User Guide for DualSPHysics code

DualSPHysics_v3.0
December 2013

dualsphysics@gmail.com
A.J.C. Crespo (alexbexe@uvigo.es)
J.M. Domínguez (jmdominguez@uvigo.es)
M.G. Gesteira (mggesteira@uvigo.es)
A. Barreiro (anxo.barreiro@uvigo.es)
B.D. Rogers (benedict.rogers@manchester.ac.uk)
S. Longshaw (Stephen.Longshaw@manchester.ac.uk)
R. Canelas (ricardo.canelas@ist.utl.pt)
R. Vacondio (renato.vacondio@unipr.it)

Universidade Vigo

MANCHESTER 1824
The University of Manchester

Acknowledgements

The development of DualSPHysics was partially supported by:

- Xunta de Galicia under project Programa de Consolidación e Estructuración de Unidades de Investigación Competitivas (Grupos de Referencia Competitiva) and also financed by European Regional Development Fund (FEDER).
- Ministerio de Economía y Competitividad under the project BIA2012-38676-C03-03
- Research Councils UK (RCUK) Research Fellowship.
- EPSRC Grants EP/H003045/1, EP/J0101235/1, Knowledge Transfer Account (University of Manchester)
- We want to acknowledge Orlando Garcia Feal for his technical help.

Abstract

This guide documents the DualSPHysics code based on the Smoothed Particle Hydrodynamics model named SPHysics. This manuscript describes how to compile and run the DualSPHysics code (a set of C++ and CUDA files). New pre-processing tools are implemented to create more complex geometries and new post-processing tools are developed to analyse easily numerical results. Several working examples are documented to enable the user to use the codes and understand how they work.

Contents

1. Introduction

Smoothed Particle Hydrodynamics is a Lagrangian meshless method that has been used in an expanding range of applications within the field of Computation Fluid Dynamics [Gómez-Gesteira et al., 2010a] where particles represent the flow, interact with structures, and exhibit large deformation with moving boundaries. The SPH model is approaching a mature stage with continuing improvements and modifications such that the accuracy, stability and reliability of the model are reaching an acceptable level for practical engineering applications.

SPHysics is an open-source SPH model developed by researchers at the Johns Hopkins University (US), the University of Vigo (Spain), the University of Manchester (UK) and the University of Rome, La Sapienza. The software is available to free download at www.sphysics.org. A complete guide of the FORTRAN code is found in [Gómez-Gesteira et al., 2012a, 2012b].

The SPHysics Fortran code was validated for different problems of wave breaking [Dalrymple and Rogers, 2006], dam-break behaviour [Crespo et al., 2008], interaction with coastal structures [Gómez-Gesteira and Dalrymple, 2004] or with a moving breakwater [Rogers et al., 2010].

Although SPHysics allows us to model problems using high resolution, the main problem for the application to real engineering problems is the long computational runtime, meaning that SPHysics is rarely applied to large domains. Hardware acceleration and parallel computing are required to make SPHysics more useful and versatile.

Graphics Processing Units (GPUs) appear as a cheap alternative to handle High Performance Computing (HPC) for numerical modelling. GPUs are designed to manage huge amounts of data and their computing power has developed in recent years much faster than conventional central processing units (CPUs). Compute Unified Device Architecture (CUDA) is a parallel programming framework and language for GPU computing using some extensions to the C/C++ language. Researchers and engineers of different fields are achieving high speedups implementing their codes with the CUDA language. Thus, the parallel power computing of GPUs can be also applied for SPH methods where the same loops for each particle during the simulation can be parallelised.

The code DualSPHysics has been developed starting from the SPH formulation implemented in SPHysics. This FORTRAN code is robust and reliable but is not properly optimised for huge simulations. DualSPHysics is implemented in C++ and CUDA language to carry out simulations on the CPU and GPU respectively. The new

CPU code presents some advantages, such as more optimised use of the memory. The object-oriented programming paradigm provides means to develop a code that is easy to understand, maintain and modify with a sophisticated control of errors available. Furthermore, better approaches are implemented, for example particles are reordered to give faster access to memory, symmetry is considered in the force computation to reduce the number of particle interactions and the best approach to create the neighbour list is implemented [Domínguez et al., 2011]. The CUDA language manages the parallel execution of threads on the GPUs. The best approaches were considered to be implemented as an extension of the C++ code, so the most appropriate optimizations to parallelise particle interaction on GPU were implemented [Domínguez et al., 2013a]. The first rigorous validations were presented in Crespo et al., 2011.

The DualSPHysics code has been developed to simulate real-life engineering problems using SPH models such as the computation of forces exerted by large waves on the urban furniture of a realistic promenade [Barreiro et al., 2013] or the study of the run-up in an existing armour block sea breakwater [Altomare et al., 2014].

In the following sections we will describe the SPH formulation available in DualSPHysics, the implementation and optimization techniques, how to compile and run the different codes of the DualSPHysics package and future developments.

2. SPH formulation

All the SPH theory implemented in DualSPHysics is taken directly from the SPHysics code. Therefore a more detailed description, equations and references are collected in [Gómez-Gesteira et al., 2012a, 2012b].

Here we only summarise the **SPH formulation already available** on the new DualSPHysics code:

- Time integration scheme:
  - Verlet [Verlet, 1967].
  - Symplectic [Leimkhuler, 1996].
- Variable time step [Monaghan and Kos, 1999].
- Kernel functions:
  - Cubic Spline kernel [Monaghan and Lattanzio, 1985].
  - Quintic Wendland kernel [Wendland, 1995].
- Density filter:
  - Shepard filter [Panizzo, 2004].
  - Delta-SPH formulation [Molteni and Colagrossi, 2009].
- Viscosity treatments:
  - Artificial viscosity [Monaghan, 1992].
  - Laminar viscosity + SPS turbulence model [Dalrymple and Rogers, 2006].
- Weakly compressible approach using Tait's equation of state.
- Dynamic boundary conditions [Crespo et al., 2007].
- Floating objects [Monaghan et al., 2003].
- Periodic open boundaries

Features that will be integrated soon on the CPU-GPU solver as **future improvements**:

- Multi-GPU implementation [Domínguez et al., 2013b].
- Double precision [Domínguez et al., 2013c].
- SPH-ALE with Riemann Solver.
- Primitive-variable Riemann Solver.
- Variable particle resolution [Vacondio et al., 2013].
- Multiphase (gas-soil-water) [Fourtakas et al., 2013a, Mokos et al., 2013].
- Inlet/outlet flow conditions.
- Extensions to the Modified virtual boundary conditions [Fourtakas et al., 2013b].
- Coupling with Discrete Element Method [Canelas et al., 2013].
- Coupling with Mass Point Lattice Spring Model [Longshaw et al., 2013].
- Coupling with SWASH Wave Propagation Model [Altomare et al., 2013].

3. CPU and GPU implementation

Detailed information about the CPU and GPU implementation can be found in the papers:

Domínguez JM, Crespo AJC, Gómez-Gesteira M, Marongiu, JC. 2011. Neighbour lists in Smoothed Particle Hydrodynamics. International Journal For Numerical Methods in Fluids, 67(12): 2026-2042. doi: 10.1002/fld.2481.

Crespo AJC, Domínguez JM, Barreiro A, Gómez-Gesteira M and Rogers BD. 2011. GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods. PLoS ONE, 6(6), e20685. doi:10.1371/journal.pone.0020685.

Valdez-Balderas D, Domínguez JM, Rogers BD, Crespo AJC. 2012. Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters. Journal of Parallel and Distributed Computing. doi:10.1016/j.jpdc.2012.07.010

Domínguez JM, Crespo AJC and Gómez-Gesteira M. 2013. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. Computer Physics Communications, 184(3): 617-627. doi:10.1016/j.cpc.2012.10.015

Domínguez JM, Crespo AJC, Valdez-Balderas D, Rogers BD. and Gómez-Gesteira M. 2013. New multi-GPU implementation for Smoothed Particle Hydrodynamics on heterogeneous clusters. Computer Physics Communications, 184: 1848-1860. doi: 10.1016/j.cpc.2013.03.008

The DualSPHysics code is the result of an optimised implementation using the best approaches for CPU and GPU with the accuracy, robustness and reliability shown by the SPHysics code. SPH simulations such as those in the SPHysics and DualSPHysics codes can be split in three main steps; (i) generation of the neighbour list, (ii) computation of the forces between particles (solving momentum and continuity equations) and (iii) the update of the physical quantities at the next time step. Thus, running a simulation means executing these steps in an iterative manner:

1st STEP: Neighbour list (Cell-linked list described in Domínguez et al., 2011):

- Domain is divided in square cells of side 2*h* (or the size of the kernel domain).
- Only a list of particles, ordered according to the cell to which they belong, is generated.
- All the arrays with the physical variables of the particles are reordered according the list of particles.

2nd STEP: Force computation:

- Particles of the same cell and adjacent cells are candidates to be neighbours.
- Each particle interacts with all its neighbouring particles (at a distance < 2*h*).

3rd STEP: System Update:

- New time step is computed.
- Physical quantities are updated in the next step starting from the values of physical variables at the present time step, the interaction forces and the new time step value.
- Particle information (velocity and density) are saved on local storage (the hard drive) at defined times.

A GPU implementation is initially focused on the force computation since following Domínguez et al., 2011, this is the most consuming part in terms of runtime. However the most efficient technique consists of minimising the communications between the CPU and GPU for the data transfers. If neighbour list and system update are also implemented on the GPU the CPU-GPU memory transfer is needed at the beginning of the simulation while relevant data will be transferred to the CPU when saving output data is required (usually infrequently). Crespo et al., 2011 used an execution of DualSPHysics performed entirely on the GPU to run a numerical experiment where the results are in close agreement with the experimental results.

The GPU implementation presents some key differences in comparison to the CPU version. The main difference is the parallel execution of all tasks that can be parallelised such as all loops regarding particles. One GPU execution thread computes the resulting force of one particle performing all the interactions with its neighbours. The symmetry of the particle interaction is employed on the CPU to reduce the runtime, but is not applied in the GPU implementation since it is not efficient due to memory coalescence issues.

DualSPHysics is unique where the same application can be run using either the CPU or GPU implementation; this facilitates the use of the code not only on workstations with a Nvidia GPU but also on machines without a CUDA-enabled GPU. The CPU version is parallelized using the OpenMP API. The main code has a common core for both the CPU and GPU implementations with only minor source code differences implemented for the two devices applying the specific optimizations for CPU and GPU. Thus, debugging or maintenance is easier and comparisons of results and computational time are more direct. Figure 3-1 shows a flow diagram to represent the differences between the CPU and GPU implementations and the different steps involved in a complete execution.

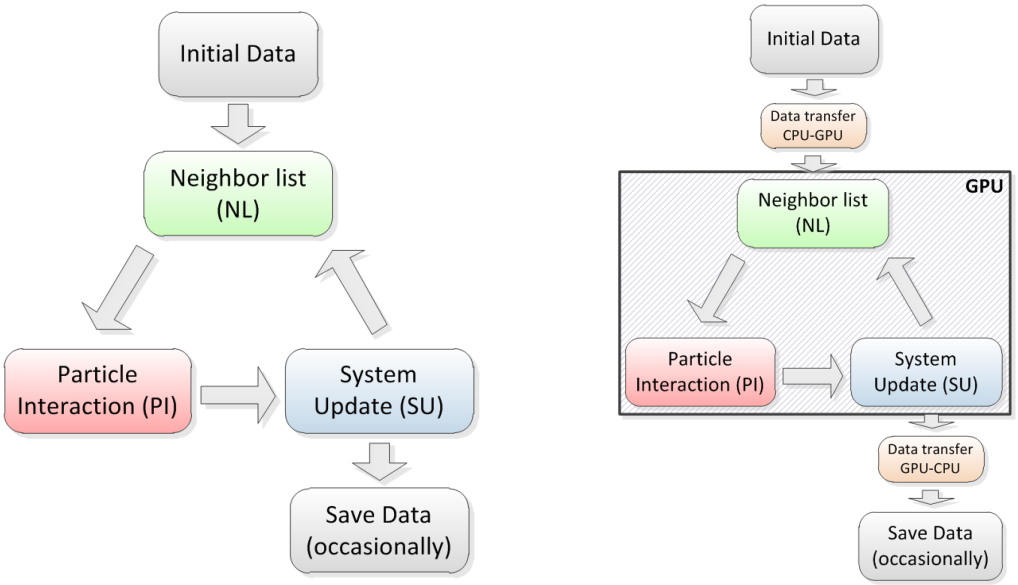


Figure 3-1. Flow diagram of the CPU (left) and total GPU implementation (right).

