194  
 SPH::ViscosityBase::ViscosityBase (*C++ function*), 194  
 SPH::VorticityBase (*C++ class*), 195  
 SPH::VorticityBase::~VorticityBase (*C++ function*), 195  
 SPH::VorticityBase::initParameters (*C++ function*), 195  
 SPH::VorticityBase::m\_vorticityCoeff (*C++ member*), 195  
 SPH::VorticityBase::VORTICITY\_COEFFICIENT (*C++ member*), 195  
 SPH::VorticityBase::VorticityBase (*C++ function*), 195  
 SPH::VorticityConfinement (*C++ class*), 195  
 SPH::VorticityConfinement::~VorticityConfinement (*C++ function*), 196  
 SPH::VorticityConfinement::creator (*C++ function*), 196  
 SPH::VorticityConfinement::m\_normOmega (*C++ member*), 196  
 SPH::VorticityConfinement::m\_omega (*C++ member*), 196  
 SPH::VorticityConfinement::performNeighborhoodSearchSort (*C++ function*), 196  
 SPH::VorticityConfinement::reset (*C++ function*), 196  
 SPH::VorticityConfinement::step (*C++ function*), 196  
 SPH::VorticityConfinement::VorticityConfinement::AverageCount (*C++ struct*), 87  
 SPH::WendlandQuinticC2Kernel (*C++ class*), 196  
 SPH::WendlandQuinticC2Kernel2D (*C++ class*), 197  
 SPH::WendlandQuinticC2Kernel2D::getRadius (*C++ function*), 197  
 SPH::WendlandQuinticC2Kernel2D::gradW (*C++ function*), 197  
 SPH::WendlandQuinticC2Kernel2D::m\_k (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel2D::m\_l (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel2D::m\_radius (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel2D::m\_W\_zero (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel2D::setRadius (*C++ function*), 197  
 SPH::WendlandQuinticC2Kernel2D::W (*C++ function*), 197  
 SPH::WendlandQuinticC2Kernel2D::W\_zero (*C++ function*), 197  
 SPH::WendlandQuinticC2Kernel::getRadius (*C++ function*), 196  
 SPH::WendlandQuinticC2Kernel::gradW (*C++ function*), 196  
 SPH::WendlandQuinticC2Kernel::m\_k (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel::m\_l (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel::m\_radius (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel::m\_W\_zero (*C++ member*), 197  
 SPH::WendlandQuinticC2Kernel::setRadius (*C++ function*), 196  
 SPH::WendlandQuinticC2Kernel::W (*C++ function*), 196  
 SPH::WendlandQuinticC2Kernel::W\_zero (*C++ function*), 196  
 START\_TIMING (*C macro*), 229  
 STOP\_TIMING (*C macro*), 229  
 STOP\_TIMING\_AVG (*C macro*), 229  
 STOP\_TIMING\_AVG\_PRINT (*C macro*), 229  
 STOP\_TIMING\_PRINT (*C macro*), 230  
 SystemMatrixType (*C++ type*), 233  
 USE\_BLOCKDIAGONAL\_PRECONDITIONER (*C macro*), 230  
 USE\_WARMSTART (*C macro*), 230  
 USE\_WARMSTART\_V (*C macro*), 230  
 Utilities::AverageCount (*C++ struct*), 87  
 Utilities::AverageCount::numberOfCalls (*C++ member*), 87  
 Utilities::AverageCount::sum (*C++ member*), 87  
 Utilities::AverageTime (*C++ struct*), 87  
 Utilities::AverageTime::counter (*C++ member*), 87  
 Utilities::AverageTime::name (*C++ member*), 87  
 Utilities::AverageTime::totalTime (*C++ member*), 87  
 Utilities::ConsoleSink (*C++ class*), 198  
 Utilities::ConsoleSink::ConsoleSink (*C++ function*), 198  
 Utilities::ConsoleSink::write (*C++ function*), 198  
 Utilities::Counting (*C++ class*), 198

```

Utilities::Counting::m_averageCounts           200
    (C++ member), 198
Utilities::Counting::reset (C++ function),   200
    198
Utilities::FileSink (C++ class), 199
Utilities::FileSink::~FileSink (C++ function), 199
Utilities::FileSink::FileSink (C++ function), 199
Utilities::FileSink::m_file (C++ member),     201
    199
Utilities::FileSink::write (C++ function),    200
    199
Utilities::FileSystem (C++ class), 199
Utilities::FileSystem::checkMD5   (C++ function), 200
Utilities::FileSystem::copyFile   (C++ function), 199
Utilities::FileSystem::fileExists (C++ function), 199