4. Running DualSPHysics

The user can download from http://dual.sphysics.org/index.php/downloads/ the following files:

- **DUALSPHYSICS DOCUMENTATION:**
  - **DualSPHysics_v3.0_GUIDE.pdf**
  - **ExternalModelsConversion_GUIDE.pdf**
  - **GenCase_XML_GUIDE.pdf**

- **DUALSPHYSICS PACKAGE:**
  - **DualSPHysics_v3.0_Linux_x64.zip**
  - **DualSPHysics_v3.0_Windows_x64t.zip**

- **DUALSPHYSICS PRE-PROCESSING INTERFACE:**
  - **DPI_v1.1_Linux_x64.zip**
  - **DPI_v1.1_Windows_x64.zip**

To start using DualSPHysics, users must follow these instructions:

1) First, download and read the **DUALSPHYSICS DOCUMENTATION**:
   - *DualSPHysics_v3.0_GUIDE*. This manuscript.
   - *GenCase_XML_GUIDE*: Helps to create a new case using the input XML file.
   - *ExternalModelsConversion_GUIDE:* Describes how to convert the file format of any external geometry of a 3D model to VTK, PLY or STL using open-source codes.

2) Download the **DUALSPHYSICS PACKAGE** (See Figure 4-1):
   - SOURCE
   - EXECS
   - HELP
   - MOTION
   - RUN_DIRECTORY:
     - CASEDAMBREAK
     - CASEFLOATING
     - CASEFORCES
     - CASEPERIODICITY
     - CASEPUMP
     - CASEWAVEMAKER

Figure 4-1 shows the structure of the content of **DUALSPHYSICS_PACKAGE**.

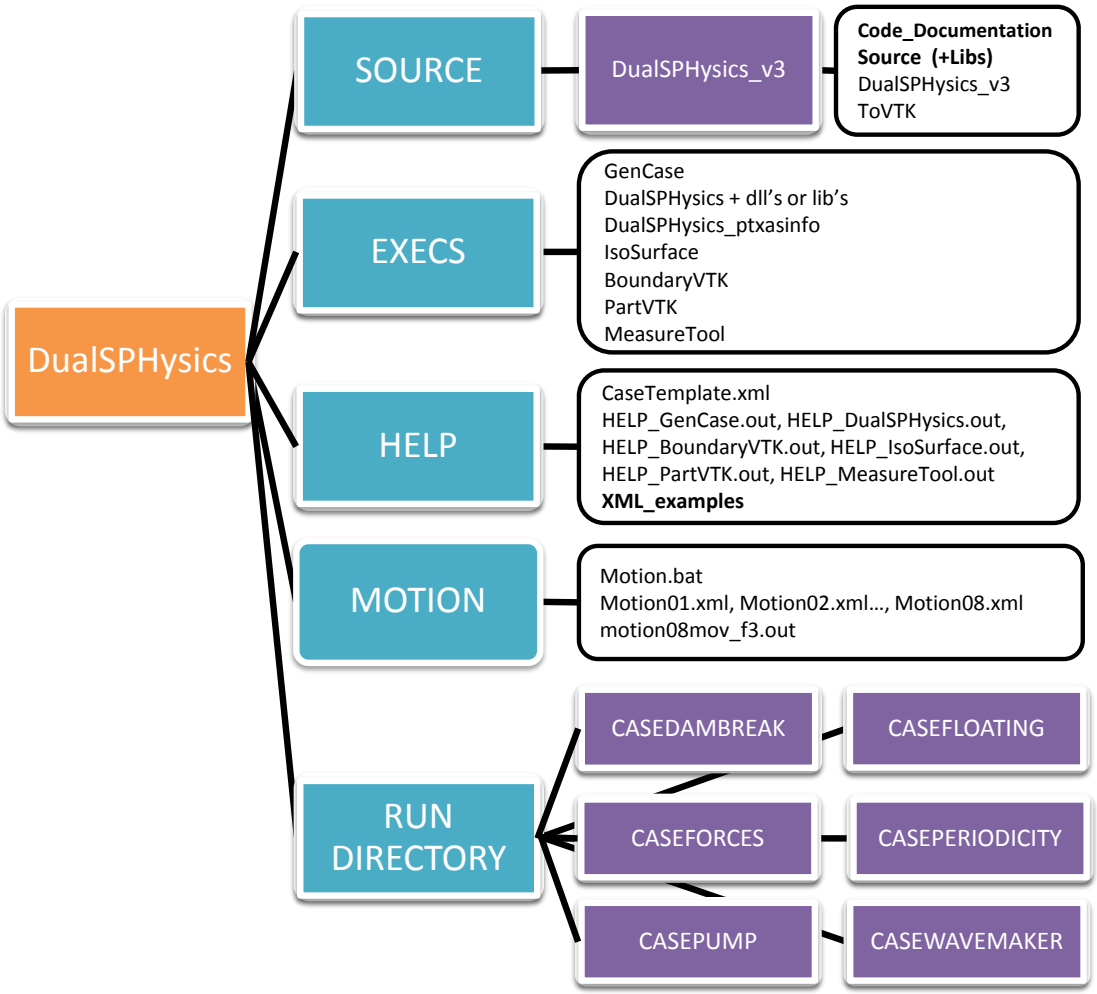


Figure 4-1. Directory tree and provided files.

3) The appropriate scripts (*.bat* in Windows and *.sh* in Linux) in the directories CASES must be used depending on a CPU or GPU execution.

4) Download the **DUALSPHYSICS PRE-PROCESSING INTERFACE.**
DualSPHysics Pre-processing Interface (DPI) software is a Java based application designed to allow the most common parameters needed to define a DualSPHysics case to be set in a more user-friendly manner. For cases involving more complex functionality (such as motion), DSI can be used to first define geometrical properties and the resulting XML file manually edited. More information can be found in the separate more detailed guide about DPI and in section http://dual.sphysics.org/index.php/dualsphysics-project/pre-processing/.

5) Paraview open-source software (www.paraview.org) is recommended to be used to visualise the results.

SOURCE:

- Contains the source files.
- Visual studio project for windows.
- Makefiles for linux.
- Code documentation using *Doxygen*.

EXECS:

- Contains the binaries.
- Some libraries needed for the codes are also included.
- The file *DualSPHysics_ptxasinfo* is used to optimise the block size for the different CUDA kernels on GPU executions.

HELP:

- Contains *CaseTemplate.xml*, a XML example with all the different labels and formats that can be used in the input XML file.
- The folder *XML_examples* includes more examples of the XML files used to run some of the simulations shown in www.youtube.com/dualsphysics.
- A description of the execution parameters of the different codes is presented in *HELP_NameCode.out*.

MOTION:

- Contains the script *Motion.bat (Motion.sh)* to perform the examples with the different type of movements that can be described with DualSPHysics. Eight examples can be carried out (*Motion01.xml…, Motion08.xml*).
- The text file motion08mov_f3.out describes the prescribed motion used in the eighth example.

RUN_DIRECTORY:

- It is the directory created to run the test cases. The directory includes input files and batch files to execute the cases and. The output files will be created in the same directory.
    - CASEDAMBREAK
    - CASEFLOATING
    - CASEFORCES
    - CASEPERIODICITY
    - CASEPUMP
    - CASEWAVEMAKER

Figure 4-2 briefs the input and output files of the executable files. This figure will be used later to explain in detail each of the codes and tools.

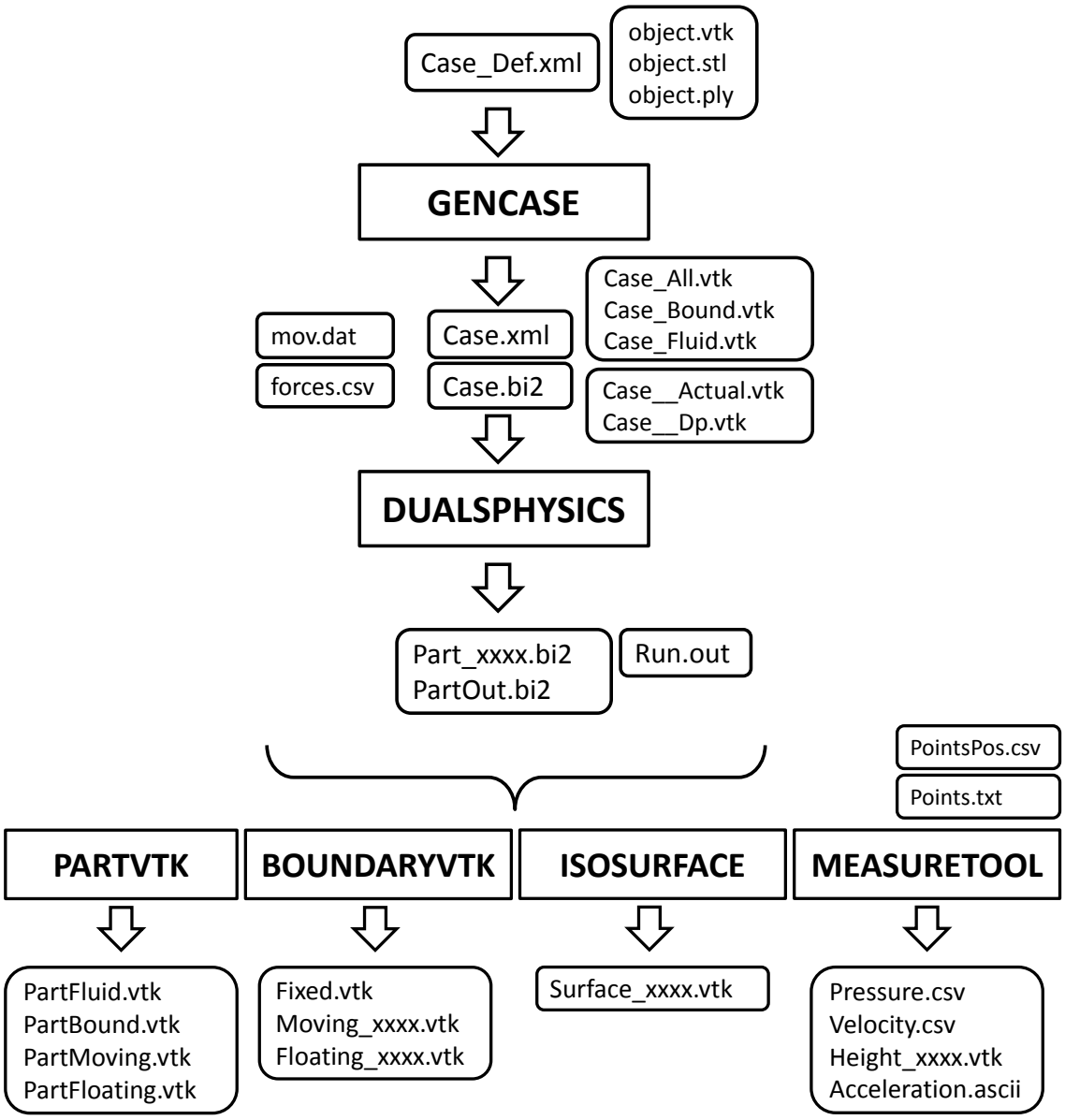


Figure 4-2. Protocol of a simulation.

5. DualSPHysics open-source code

This section reports a brief description of the source files of DualSPHysics v3.0. The source code is freely redistributable under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation (www.gnu.org/licenses/). Thus, users can download the files from
DUALSPHYSICS/SOURCE/DualSPHysics_3/Source.

First, note that a more complete documentation is provided in directory **Code_documentation**. This documentation has been created using the documentation system Doxygen (www.doxygen.org).
To navigate through the full documentation open the HTML file *index.html* as it is shown in Figure 5-1.

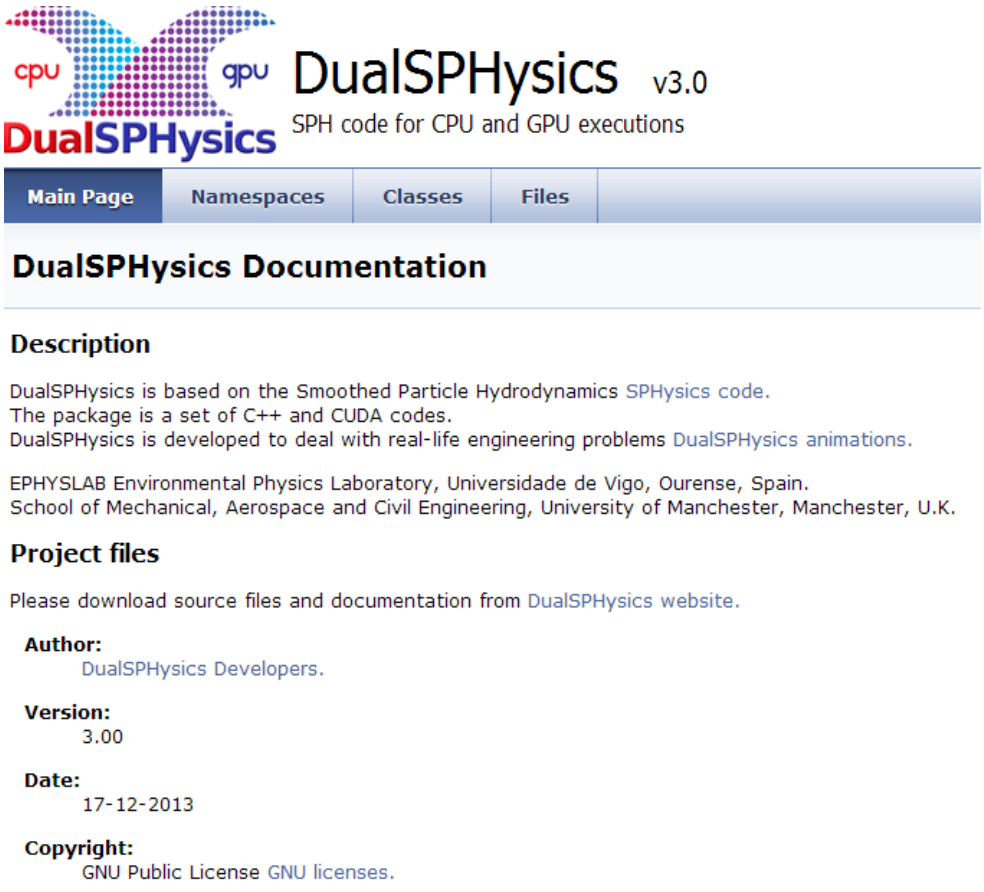


Figure 5-1. Documentation for DualSPHysics code generated with Doxygen.

Open source files are in **DUALSPHYSICS/SOURCE/DualSPHysics_3/Source**. A complete list of these source files is summarised in Table 5-1. Some files are shared with other codes such as GenCase, BoundaryVTK, PartVTK, MeasureTool and IsoSurface. The rest of the files implement the SPH solver, some of them are used both for CPU/GPU executions and others are specific.

Table 5-1. List of source files of DualSPHysics code.

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp) JException (.h .cpp) JFloatingData (.h .cpp) JLog2 (.h .cpp) JObject (.h .cpp) JObjectGpu (.h .cpp) JPartData (.h .cpp) JPtxasInfo (.h .cpp) JSpaceCtes (.h .cpp) JSpaceEParms (.h .cpp) JSpaceParts (.h .cpp) JSpaceProperties (.h .cpp) JRangeFilter (.h .cpp) JTimer.h JTimerCuda.h JVarsAscii (.h .cpp) TypesDef.h JFormatFiles2.h *JFormatFiles2.lib / libjformatfiles2.a* JSphMotion.h *JSphMotion.lib / libjsphmotion.a* JXml.h *JXml.lib / libjxml.a*	main.cpp JCfgRun (.h .cpp) JSph (.h .cpp) JPartsLoad (.h .cpp) JPartsOut (.h .cpp) JSphDtFixed (.h .cpp) JSphVarAcc (.h .cpp) Types.h	
	JSphCpu (.h .cpp)	JSphGpu (.h .cpp) JSphGpu_ker (.h .cu)
	JSphCpuSingle (.h .cpp)	JSphGpuSingle (.h .cpp)
	JSphTimersCpu.h	JSphTimersGpu.h
	JCellDivCpu (.h .cpp)	JCellDivGpu (.h .cpp) JCellDivGpu_ker (.h .cu)
	JCellDivCpuSingle (.h .cpp)	JCellDivGpuSingle (.h .cpp) JCellDivGpuSingle_ker (.h .cu)
	JPeriodicCpu (.h .cpp)	JPeriodicGpu (.h .cpp) JPeriodicGpu_ker (.h .cu)
		JGpuArrays (.h .cpp)

COMMON FILES:

Functions.h & Functions.cpp
Declares/implements basic/general functions for the entire application.

JException.h & JException.cpp
Declares/implements the class that defines exceptions with the information of the class and method.

JFloatingData.h & JFloatingData.cpp
Declares/implements the class that allows reading/writing files with data of floating bodies.

JLog2.h & JLog2.cpp
Declares/implements the class that manages the output of information in the file Run.out and on screen.

JObject.h & JObject.cpp
Declares/implements the class that defines objects with methods that throws exceptions.

JObjectGpu.h & JObjectGpu.cpp
Declares/implements the class that defines objects with methods that throws exceptions about tasks in GPU.

JPartData.h & JPartData.cpp
Declares/implements the class that allows reading/writing files with data of particles in formats binx2, ascii..

JPtxasInfo.h & JPtxasInfo.cpp
Declares/implements the class that returns the number of registers of each CUDA kernel.

JSpaceCtes.h & JSpaceCtes.cpp
Declares/implements the class that manages the info of constants from the input XML file.

JSpaceEParms.h & JSpaceEParms.cpp
Declares/implements the class that manages the info of execution parameters from the input XML file.

JSpaceParts.h & JSpaceParts.cpp
Declares/implements the class that manages the info of particles from the input XML file.

JSpaceProperties.h & JSpaceProperties.cpp
Declares/implements the class that manages the properties assigned to the particles in the XML file.

JRangeFilter.h & JRangeFilter.cpp
Declares/implements the class that facilitates filtering values within a list.

JTimer.h
Declares the class that defines a class to measure short time intervals.

JTimerCuda.h
Declares the class that defines a class to measure short time intervals in GPU using cudaEvent.

JVarsAscii.h & JVarsAscii.cpp
Declares/implements the class that reads variables from a text file in ASCII format.

TypesDef.h
Declares general types and functions for the entire application.

JFormatFiles2.h
Declares the class that provides functions to store particle data in formats VTK, CSV, ASCII.

JFormatFiles2.lib (libjformatfiles2.a)
Precompiled library that provides functions to store particle data in formats VTK, CSV, ASCII.

JSphMotion.h
Declares the class that provides the displacement of moving objects during a time interval

JSphMotion.lib (libjsphmotion.a)
Precompiled library that provides the displacement of moving objects during a time interval.

JXml.h
Declares the class that helps to manage the XML document using library TinyXML

JXml.lib (libjxml.a)
Precompiled library that helps to manage the XML document using library TinyXML.

SPH SOLVER:

main.cpp
Main file of the project that executes the code on CPU or GPU.

JCfgRun.h & JCfgRun.cpp
Declares/implements the class that defines the class responsible of collecting the execution parameters by command line.

JSph.h & JSph.cpp
Declares/implements the class that defines all the attributes and functions that CPU and GPU simulations share.

JPartsLoad.h & JPartsLoad.cpp
Declares/implements the class that manages the initial load of particle data.

JPartsOut.h & JPartsOut.cpp
Declares/implements the class that stores excluded particles at each instant till writing the output file.

JSphDtFixed.h & JSphDtFixed.cpp
Declares/implements the class that manages the use of prefixed values of DT loaded from an input file.

JSphVarAcc.h & JSphVarAcc.cpp
Declares/implements the class that manages the application of external forces to different blocks of particles (with the same MK).

Types.h
Defines specific types for the SPH application.

SPH SOLVER ONLY FOR CPU EXECUTIONS:

JSphCpu.h & JSphCpu.cpp
Declares/implements the class that defines the attributes and functions used only in CPU simulations.

JSphCpuSingle.h & JSphCpuSingle.cpp
Declares/implements the class that defines the attributes and functions used only in Single-CPU.

JSphTimersCpu.h
Measures time intervals during CPU execution.

JCellDivCpu.h & JCellDivCpu.cpp
Declares/implements the class responsible of computing the Neighbour List in CPU.

JCellDivCpuSingle.h & JCellDivCpuSingle.cpp
Declares/implements the class responsible of computing the Neighbour List in Single-CPU.

JPeriodicCpu.h & JPeriodicCpu.cpp
Declares/implements the class that manages the interactions between periodic edges in CPU.

SPH SOLVER ONLY FOR GPU EXECUTIONS:

JSphGpu.h & JSphGpu.cpp
Declares/implements the class that defines the attributes and functions used only in GPU simulations.

JSphGpu_ker.h & JSphGpu_ker.cu
Declares/implements functions and CUDA kernels for the particle interaction and system update.

JSphGpuSingle.h & JSphGpuSingle.cpp
Declares/implements the class that defines the attributes and functions used only in Single-GPU.

JSphTimersGpu.h
Measures time intervals during GPU execution.

JCellDivGpu.h & JCellDivGpu.cpp
Declares/implements the class that defines the class responsible of computing the Neighbour List in GPU.

JCellDivGpu_ker.h & JCellDivGpu_ker.cu
Declares/implements functions and CUDA kernels to compute operations of the Neighbour List.

JCellDivGpuSingle.h & JCellDivGpuSingle.cpp
Declares/implements the class that defines the class responsible of computing the Neighbour List in Single-GPU.

JCellDivGpuSingle_ker.h & JCellDivGpuSingle_ker.cu
Declares/implements functions and CUDA kernels to compute operations of the Neighbour List.

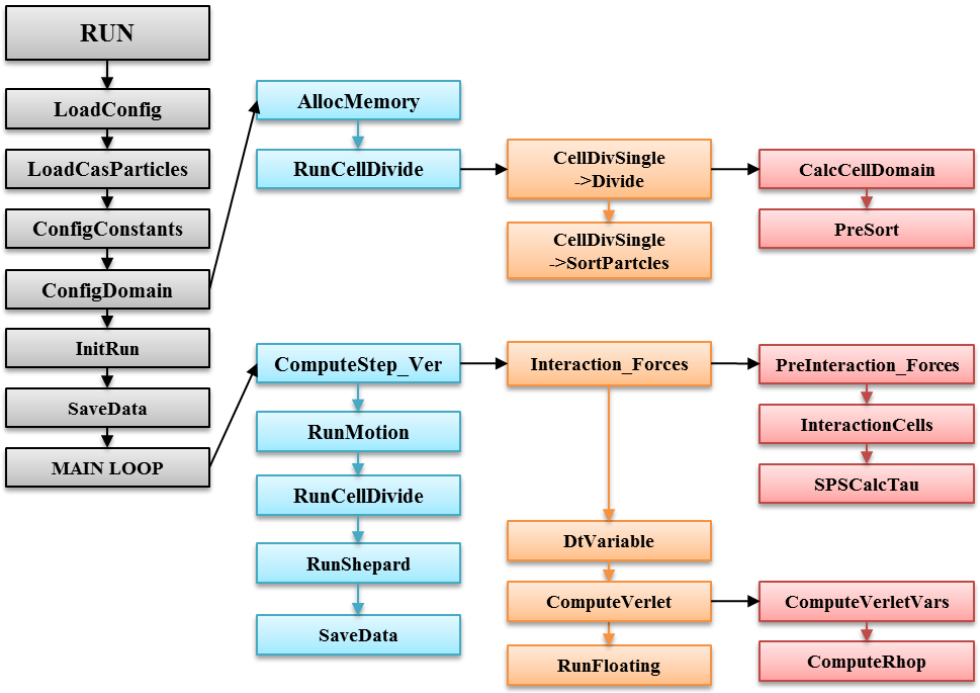
JPeriodicGpu.h & JPeriodicGpu.cpp
Declares/implements the class that manages the interactions between periodic edges in GPU.

JPeriodicGpu_ker.h & JPeriodicGpu_ker.cu
Declares/implements functions and CUDA kernels to obtain particles that interact with periodic edges.

JGpuArrays.h & JGpuArrays.cpp
Declares/implements the class that manages arrays with memory allocated in GPU.