Utilities::FileSystem::getFileExt (C++ function), 199
Utilities::FileSystem::getFileName (C++ function), 199
Utilities::FileSystem::getFileNameWithEx  Utilities::LogSink (C++ class), 201
    (C++ function), 199
Utilities::FileSystem::getFilePath (C++ function), 199
Utilities::FileSystem::getFilesInDirectory (C++ function), 200
Utilities::FileSystem::getProgramPath   (C++ function), 199
Utilities::FileSystem::isDirectory (C++ function), 200
Utilities::FileSystem::isFile   (C++ function), 200
Utilities::FileSystem::isRelativePath (C++ function), 199
Utilities::FileSystem::makeDir   (C++ function), 199
Utilities::FileSystem::makeDirs   (C++ function), 199
Utilities::FileSystem::normalizePath (C++ function), 199
Utilities::FileSystem::writeMD5File   (C++ function), 200
Utilities::IDFactory (C++ class), 200
Utilities::IDFactory::getId (C++ function), 200
Utilities::Logger (C++ class), 200
Utilities::logger (C++ member), 222
Utilities::Logger::~Logger (C++ function), 202
Utilities::Logger::activate (C++ function), 200
Utilities::Logger::addSink (C++ function), 200
Utilities::Logger::Logger (C++ function), 200
Utilities::Logger::m_active (C++ member), 201
Utilities::Logger::m_sinks (C++ member), 201
Utilities::Logger::write (C++ function), 200
Utilities::LogLevel (C++ enum), 213
Utilities::LogLevel::DEBUG (C++ enumerator), 213
Utilities::LogLevel::ERR (C++ enumerator), 213
Utilities::LogLevel::INFO (C++ enumerator), 213
Utilities::LogLevel::WARN (C++ enumerator), 213
Utilities::LogSink (C++ class), 201
Utilities::LogSink::~LogSink (C++ function), 201
Utilities::LogSink::LogSink (C++ function), 201
Utilities::LogSink::m_minLevel (C++ member), 201
Utilities::LogSink::write (C++ function), 201
Utilities::LogStream (C++ class), 201
Utilities::LogStream::~LogStream (C++ function), 202
Utilities::LogStream::LogStream (C++ function), 202
Utilities::LogStream::m_buffer (C++ member), 202
Utilities::LogStream::m_level (C++ member), 202
Utilities::LogStream::m_logger (C++ member), 202
Utilities::LogStream::operator<<  (C++ function), 202
Utilities::MeshFaceIndices (C++ struct), 88
Utilities::MeshFaceIndices::normalIndices (C++ member), 88
Utilities::MeshFaceIndices::posIndices (C++ member), 88
Utilities::MeshFaceIndices::texIndices (C++ member), 88
Utilities::OBJLoader (C++ class), 202
Utilities::OBJLoader::loadObj (C++ function), 202
Utilities::OBJLoader::Vec2f (C++ type), 202

```

Utilities::OBJLoader::Vec3f (*C++ type*), 202 Utilities::SceneLoader::BoundaryData::samplingMode  
Utilities::PartioReaderWriter (*C++ class*), (*C++ member*), 89, 205  
203 Utilities::SceneLoader::BoundaryData::scale  
Utilities::PartioReaderWriter::readParticles (*C++ member*), 89, 205  
(*C++ function*), 203 Utilities::SceneLoader::BoundaryData::translation  
Utilities::PartioReaderWriter::writeParticles (*C++ member*), 89, 205  
(*C++ function*), 203 Utilities::SceneLoader::Box (*C++ struct*),  
Utilities::SceneLoader (*C++ class*), 203 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::Box::m\_maxX  
(*C++ struct*), 88, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::Box::m\_minX  
(*C++ member*), 88, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::EmitterData  
(*C++ member*), 88, 204 (*C++ struct*), 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::EmitterData::emitEndTime  
(*C++ member*), 88, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::EmitterData::emitStartTime  
(*C++ member*), 88, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::EmitterData::height  
(*C++ member*), 88, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::EmitterData::id  
(*C++ member*), 88, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::AnimationFieldData Utilities::SceneLoader::EmitterData::rotation  
(*C++ member*), 88, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::BoundaryData Utilities::SceneLoader::EmitterData::velocity  
(*C++ struct*), 89, 204 (*C++ member*), 90, 205  
Utilities::SceneLoader::BoundaryData::capabilities Utilities::SceneLoader::EmitterData::width  
(*C++ member*), 89, 205 (*C++ member*), 90, 205  
Utilities::SceneLoader::BoundaryData::density Utilities::SceneLoader::EmitterData::x  
(*C++ member*), 89, 205 (*C++ member*), 90, 205  
Utilities::SceneLoader::BoundaryData::dy Utilities::SceneLoader::FluidBlock (*C++ struct*),  
91, 205  
Utilities::SceneLoader::BoundaryData::isAnimatable Utilities::SceneLoader::FluidBlock::box  
(*C++ member*), 89, 205 (*C++ member*), 91, 206  
Utilities::SceneLoader::BoundaryData::isWet Utilities::SceneLoader::FluidBlock::id  
(*C++ member*), 89, 205 (*C++ member*), 91, 206  
Utilities::SceneLoader::BoundaryData::mapFile Utilities::SceneLoader::FluidBlock::initialAngularVelocity  
(*C++ member*), 89, 205 (*C++ member*), 91, 206  
Utilities::SceneLoader::BoundaryData::mapFile Utilities::SceneLoader::FluidBlock::initialVelocity  
(*C++ member*), 89, 205 (*C++ member*), 91, 206  
Utilities::SceneLoader::BoundaryData::mapFile Utilities::SceneLoader::FluidBlock::mode  
(*C++ member*), 89, 205 (*C++ member*), 91, 206  
Utilities::SceneLoader::BoundaryData::mapFile Utilities::SceneLoader::FluidBlock::visMeshFile  
(*C++ member*), 89, 205 (*C++ member*), 91, 206  
Utilities::SceneLoader::BoundaryData::mesh Utilities::SceneLoader::FluidData (*C++ struct*),  
91, 206  
Utilities::SceneLoader::BoundaryData::rigidBody Utilities::SceneLoader::FluidData::id  
(*C++ member*), 89, 205 (*C++ member*), 91, 206  
Utilities::SceneLoader::BoundaryData::roughness Utilities::SceneLoader::FluidData::initialAngularVelocity  
(*C++ member*), 89, 205 (*C++ member*), 92, 206  
Utilities::SceneLoader::BoundaryData::sampleSize Utilities::SceneLoader::FluidData::initialVelocity  
(*C++ member*), 89, 205 (*C++ member*), 91, 206

```

Utilities::SceneLoader::FluidData::invertUtilities::SceneLoader::Scene::boundaryModels
    (C++ member), 92, 206                               (C++ member), 93, 207
Utilities::SceneLoader::FluidData::mode Utilities::SceneLoader::Scene::camLookat
    (C++ member), 92, 206                               (C++ member), 93, 207
Utilities::SceneLoader::FluidData::resolutionUtilities::SceneLoader::Scene::camPosition
    (C++ member), 92, 206                               (C++ member), 93, 207
Utilities::SceneLoader::FluidData::rotationUtilities::SceneLoader::Scene::emitters
    (C++ member), 91, 206                               (C++ member), 93, 207
Utilities::SceneLoader::FluidData::samplingUtilities::SceneLoader::Scene::fluidBlocks
    (C++ member), 91, 206                               (C++ member), 93, 207
Utilities::SceneLoader::FluidData::scaleUtilities::SceneLoader::Scene::fluidModels
    (C++ member), 91, 206                               (C++ member), 93, 207
Utilities::SceneLoader::FluidData::transformationUtilities::SceneLoader::Scene::materials
    (C++ member), 91, 206                               (C++ member), 93, 207
Utilities::SceneLoader::FluidData::visMeshUtilities::SceneLoader::Scene::particleRadius
    (C++ member), 91, 206                               (C++ member), 93, 207
Utilities::SceneLoader::hasValue (C++ function), 204 Utilities::SceneLoader::Scene::sim2D
    (C++ member), 93, 207
Utilities::SceneLoader::m_jsonData (C++ member), 204 Utilities::SceneLoader::Scene::timeStepSize
    (C++ class), 207
Utilities::SceneLoader::MaterialData (C++ struct), 92, 206 Utilities::SDFFunctions (C++ class), 207
Utilities::SceneLoader::MaterialData::colorField (C++ function), 207 Utilities::SDFFunctions::computeBoundingBox