5.1 CPU source files

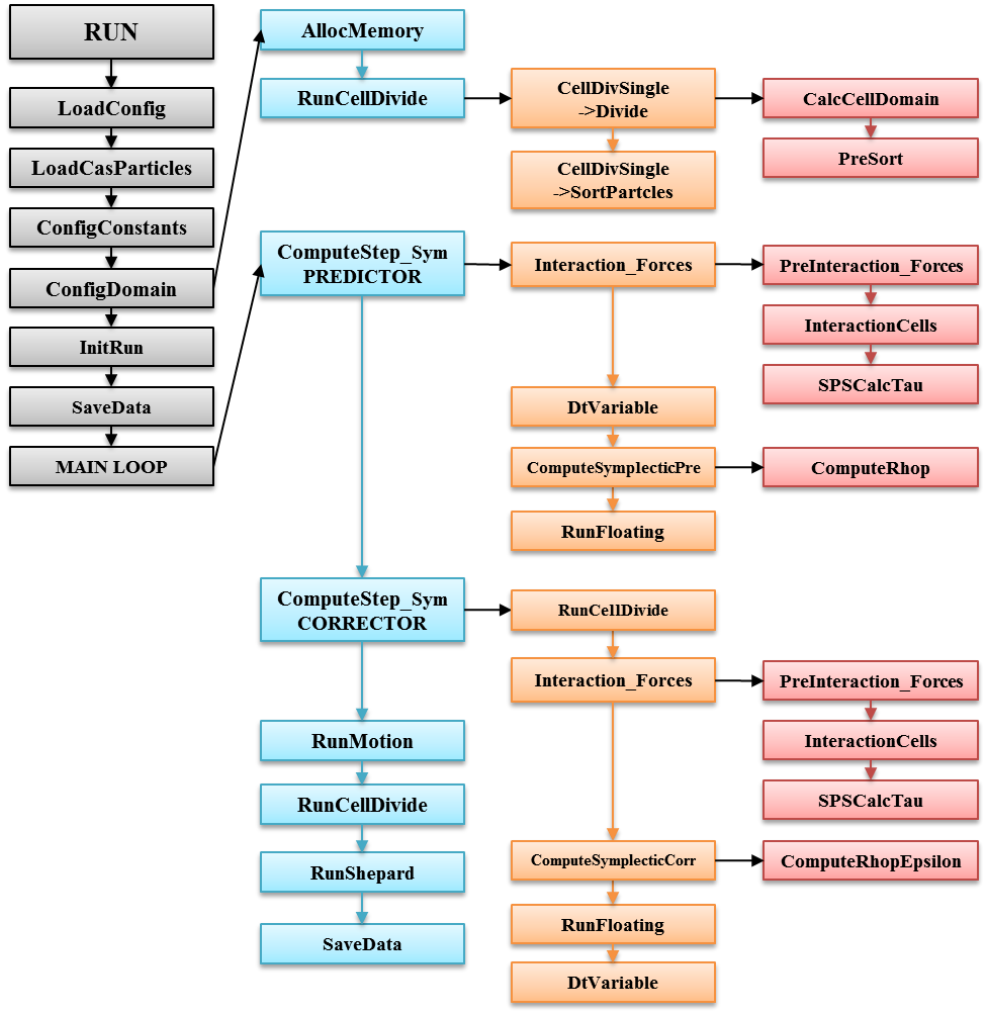
The source file **JSphCpuSingle.cpp** can be better understood with the help of the outline represented in Figure 5-2:



RUN	Starts simulation.
LoadConfig	Loads the configuration of the execution.
LoadCaseParticles	Loads particles of the case to be processed.
ConfigConstants	Configures value of constants.
ConfigDomain	Configuration of the current domain.
AllocMemory	Allocates memory of main data.
RunCellDivide	Computes neighbour list.
CellDivSingle->Divide	Division of particles in cells.
CalcCellDomain	Computes limits of the domain.
PreSort	Computes cell of each particle.
CellDivSingle->SortParticles	Reorders data of all particles.
InitRun	Initialization of arrays and variables for the execution.
SaveData	Stores file with particle data of the initial instant.
MAIN LOOP	Main loop of the simulation.
ComputeStep_Ver	Computes particle interaction and updates system using Verlet.
Interaction_Forces	Computes particle interaction.
PreInteraction_Forces	Prepares variables for particle interaction.
InteractionCells	Interaction between cells.
SPSCalcTau	Computes sub-particle stress tensor for SPS model.
DtVariable	Computes a variable time step.
ComputeVerlet	Updates system using Verlet.
ComputeVerletVars	Computes new values of position and velocity using Verlet.
ComputeRhop	Computes new values of density.
RunFloating	Processes movement of particles of floating objects.
RunMotion	Processes movement of moving boundary particles.
RunCellDivide	Computes neighbour list.
RunShepard	Applies Shepard density filter.
SaveData	Creates files with output data.

Figure 5-2. Outline of **JSphCpuSingle.cpp** when using Verlet time algorithm.

The previous outline corresponds to Verlet algorithm, but if Symplectic is used the step is split in predictor and corrector. Thus, Figure 5-3 shows the structure of the CPU code using this time scheme:



ComputeStep_Sym	Computes particle interaction and updates system with Symplectic.
PREDICTOR	Predictor step.
Interaction_Forces	Computes particle interaction.
PreInteraction_Forces	Prepares variables for particle interaction.
InteractionCells	Interaction between cells.
SPSCalcTau	Computes sub-particle stress tensor for SPS model.
DtVariable	Computes a variable time step.
ComputeSymplecticPre	Computes new values of position and velocity with Symplectic-Predictor.
ComputeRhop	Computes new values of density.
RunFloating	Processes particles of floating objects.
CORRECTOR	Corrector step.
RunCellDivide	Computes neighbour list.
Interaction_Forces	Computes particle interaction.
PreInteraction_Forces	Prepares variables for particle interaction.
InteractionCells	Interaction between cells using corrected force in Predictor.
SPSCalcTau	Computes sub-particle stress tensor for SPS model.
ComputeSymplecticCorr	Computes new values of position and velocity with Symplectic-Corrector.
ComputeRhopEpsilon	Computes new values of density in Symplectic-Corrector.
RunFloating	Processes movement of particles of floating objects.

Figure 5-3. Outline of **JSphCpuSingle.cpp** when using Symplectic time algorithm.

Note that *JSphCpuSingle::InteractionCells* will perform the particle interaction in CPU in terms of cells. Thus, the interaction between cells can be performed with a single execution thread (*InteractionCells_Single*), or using different threads thanks to the OpenMP implementation (*InteractionCells_Static* & *InteractionCells_Dynamic*). Furthermore, particles of a cell can interact with particles of the cell itself (*InteractCelij*) or with particles of the adjacent cells (*InteractSelf*).

Note that *JSphCpuSingle::InteractionCells* will call different computations depending on the parameter <INTER_XXXX>; the interaction between cells is carried out to compute different options as it can be seen in Figure 5-4 and Table 5-2:

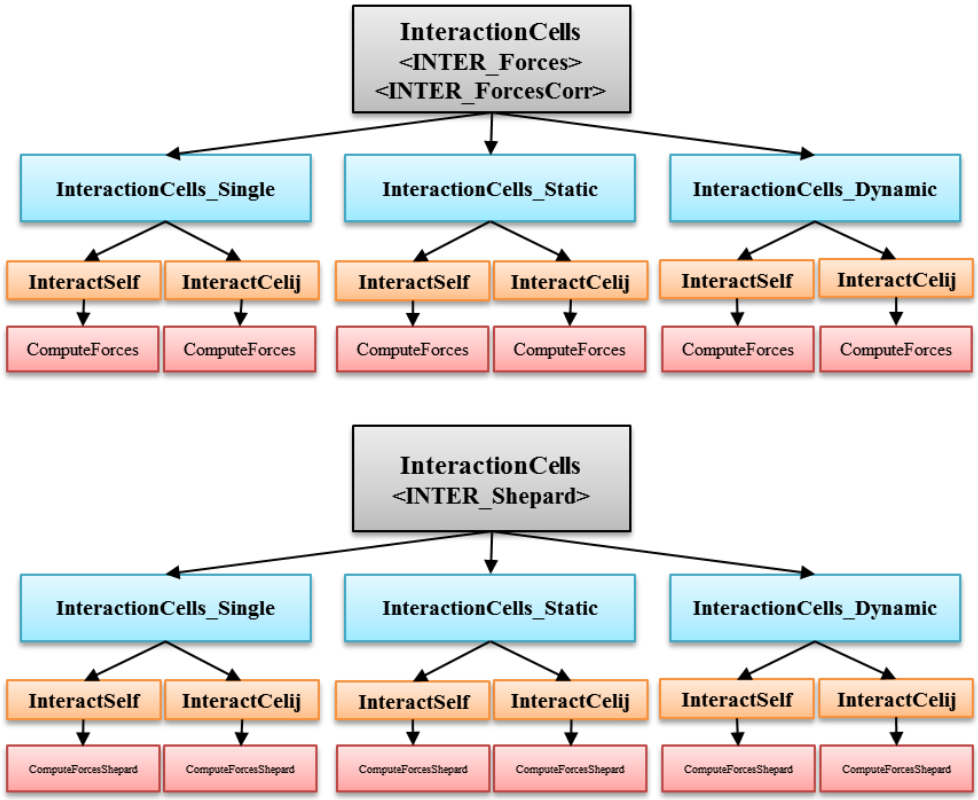


Figure 5-4. Call graph for the function *InteractionCells*.

Table 5-2. Different options can be computed during cells interactions on CPU executions.

InteractionCells<INTER_Forces>->ComputeForces	Interaction between cells in Verlet and Symplectic-predictor.
InteractionCells<INTER_ForcesCorr>->ComputeForces	Interaction between cells in Symplectic-corrector.
InteractionCells<INTER_Shepard>->ComputeForcesShepard	Interaction between cells for Shepard filter.

As mentioned before, a more complete documentation was generated using Doxygen. For example the call graph of *ComputeStep_Ver* function can be seen in Figure 5-5:

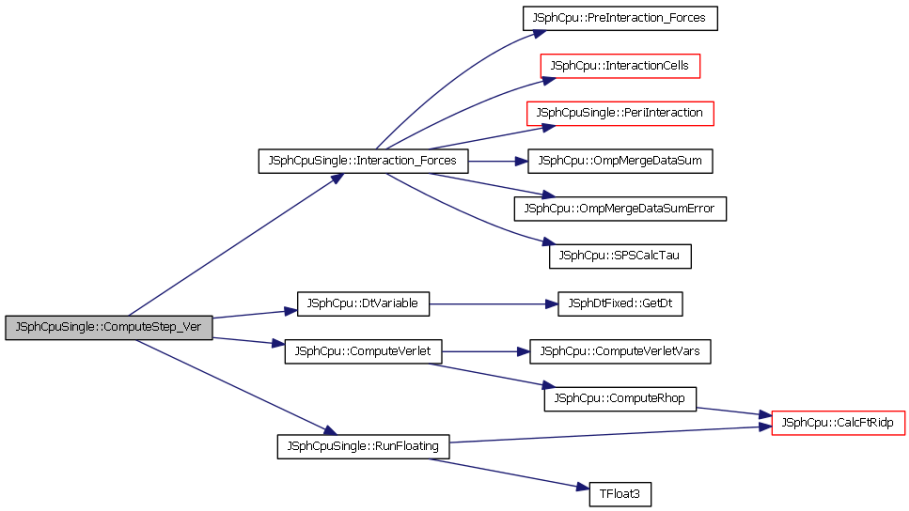


Figure 5-5. Call graph for the function *ComputeStep_Ver* in Doxygen.

And the call graph of *ComputeStep_Sym* function can be seen in Figure 5-6:

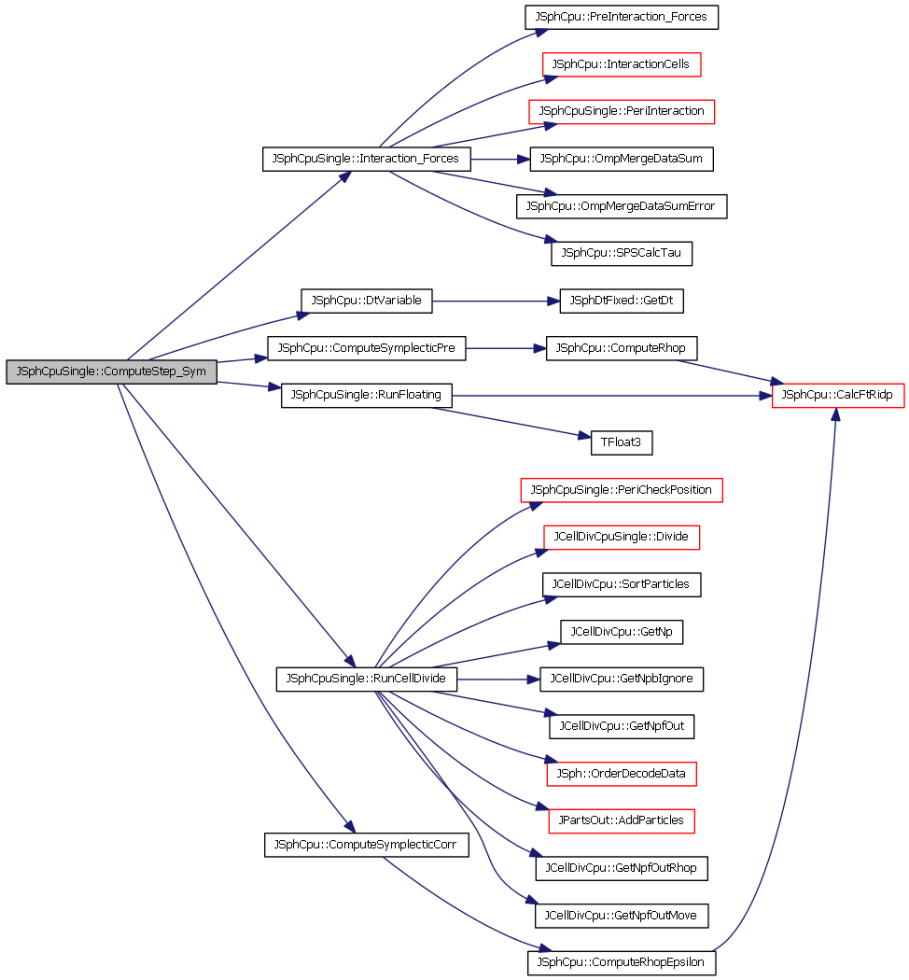
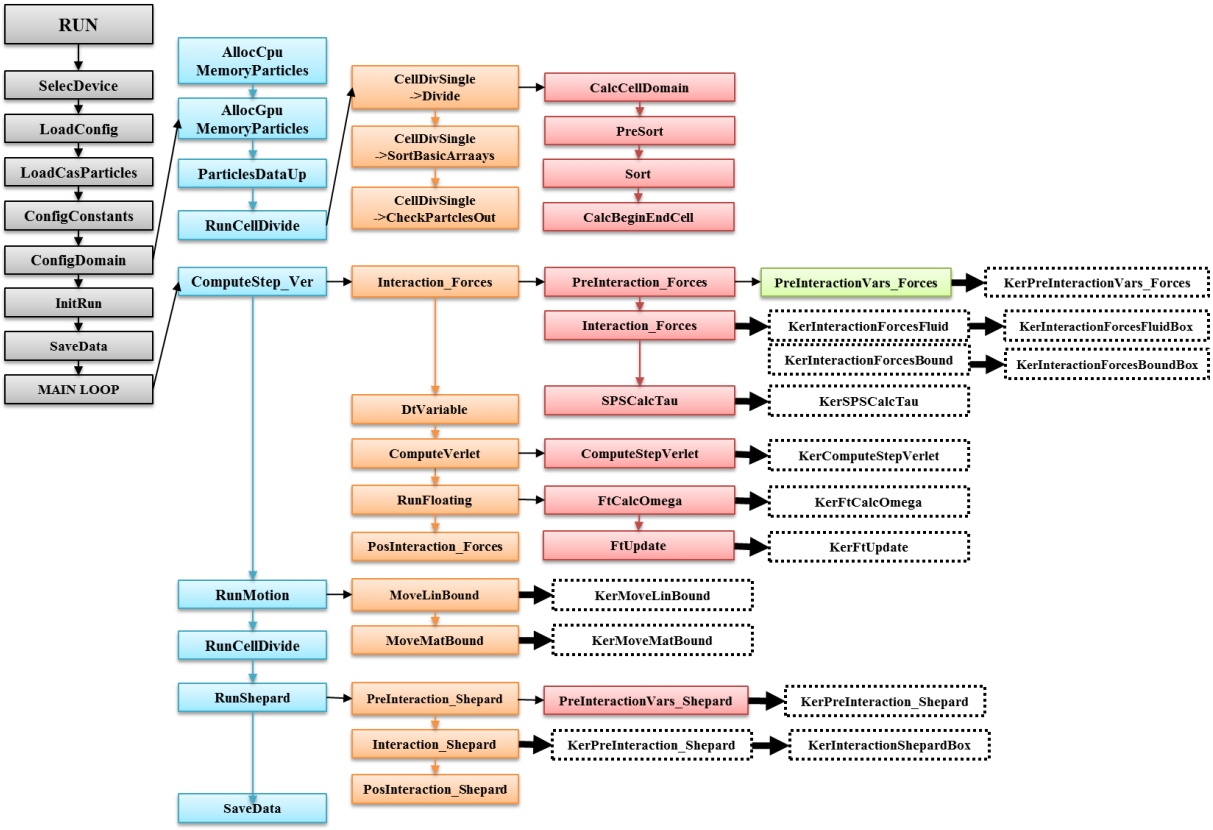


Figure 5-6. Call graph for the function *ComputeStep_Sym* in Doxygen.

5.2 GPU source files

The source file **JSphGpuSingle.cpp** can also be understood better with the outline represented in Figure 5-7 that includes the functions implemented in the GPU files:



RUN	Starts simulation.
SelecDevice	Initialises CUDA devic
LoadConfig	Loads the configuration of the execution.
LoadCaseParticles	Loads particles of the case to be processed.
ConfigConstants	Configures value of constants.
ConfigDomain	Configuration of the current domain.
AllocCpuMemoryParticles	Allocates memory in CPU of main data of particles.
AllocGpuMemoryParticles	Allocates memory in GPU of main data of particles.
RunCellDivide	Computes neighbour list.
CellDivSingle->Divide	Division of particles in cells.
CalcCellDomain	Computes limits of the domain.
PreSort	Computes cell of each particle.
Sort	Reorders values.
CalcBeginEndCell	Computes initial and final particle of each cell.
CellDivSingle->SortBasicArrays	Reorders basic arrays.
CellDivSingle->CheckParticlesOut	Checks excluded particles.
InitRun	Initialization of arrays and variables for the execution.
SaveData	Stores file with particle data of the initial instant.

MAIN LOOP	Main loop of the simulation.
ComputeStep_Ver	Computes particle interaction and updates system using Verlet.
Interaction_Forces	Computes particle interaction.
PreInteraction_Forces	Prepares variables for particle interaction.
PreInteractionVars_Forces	Prepares variables for particle interaction.
InteractionForces	Interaction between particles.
SPSCalcTau	Computes sub-particle stress tensor for SPS model.
DtVariable	Computes a variable time step.
ComputeVerlet	Updates system using Verlet.
ComputeStepVerlet	Computes new values using Verlet.
RunFloating	Processes movement of particles of floating objects.
FtCalcOmega	Computes values for a floating body.
FtUpdate	Updates particles of a floating body.
PosInteractionForces	Releases assigned memory of GPU arrays.
RunMotion	Processes movement of moving boundary particles.
MoveLinBound	Applies a linear movement to a set of particles.
MoveMatBound	Applies a matrix movement to a set of particles.
RunCellDivide	Computes neighbour list.
RunShepard	Applies Shepard density filter.
PreInteraction_Shepard	Prepares variables for Shepard interaction.
PreInteractionVars_Shepard	Prepares variables for Shepard interaction.
Interaction_Shepard	Particle interaction to compute Shepard density filter.
PosInteraction_Shepard	Releases assigned memory of GPU arrays for Shepard interaction.
SaveData	Creates files with output data.

Figure 5-7. Outline of **JSphGpuSingle.cpp** when using Verlet time algorithm.

The dashed boxes indicates the CUDA kernels implemented in the CUDA files (*.*cu*).

6. Compiling DualSPHysics

The code can be compiled for either CPU executions or GPU executions (but not both simultaneously). Please note that both the C++ and CUDA version of the code contain the same features and options. Most of the source code is common to CPU and GPU which allows the code to be run on workstations without a CUDA-enabled GPU, using only the CPU implementation.

To run DualSPHysics on GPU using an executable, only an Nvidia CUDA-enabled GPU card is needed and the latest version of the GPU driver must be installed. However, to compile the source code, the GPU programming language CUDA and *nvcc* compiler must be installed on your computer. CUDA Toolkit X.X can be downloaded from Nvidia website http://developer.nvidia.com/cuda-toolkit-XX. CUDA versions 4.0, 4.1, 4.2, 5.0, and 5.5 have been tested.

Once the C++ compiler (for example *gcc*) and the CUDA compiler (*nvcc*) have been installed in your machine, you can download the relevant files from the directory **DUALSPHYSICS/SOURCE/DualSPHysics_3:**

6.1 Windows compilation

In **DUALSPHYSICS/SOURCE/DualSPHysics_3** there are also several folders:
-Source: contains all the source files and *DualSPHysics_v3.sln*;
-Libs: precompiled libraries for x64;
-Code_documentation: including the documentation generated by Doxygen;

The project file *DualSPHysics_v3.sln* is provided to be opened with Visual Studio 2010. Also different configurations can be chosen for compilation:

a) **Release** for CPU and GPU

b) **ReleaseCPU** only for CPU

The result of the compilation is the executable *DualSPHysics._win64.exe* or *DualSPHysicsCPU_win64.exe* created in **DUALSPHYSICS/EXECS.**

The Visual Studio project is created including the libraries for OpenMP in the executable. To not include them, user can modify *Props config -> C/C++ -> Language -> OpenMp* and compile again
The use of OpenMP can be also deactivated by commenting the code line in **Types.h**:

```
#define _WITHOMP          ///<Enables/Disables OpenMP.
```

In order to compile for GPU, some properties of compilation of the CUDA source files have been defined by default.
Checking the command line of compilation of ***.cu*** files:

```
rem ***Variables of compilation
set CudaPath=$(CUDA_PATH_V5_5)
set CudaNvcc="%CudaPath%\bin\nvcc.exe"
set CudaInc="%CudaPath%include"

rem ***Selection of GPU hardware for compilation
rem set gpuhw=-gencode=arch=compute_30,code=\"sm_30,compute_30\"
rem set gpuhw=-gencode=arch=compute_20,code=\"sm_20,compute_20\"
rem set gpuhw=-gencode=arch=compute_12,code=\"sm_12,compute_12\"
set gpuhw=-gencode=arch=compute_12,code=\"sm_12,compute_12\"
-gencode=arch=compute_20,code=\"sm_20,compute_20\"

rem ***Activate the visualisation of register per kernel
set ptxinfo=--ptxas-options -v
set ptxinfo=

rem ***Sentence of compilation
%CudaNvcc% -ccbin "$(VCInstallDir)bin" %gpuhw% -use_fast_math -c %ptxinfo% -DWIN32
-D_CONSOLE -D_MBCS -Xcompiler /EHsc,/W3,/nologo,/O2,/Zi,/MT -m 64 -I%CudaInc% -I./ -o
Intermediate\$(ProjectName)_$(Configuration)_$(PlatformName)\%(Filename).obj
Source\%(Filename)%(Extension)
```

The user can modify the *CudaPath*, *CudaNvcc* and *CudaInc*. As it is, the GPU code is compiled for "sm_12,compute_12" and "sm_20,compute_20" and CUDA v5.5. In the windows package, in EXECS, the executable was already compiled using CUDA v4.0 to ensure compatibility.

By default, the log file generated by the compiler is stored in the file *DualSPHysics_ ptxasinfo*. Thus, any possible error in the compilation of *JSphGpu_ker.cu* can be identified in this *ptxasinfo* file.

6.2 Linux compilation

In **DUALSPHYSICS/SOURCE/DualSPHysics_3** there are also several folders:
-Source: contains all the source files and *makefiles*;
-Libs: precompiled libraries for x64;
-Code_documentation: including the documentation generated by Doxygen;

Makefiles can be used to compile the code in linux:

a) make –f **Makefile** full compilation just using *make* command
b) make –f **Makefile_cpu** only for CPU

The result of the compilation is the executable *DualSPHysics_linux64* or *DualSPHysicsCPU_linux64* created in **DUALSPHYSICS/EXECS.**

To exclude the use of OpenMP you have to remove the flags *–fopenmp* and *-lgomp* in the Makefile and comment line `#define _WITHOMP` in **Types.h**.
This is the content of the file **Makefile** for linux:

```
#=============== Compilation Options ===============
USE_DEBUG=NO
USE_FAST_MATH=YES
USE_NATIVE_CPU_OPTIMIZATIONS=YES

#=============== CUDA toolkit directory (make appropriate for local CUDA installation)==============
DIRTOOLKIT=/usr/local/cuda

#=============== Files to compile ===============
OBJ_BASIC=main.o Functions.o JCellDivCpu.o JCfgRun.o JException.o JFloatingData.o JLog2.o
JObject.o JPartData.o JPartsOut.o JPeriodicCpu.o JSpaceCtes.o JSpaceEParms.o JSpaceParts.o
JSph.o JSphCpu.o JSphDtFixed.o JSphVarAcc.o JVarsAscii.o
OBJ_CPU_SINGLE=JCellDivCpuSingle.o JSphCpuSingle.o JPartsLoad.o
OBJ_GPU=JCellDivGpu.o JGpuArrays.o JObjectGpu.o JPeriodicGpu.o JPtxasInfo.o JSphGpu.o
OBJ_GPU_SINGLE=JCellDivGpuSingle.o JSphGpuSingle.o
OBJ_CUDA=JCellDivGpu_ker.o JSphGpu_ker.o JPeriodicGpu_ker.o
OBJ_CUDA_SINGLE=JCellDivGpuSingle_ker.o
OBJECTS=$(OBJ_BASIC) $(OBJ_CPU_SINGLE) $(OBJ_GPU) $(OBJ_CUDA) $(OBJ_GPU_SINGLE)
$(OBJ_CUDA_SINGLE)

#=============== Select GPU architectures ===============
GENCODE:=$(GENCODE) -gencode=arch=compute_12,code=\"sm_12,compute_12\"
GENCODE:=$(GENCODE) -gencode=arch=compute_20,code=\"sm_20,compute_20\"

#=============== DualSPHysics libs to be included ===============
ifeq ($(USE_DEBUG), NO)
JLIBS=-L./ -ljxml_64 -ljformatfiles2_64 -ljsphmotion_64
else
JLIBS=-L./ -ljxml_64_debug -ljformatfiles2_64_debug -ljsphmotion_64_debug
endif

#=============== CPU Code Compilation (make appropriate for chosen compiler) ===============
CC=g++
ifeq ($(USE_DEBUG), NO)
CCFLAGS=-c -O3 -fopenmp -D_WITHGPU
else
CCFLAGS=-c -O0 -g -Wall -fopenmp -D_WITHGPU
endif
CCLINKFLAGS=-fopenmp -lgomp

ifeq ($(USE_FAST_MATH), YES)
CCFLAGS+= -ffast-math
endif
ifeq ($(USE_NATIVE_CPU_OPTIMIZATIONS), YES)
CCFLAGS+= -march=native
endif

#=============== GPU Code Compilation ===============
CCFLAGS := $(CCFLAGS) -I./ -I$(DIRTOOLKIT)/include
CCLINKFLAGS := $(CCLINKFLAGS) -L$(DIRTOOLKIT)/lib64 -lcudart
NCC=nvcc
ifeq ($(USE_DEBUG), NO)
NCCFLAGS=-c $(GENCODE) -O3
else
NCCFLAGS=-c $(GENCODE) -O0 -g
endif
ifeq ($(USE_FAST_MATH), YES)
NCCFLAGS+= -use_fast_math
endif
```

```
all:DualSPHysics_linux64
rm -rf *.o
ifeq ($(USE_DEBUG), NO)
@echo " --- Compiled Release GPU version ---"
else
@echo " --- Compiled Debug GPU version ---"
mv DualSPHysics_linux64 DualSPHysics_linux64_debug
mv DualSPHysics_linux64_ptxasinfo DualSPHysics_linux64_debug_ptxasinfo
endif
DualSPHysics_linux64: $(OBJECTS)
$(CC) $(OBJECTS) $(CCLINKFLAGS) -o $@ $(JLIBS)
.cpp.o:
$(CC) $(CCFLAGS) $<
JSphGpu_ker.o: JSphGpu_ker.cu
$(NCC) $(NCCFLAGS) --ptxas-options -v JSphGpu_ker.cu 2>DualSPHysics_linux64_ptxasinfo
JCellDivGpu_ker.o: JCellDivGpu_ker.cu
$(NCC) $(NCCFLAGS) JCellDivGpu_ker.cu
JCellDivGpuSingle_ker.o: JCellDivGpuSingle_ker.cu
$(NCC) $(NCCFLAGS) JCellDivGpuSingle_ker.cu
JCellDivGpuMpi_ker.o: JCellDivGpuMpi_ker.cu
$(NCC) $(NCCFLAGS) JCellDivGpuMpi_ker.cu
JPeriodicGpu_ker.o: JPeriodicGpu_ker.cu
$(NCC) $(NCCFLAGS) JPeriodicGpu_ker.cu
clean:
rm -rf *.o DualSPHysics_linux64 DualSPHysics_linux64_ptxasinfo
DualSPHysics_linux64_debug DualSPHysics_linux64_debug_ptxasinfo
```

The user can modify the compilation options, the path of the CUDA toolkit directory, the GPU architecture... As it is, the GPU code is compiled for “sm_12,compute_12” and “sm_20,compute_20” and CUDA v5.0. In the linux package, in EXECS, the executable was already compiled using CUDA v4.0 to ensure compatibility.

By default, the log file generated by the compiler is stored in the file *DualSPHysics_ptxasinfo*. Thus, any possible error in the compilation of *JSphGpu_ker.cu* can be identified in this *ptxasinfo* file.

7. Format Files

The codes provided within the DualSPHysics package present some important improvements in comparison to the codes available within SPHysics. One of them is related to the format of the files that are used as input and output data along the execution of DualSPHysics and the pre-processing and post-processing codes. Three new formats are introduced now in comparison to SPHysics: XML, binary and VTK-binary.

XML File

The E*X*tensible *M*arkup *L*anguage is textual data format compatible with any hardware and software. It is based on a set of labels to organise the information and they can be loaded or written easily using any standard text editor. This format will be used for the input files for the pre-processing code and the SPH solver.

BINARY File

The output data in the SPHysics code is written in text files, so ASCII format is used. ASCII files present some interesting advantages such as visibility and portability, however they also present important disadvantages: data stored in text format consumes at least six times more memory than the same data stored in binary format, precision is reduced when values are converted from real numbers to text and read or write data in ASCII is more expensive (two orders of magnitude). Since DualSPHysics allows performing simulations with a high number of particles, a binary file format is necessary to avoid these problems. Binary format reduces the volume of the files and the time dedicated to generate them. The file format used in DualSPHysics is named BINX2 (.bi2). These files contain only the meaningful information of particle properties. In this way, some variables can be removed, e.g., the pressure is not stored since it can be calculated starting from the density using the equation of state. The mass values are constant for fluid particles and for boundaries so only two values are used instead of an array. The position of fixed boundary particles is only stored in the first file since they remain unchanged throughout the simulation. Data for particles that leave the limits of the domain are stored in an independent file which leads to an additional saving.

Advantages of BINX2:

- Memory storage reduction: an important reduction in the file size is obtained using the binary format by saving only the values of the particles that change along the simulation.
- Fast Access: read-write timing is decreased since binary conversion is faster than the ASCII one.
- No precision lost.
- Portability (to different architectures, to Linux or Windows…)
- Additional information: a header is stored at the beginning of each file with information about number of particles, mass value, the smoothing length, inter-particle spacing, time…

VTK File

This format is basically used for visualization of the results; not only the particle positions, but also physical magnitudes obtained numerically for the particles involved in the simulations.
VTK supports many data types, such as scalar, vector, tensor, texture, and also supports different algorithms such as polygon reduction, mesh smoothing, cutting, contouring and Delaunay triangulation. Essentially, VTK consists of a header that describes the data and includes any other useful information, the dataset structure with the geometry and topology of the dataset and its attributes. Here we use VTK of POLYDATA type with legacy-binary format that is easy for read-write operations.

8. Pre-processing

A program named **GenCase** is included to define the initial configuration of the simulation, movement description of moving objects and the parameters of the execution in DualSPHysics. All this information is contained in an **input file** in XML format; **Case_Def.xml**. Two **output files** are created after running GenCase: **Case.xml** and **Case.bi2** (the input files for DualSPHysics code). These input (red boxes) and output files (blue boxes) can be observed in Figure 8-1. **Case.xml** contains all the parameters of the system configuration and its execution such as key variables (smoothing length, reference density, gravity, coefficient to calculate pressure, speed of sound…), the number of particles in the system, movement definition of moving boundaries and properties of moving bodies. **Case.bi2** contains the initial state of the particles (number of particles, position, velocity and density).

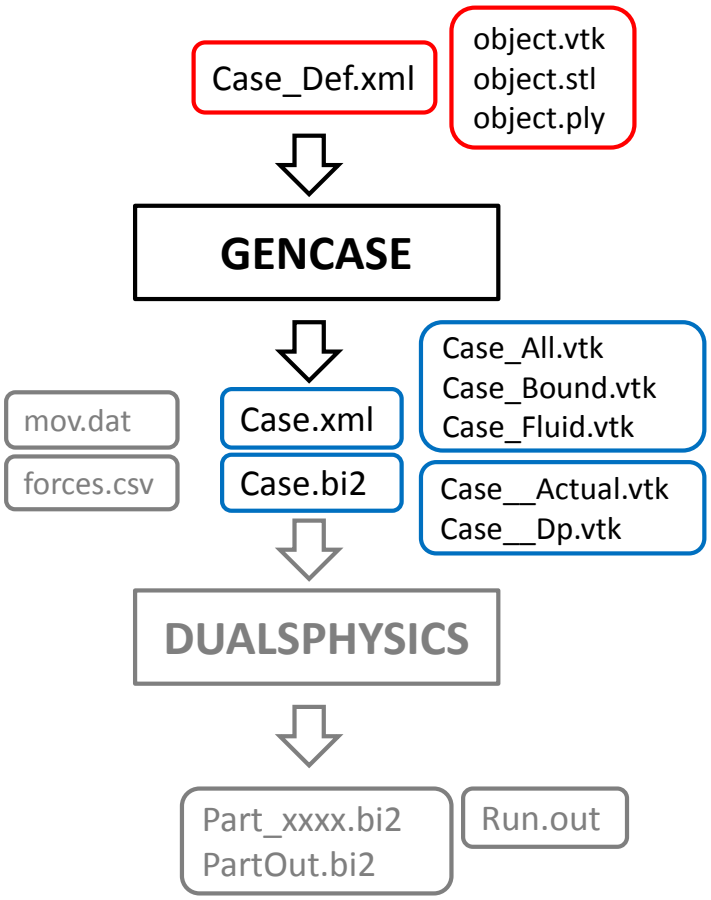


Figure 8-1. Input (red) and output (blue) files of GenCase code.

GenCase employs a 3D Cartesian mesh to locate particles. The idea is to build any object using particles. These particles are created at the nodes of the 3-D Cartesian mesh. Firstly, the mesh nodes around the object are defined and then particles are created only in the nodes needed to draw the desired geometry. Figure 8-2 illustrates how this mesh is used; in this case a triangle is generated in 2D. First the nodes of a mesh are defined starting from the maximum dimensions of the desired triangle, then

the edges of the triangle are defined and finally particles are created at the nodes of the Cartesian mesh which are inside the triangle.

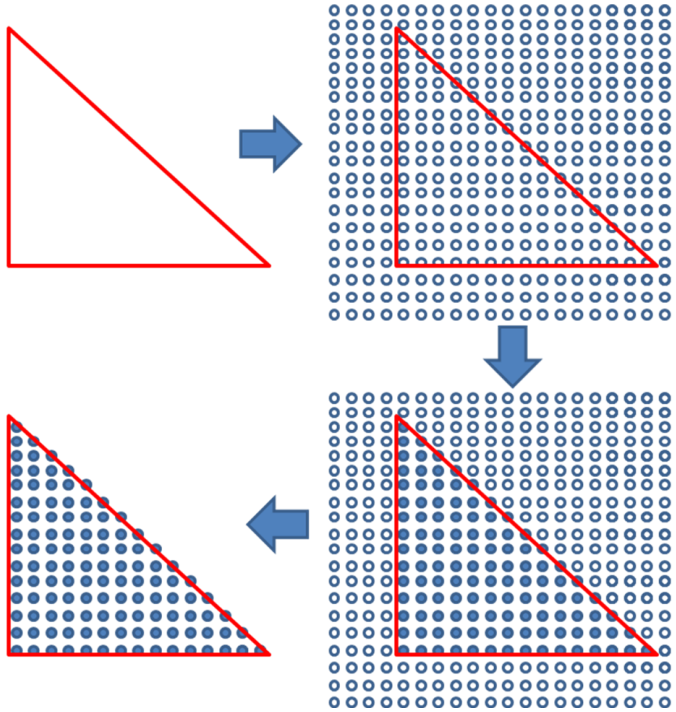


Figure 8-2. Generation of a 2-D triangle formed by particles using GenCase.

All particles are placed over a regular Cartesian grid. The geometry of the case is defined independently to the inter-particle distance. This allows the discretization of each test case with different number of particles simply by varying *dp*. Furthermore, **GenCase** is very fast and able to generate millions of particles only in seconds on CPU.

Very complex geometries can be easily created since a wide variety of commands (**labels in the XML file**) are available to create different objects; points, lines, triangles, quadrilateral, polygons, pyramids, prisms, boxes, beaches, spheres, ellipsoids, cylinders, waves (**<drawpoint />, <drawpoints />, <drawline />, <drawlines />, <drawtriangle />, <drawquadri />, <drawtrianglesstrip />, <drawtrianglesfan />, <drawpolygon>, <drawtriangles />, <drawpyramid />, <drawprism />, <drawbox />, <drawbeach />, <drawsphere />, <drawellipsoid />, <drawcylinder />, <drawwave />**).

Once the mesh nodes that represent the desired object are selected, these points are stored as a matrix of nodes. The shape of the object can be transformed using a translation (**<move />**), a scaling (**<scale />**) or a rotation (**<rotate />**, **<rotateline />**). The main process is creating particles at the nodes, different type of particles can be created; a fluid particle (**<setmkfluid />**), a boundary particle (**<setmkbound />**) or none (**<setmkvoid />**). Hence, **mk** is the value used to mark a set of particles with a common feature in the simulation. Particles can be created only at the object surface (**<setdrawmode mode="face" />**), or only inside the bounds of the objects (**<setdrawmode mode="solid" />**) or both (**<setdrawmode mode="full" />**).