    (C++ member), 92, 206                               Utilities::SDFFunctions::distance (C++ function)
Utilities::SceneLoader::MaterialData::colorMap (C++ function), 207 Utilities::SDFFunctions::generateSDF
    (C++ member), 92, 206                               Utilities::SDFFunctions::generateSDF
Utilities::SceneLoader::MaterialData::emitterBox (C++ function), 207 Utilities::StringTools (C++ class), 208
Utilities::SceneLoader::MaterialData::emitSphere (C++ function), 207 Utilities::StringTools::real2String
    (C++ member), 92, 206                               (C++ function), 208
Utilities::SceneLoader::MaterialData::emitSphere (C++ function), 207 Utilities::StringTools::to_upper (C++ function)
    (C++ member), 92, 206                               (C++ function), 208
Utilities::SceneLoader::MaterialData::idUtilities::StringTools::tokenize (C++ function)
    (C++ member), 92, 206                               (C++ function), 208
Utilities::SceneLoader::MaterialData::maxEmissivityBarSystemInfo (C++ class), 208 Utilities::SystemInfo::getHostName (C++ member)
    (C++ member), 92, 206
Utilities::SceneLoader::MaterialData::maxVal (C++ function), 208 Utilities::Timing (C++ class), 209
    (C++ member), 92, 206
Utilities::SceneLoader::MaterialData::minVelocities::Timing::m_averageTimes (C++ member)
    (C++ member), 92, 206                               (C++ member), 209
Utilities::SceneLoader::readMaterialParameters::Timing::m_dontPrintTimes
    (C++ function), 204                               (C++ member), 209
Utilities::SceneLoader::readParameterObjectUtilities::Timing::m_startCounter (C++ member)
    (C++ function), 204                               (C++ member), 209
Utilities::SceneLoader::readScene (C++ function), 204 Utilities::Timing::m_stopCounter (C++ member)
    (C++ member), 209
Utilities::SceneLoader::readValue (C++ function), 204 Utilities::Timing::m_timingStack (C++ member)
    (C++ member), 209
Utilities::SceneLoader::readVector (C++ function), 204 Utilities::Timing::reset (C++ function), 209
    (C++ member), 93, 207
Utilities::SceneLoader::Scene (C++ struct), 93, 207 Utilities::TimingHelper (C++ struct), 93
Utilities::SceneLoader::Scene::animatedFields::TimingHelper::start (C++ member)
    (C++ member), 93, 207                               (C++ member), 93

```

Utilities::VolumeSampling (*C++ class*), 209  
Utilities::VolumeSampling::sampleMesh  
    (*C++ function*), 209  
Utilities::WindingNumbers (*C++ class*), 210  
Utilities::WindingNumbers::computeGeneralizedWindingNumber  
    (*C++ function*), 210

## V

Vector2i (*C++ type*), 234  
Vector2r (*C++ type*), 234  
Vector3f (*C++ type*), 234  
Vector3f8 (*C++ class*), 210  
Vector3f8::blend (*C++ function*), 211  
Vector3f8::cross (*C++ function*), 211  
Vector3f8::dot (*C++ function*), 211  
Vector3f8::norm (*C++ function*), 211  
Vector3f8::normalize (*C++ function*), 211  
Vector3f8::operator\* (*C++ function*), 211  
Vector3f8::operator\*= (*C++ function*), 211  
Vector3f8::operator/ (*C++ function*), 211  
Vector3f8::operator/= (*C++ function*), 211  
Vector3f8::operator% (*C++ function*), 211  
Vector3f8::operator- (*C++ function*), 211  
Vector3f8::operator[] (*C++ function*), 211  
Vector3f8::reduce (*C++ function*), 211  
Vector3f8::setZero (*C++ function*), 211  
Vector3f8::squaredNorm (*C++ function*), 211  
Vector3f8::store (*C++ function*), 211  
Vector3f8::v (*C++ member*), 211  
Vector3f8::Vector3f8 (*C++ function*), 210  
Vector3f8::x (*C++ function*), 211  
Vector3f8::y (*C++ function*), 211  
Vector3f8::z (*C++ function*), 211  
Vector3r (*C++ type*), 234  
Vector4f (*C++ type*), 234  
Vector4r (*C++ type*), 235  
Vector5r (*C++ type*), 235  
Vector6r (*C++ type*), 235