The set of fluid particles can be labelled with features or special behaviours (<**initials** />). For example, initial velocity (<**velocity** />) can be imposed for fluid particles or a solitary wave can be defined (<**velwave** />). Furthermore, particles can be defined as part of a floating object (<**floatings** />).

Once boundaries are defined, filling a region with fluid particles can be easily obtained using the following commands: (<**fillpoint** />, <**fillbox** />, <**fillfigure** />, <**fillprism** />). This works also in the presence of arbitrarily complex geometries.

In cases with more complex geometries, external objects can be imported from 3DS files (Figure 8-3) or CAD files (Figure 8-4). This enables the use of realistic geometries generated by 3D designing application with the drawing commands of GenCase to be combined. These files (3Ds or CAD) must be converted to STL format (<**drawfilestl** />), PLY format (<**drawfileply** />) or VTK format (<**drawfilevtk** />), formats that are easily loaded by **GenCase**. Any object in STL, PLY or VTK (*object.vtk*, *object.stl* or *object.ply* in Figure 8-1) can be split in different triangles and any triangle can be converted into particles using the **GenCase** code.

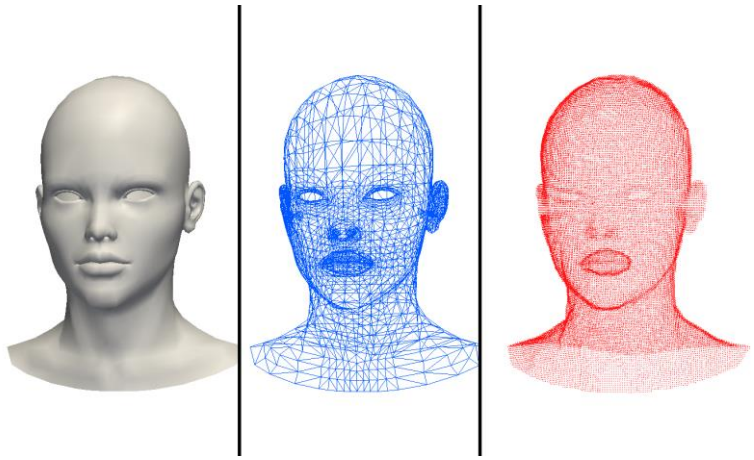


Figure 8-3. Example of a 3D file imported by **GenCase** and converted into particles.

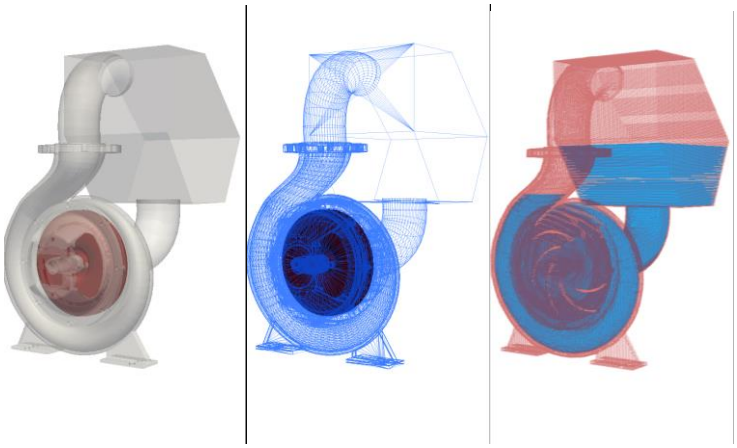


Figure 8-4. Example of a CAD file imported by **GenCase** and converted into particles.

Different kinds of movements can be imposed to a set of particles; linear, rotational, circular, sinusoidal, etc. To help users define movements, a directory with some examples is also included in the DualSPHysics package. Thus, the directory **MOTION** includes:

- *Motion01*: uniform rectilinear motion (<**mvrect** />) that also includes pauses (<**wait** />)
- *Motion02*: combination of two uniform rectilinear motion (<**mvrect** />)
- *Motion03*: movement of an object depending on the movement of another (hierarchy of objects)
- *Motion04*: accelerated rectilinear motion (<**mvrectace** />)
- *Motion05*: rotational motion (<**mvrot** />). See Figure 8-5.
- *Motion06*: accelerated rotation motion (<**mvrotace** />) and accelerated circular motion (<**mvcirace** />). See Figure 8-6.
- *Motion07*: sinusoidal movement (<**mvrectsinu** />**,** <**mvrotsinu** />**,** <**mvcirsinu** />)
- *Motion08*: prescribed movement with data from an external file (<**mvpredef** />)

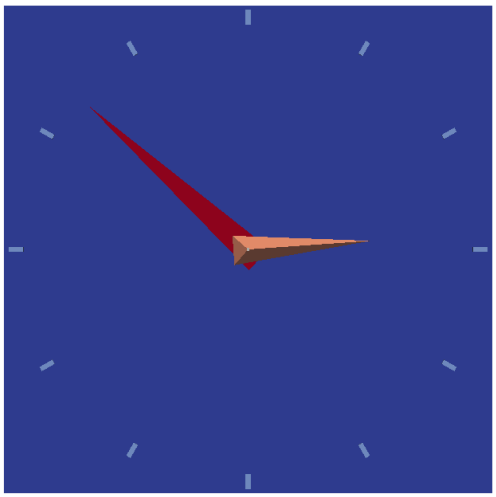


Figure 8-5. Example of rotational motion.

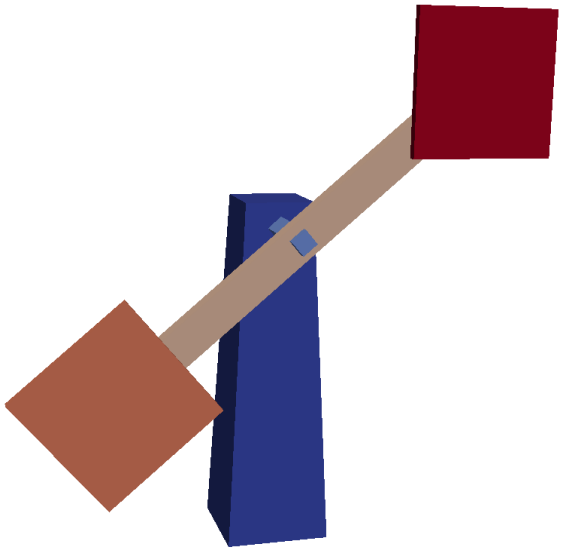


Figure 8-6. Example of accelerated rotation motion and accelerated circular motion.

TO RUN GENCASE: GenCase Case_Def Case [options]

where *Case_Def* is the name of the input file (Case_Def.xml as seen in Figure 8-1) and *Case* will be the name of the output files (Case.xml and Case.bi2) and the *options* are:

-h
Shows information about the different parameters. Typing "GenCase –h" in the command window generates a brief help manual (available in **HELP** folder).

-template
Generates the example file 'CaseTemplate.xml' that can be also found in **HELP** folder.

-dp:<float>
Defines the distance between particles. Just varying this parameter the number of particles will be modifying without changing any other data since all dimensions are given in general dimensions.

-ompthreads:<int>
Indicates the number of threads by host for parallel execution, it takes the number of cores of the device by default (or using zero value).

-save:<values>
Indicates the format of output files.

+/-all:	To choose or reject all options
+/-binx2:	Binary format for the initial configuration (by default)
+/-vtkall:	VTK with all particles of the initial configuration
+/-vtkbound:	VTK with boundary particles of the initial configuration
+/-vtkfluid:	VTK with fluid particles of the initial configuration

Note that when using –save:all, the files **Case_All.vtk**, **Case_Bound.vtk** and **Case_Fluid.vtk** of Figure 8-1 are also generated.

-debug:<int>
Debug level (-1:Off, 0:Explicit, n:Debug level)

A more complete description of the code can be found at the PDF files; GenCase_XML_GUIDE.pdf and ExternalModelsConversion_GUIDE.pdf already available to be downloaded from the DualSPHysics website.

In addition, the NEW **DualSPHysics Pre-processing Interface** (**GenCase GUI)** can also be used for a more user friendly way to define the initial geometry and configuration. All source files and documentation can be also found in the download section of the website.

An example of input XML is shown here; the CaseWavemaker_Def.xml included in **RUN_DIRECTORY/CASEWAVEMAKER**:

```
<casedef>
    <constantsdef>
        <lattice bound="2" fluid="1" />
        <gravity x="0" y="0" z="-9.81" />
        <cflnumber value="0.2" />
        <hswl value="0" auto="true" />
        <coefsound value="10" />
        <coefficient value="0.866025" />
        <gamma value="7" />
        <rhop0 value="1000" />
        <eps value="0.5" />
    </constantsdef>
    <mkconfig boundcount="230" fluidcount="10">
        <mkorientfluid mk="0" orient="Xyz" />
    </mkconfig>
    <geometry>
        <definition dp="0.0125">
            <pointmin x="-0.05" y="-0.05" z="-0.05" />
            <pointmax x="5.1" y="2.1" z="2" />
        </definition>
        <commands>
            <mainlist>
                <setshapemode>real | dp | bound</setshapemode>
                <setdrawmode mode="full" />
                <setmkfluid mk="0" />
                <drawprism mask="0">
                    <point x="5" y="0" z="1.5" />
                    <point x="5" y="0" z="1.1" />
                    <point x="1" y="0" z="0" />
                    <point x="0" y="0" z="0" />
                    <point x="0" y="0" z="1.5" />
                    <point x="5" y="2" z="1.5" />
                    <point x="5" y="2" z="1.1" />
                    <point x="1" y="2" z="0" />
                    <point x="0" y="2" z="0" />
                    <point x="0" y="2" z="1.5" />
                </drawprism>
                <setmkvoid />
                <drawbox>
                    <boxfill>solid</boxfill>
                    <point x="0" y="0" z="0.75" />
                    <size x="5" y="2" z="1" />
                </drawbox>
                <setdrawmode mode="face" />
                <setmkbound mk="10" />
                <drawbox>
                    <boxfill>solid</boxfill>
                    <point x="0" y="0" z="0" />
                    <size x="0.02" y="2" z="1.5" />
                </drawbox>
                <setmkbound mk="0" />
                <drawprism mask="1 | 2 | 6 | 7">
                    <point x="5" y="0" z="1.5" />
                    <point x="5" y="0" z="1.1" />
                    <point x="1" y="0" z="0" />
                    <point x="0" y="0" z="0" />
                    <point x="0" y="0" z="1.5" />
                    <point x="5" y="2" z="1.5" />
                    <point x="5" y="2" z="1.1" />
                    <point x="1" y="2" z="0" />
                    <point x="0" y="2" z="0" />
                    <point x="0" y="2" z="1.5" />
                </drawprism>
                <shapeout file="" reset="true" />
            </mainlist>
        </commands>
    </geometry>
    <motion>
        <objreal ref="10">
            <begin mov="1" start="0" finish="100" />
            <mvrectsinu id="1" duration="5" next="2">
                <freq x="0.5" y="0" z="0" />
                <ampl x="0.25" y="0" z="0" />
                <phase x="4.71238898" y="0" z="0" />
            </mvrectsinu>
            <mvrectsinu id="2" duration="10" next="3">
                <freq x="0.75" y="0" z="0" />
                <ampl x="0.25" y="0" z="0" />
            </mvrectsinu>
            <mvrectsinu id="3" duration="10" next="2">
                <freq x="0.5" y="0" z="0" />
                <ampl x="0.25" y="0" z="0" />
            </mvrectsinu>
        </objreal>
    </motion>
</casedef>
```

Different constants are defined:
lattice= staggered grid (2) for boundaries and cubic grid (1) for fluid particles
gravity=gravity acceleration
cflnumber=coefficient in the Courant condition
hswl=maximum still water level, automatically calculated with TRUE
coefsound=coefficient needed to compute the speed of sound
coefficient=coefficient needed to compute the smoothing length
ALL THESE CONSTANTS ARE ALSO DEFINED IN SPHYSICS

To create particles in the order that grows the x-direction, then y-direction and finally z-direction. This can be changed to plot ID

dp=distance between particles
WHEN CHANGING THIS PARAMETER, THE TOTAL NUMBER OF PARTICLES IS MODIFIED
x,y and *z* values are used to defined the limits of the domain

Volume of fluid:
setmkfluid mk=0,
full to plot particles under the volume limits and the faces
drawprism to plot a figure that mimics a beach

setmkvoid is used to remove particles, to define the maximum water level at z=0.75m since all particles above are removed

Piston Wavemaker:
setmkbound mk=10,
face to plot particles only in the sides of the defined box that formed the wavemaker

Boundary Tank:
setmkbound mk=0,
drawprism to plot the beach
face to plot only in the sides, except faces 1,2,6,7 (see page 26 of *GenCase_XML_GUIDE_Nov2013.pdf*)
boundary particles will replace the fluid ones in the sides of the beach

To create *CaseWavemaker__Dp.vtk* and *CaseWavemaker__Real.vtk*

Piston movement:
Particles associated to mk=10 move following a sinusoidal movement
The movement is the combination of three different movements with different frequencies, amplitudes and phases
The duration and the order of the moves are also indicated

Parameters of configuration

```
<execution>
    <parameters>
        <parameter key="StepAlgorithm" value="2" comment="Step Algorithm 1:Verlet, 2:Symplectic (def=1)" />
        <parameter key="VerletSteps" value="40" comment="Verlet only: Number of steps to apply Eulerian equations (def=40)" />
        <parameter key="Kernel" value="2" comment="Interaction Kernel 1:Cubic Spline, 2:Wendland (def=1)" />
        <parameter key="ViscoTreatment" value="1" comment="Viscosity Formulation 1:Artificial, 2:Laminar+SPS (def=1)" />
        <parameter key="Visco" value="0.4" comment="Viscosity value" />
        <parameter key="ShepardSteps" value="0" comment="Number of steps to apply Shepard density filter, 0=non applied (def=0)" />
        <parameter key="DeltaSPH" value="0.1" comment="DeltaSPH value, 0.1 is the typical value, with 0 disabled (def=0)" />
        <parameter key="DtIni" value="0.0001" comment="Initial time step" />
        <parameter key="DtMin" value="0.00001" comment="Minimum time step (def=0.00001)" />
        <parameter key="TimeMax" value="10" comment="Time of simulation" />
        <parameter key="TimeOut" value="0.1" comment="Time between output files" />
        <parameter key="IncZ" value="0.5" comment="Increase of Z+" />
        <parameter key="PartsOutMax" value="1" comment="Allowed percentage of fluid particles out the domain (def=1)" />
        <parameter key="YPeriodicIncZ" value="0.0" comment="Increase of Z with periodic BC" />
    </parameters>
</execution>
```

8.1 DualSPHysics Pre-processing Interface (DPI)

Input XML files for GenCase can be created either through manual definition or by using the DualSPHysics Pre-processing Interface (DPI) software. The DPI is a Java based application designed to allow the most common parameters needed to define a DualSPHysics file to be set in a user-friendly manner. For cases involving complex functionality (such as motion), DPI can be used to first define geometrical properties and the resulting XML file manually edited.

DPI utilises OpenGL technology, combined with a Java interface, to allow the geometry and parameters of a DualSPHysics file to be built up and instantaneously visualised, this can make the process of creating new simulations more intuitive and less time-consuming. Previous GenCase XML documents can also be opened and edited by the DPI.

The DPI has the following features and requirements:

Features:

- Instantaneous visual feedback of the geometry of objects making up a DualSPHysics case.
- Textual *tool-tip* description of the meaning of each parameter.
- Defined limits of numerical parameters to ensure correct operation.
- Editing of existing GenCase XML files.

Requirements:

- Either Windows or Linux 64 bit operating system.
- A correctly defined Java runtime environment (paths set and version 1.6 or greater).
- (Within Linux) A functioning graphical user interface within the operating system.
- A functioning OpenGL driver installation.

For further details please see the available operating manual.

Please note: The DPI is open-source and is provided as a 64 bit compiled JAR executable and Oracle NetBeans project. It has been primarily designed to work under 64 bit operating systems and relies on native binaries for OpenGL and VTK operation. The Java interface is platform independent, therefore operation under an alternative OS such as MacOS is possible but will require modification of the package in order to provide appropriate binaries. Further details are provided in the operating manual and packaged files.

Future Developments: The DPI is currently designed to ease the process of defining a DualSPHysics case, however it is constantly undergoing development and has the following future aims:

- To act as a *hub* application for the individual executables that comprise the DualSPHysics package.
- Definition of parameters for pre-processing, processing and post-processing tools.
- The ability to launch executable directly from within the GUI.
- Offer instantaneous feedback by way of visualisation as a simulation evolves either locally or by connecting remotely to a job being processed on a HPC resource.
- Provide the ability to kill or pause a job.

9. Processing

The main code which performs the SPH simulation is named **DualSPHysics**.

The **input files** to run DualSPHysics code include one XML file (**Case.xml** in Figure 9-1) and a binary file (**Case.bi2** in Figure 9-1). **Case.xml** contains all the parameters of the system configuration and its execution such as key variables (smoothing length, reference density, gravity, coefficient to calculate pressure, speed of sound…), the number of particles in the system, movement definition of moving boundaries and properties of moving bodies. The binary file **Case.bi2** contains the particle data; arrays of position, velocity and density and headers. The **output files** consist of binary format files with the particle information at different instants of the simulation; Part0000.bi2, **Part0001.bi2, Part0002.bi2** …, **PartOut.bi2** with excluded particles and **Run.out** with a brief of the simulation (red boxes).

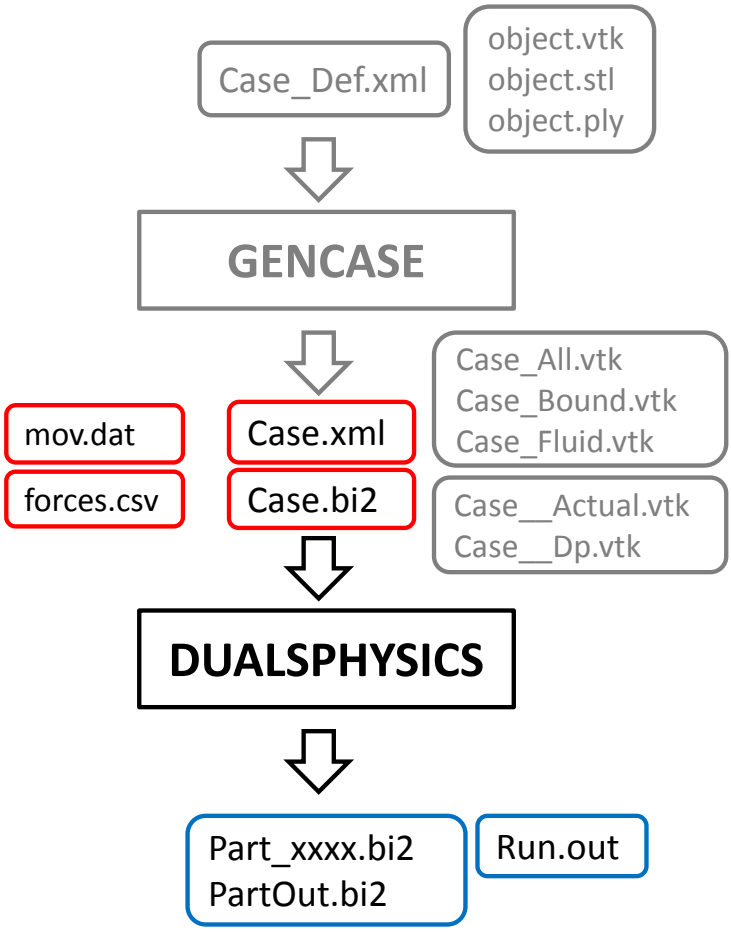


Figure 9-1. Input (red) and output (blue) files of DualSPHysics code.

Different execution parameters can be changed: time stepping algorithm specifying Symplectic or Verlet (**-symplectic, -verlet[:steps])**, choice of kernel function which can be Cubic or Wendland **(-cubic, -wendland)**, the value for artificial viscosity (**-viscoart:**) or laminar+SPS viscosity treatment (**-viscolamsps:**), activation of the Shepard density filter and how often it is applied (**-shepard:steps)**, the deltaSPH correction (**-deltasph:**), the maximum time of simulation and time intervals to save the output data (**-tmax:, -tout:>).** To run the code, it is also necessary to specify whether the simulation is going to run in CPU or GPU mode (**-cpu, -gpu[:id]**), the format of the output files (**-sv:[formats,...], none, binx2, ascii, vtk, csv**), that summarises the execution process (**-svres:<0/1**>) with the computational time of each individual process (**-svtimers:<0/1**>). It is also possible to exclude particles out of limits of some given values of density (**-rhopout:min:max**).

For CPU executions, a multi-core implementation using OpenMP enables executions in parallel using the different cores of the machine. It takes the maximum number of cores of the device by default or users can specify it (**-ompthreads:<int**>). In addition, the parallel execution can use dynamic (**-ompdynamic**) or static (**-ompstatic**) load balancing. On the other hand, different cell divisions of the domain can be used (**-cellmode:<mode**>) that differ in memory usage and efficiency.

TO RUN DUALSPHYSICS: DualSPHysics Case [options]

where *Case* is the name of the input files (Case.xml and Case.bi2 as seen in Figure 9-1). The configuration of the execution is mostly defined in the XML file, but it can be also defined or changed using **execution parameters**. Furthermore, new options and possibilities for the execution can be imposed using **[options]**:

-h
Shows information about parameters. Typing "DualSPHysics –h" in the command window generates a brief help manual (available in **HELP** folder).

-opt <file>
Loads configuration from a file.

-cpu
Execution on Cpu (option by default).

-gpu[:id]
Execution on Gpu and id of the device.

-stable
Ensures the same results when repeated a simulation since operations are always carried out in the same order.

-ompthreads:<int>
Only for Cpu. Indicates the number of threads by host for parallel execution, it takes the number of cores of the device by default (or using zero value).

-ompdynamic
Only for Cpu. Parallel execution with symmetry in interaction and dynamic load balancing. Not compatible with **–stable.**

-ompstatic
Only for Cpu. Parallel execution with symmetry in interaction and static load balancing.

-cellorder:<axis>
Indicates the order of the axis. (xyz/xzy/yxz/yzx/zxy/zyx).

-cellmode:<mode>
Specifies the cell division mode, by default, the fastest mode is chosen

- **h** fastest and the most expensive in memory
- **2h** lowest and the least expensive in memory

-symplectic
Symplectic algorithm as time step algorithm.

-verlet[:steps]
Verlet algorithm as time step algorithm and number of time steps to switch equations.

-cubic
Cubic spline kernel.

-wendland
Wendland kernel.

-viscoart:<float>
Artifitical viscosity [0-1].

-viscolamsps:<float>
Laminar+SPS viscosity [order of 1E-6].

-shepard:steps
Shepard filter and number of steps to be applied.

-deltasph:<float>
Constant for DeltaSPH. By default 0.1 and 0 to disable.

-sv:[formats,...]
Specifies the output formats:

- **none** No files with particle data are generated
- **binx** Bynary files (option by default)
- **vtk** VTK files
- **ascii** ASCII files (PART_xxxx of SPHysics)
- **csv** CSV files

-svres:<0/1>
Generates file that summarizes the execution process.

-svtimers:<0/1>
Obtains timing for each individual process.

-svdomainvtk:<0/1>
Generates VTK file with domain limits.

-name <string>
Specifies path and name of the case.

-runname <string>
Specifies name for case execution.

-dirout <dir>
Specifies the output directory.

-partbegin:begin[:first] dir
RESTART option. Specifies the beginning of the simulation starting from a given PART (begin) and located in the directory (dir), (first) indicates the number of the first PART to be generated.

-incz:<float>
Allowable increase in Z+ direction. Case domain is fixed as function of the initial particles, however the maximum Z position can be increased with this option in case particles reach higher positions.

-rhopout:min:max
Excludes fluid particles out of these density limits.

-ftpause:<float>
Time to start floating bodies movement. By default 0.

-tmax:<float>
Maximum time of simulation.

-tout:<float>
Time between output files.

-domain_particles[:xmin,ymin,zmin,xmax,ymax,zmax]
Case domain is fixed as function of the initial particles and is modified by xmin,...

-domain_particles_prc:xmin,ymin,zmin,xmax,ymax,zmax
Case domain is fixed with values in proportion to the dimensions of the case according to the initial particles.

-domain_fixed:xmin,ymin,zmin,xmax,ymax,zmax
Case domain is fixed with the given values.

-ptxasfile <file>
Indicates the file with information about the compilation kernels in CUDA to adjust the size of the blocks depending on the needed registers for each kernel (only for gpu). By default, it takes the path and the name of the executable + _ptxasinfo,

10.Post-processing

10.1 Visualization of particle output data

The **PartVTK code** is used to convert the output binary files of DualSPHysics into different formats that can be visualized and /or analysed. Thus, the output files of DualSPHysics, the binary files (.bi2), are now the **input files** for the post-processing code **PartVTK**.

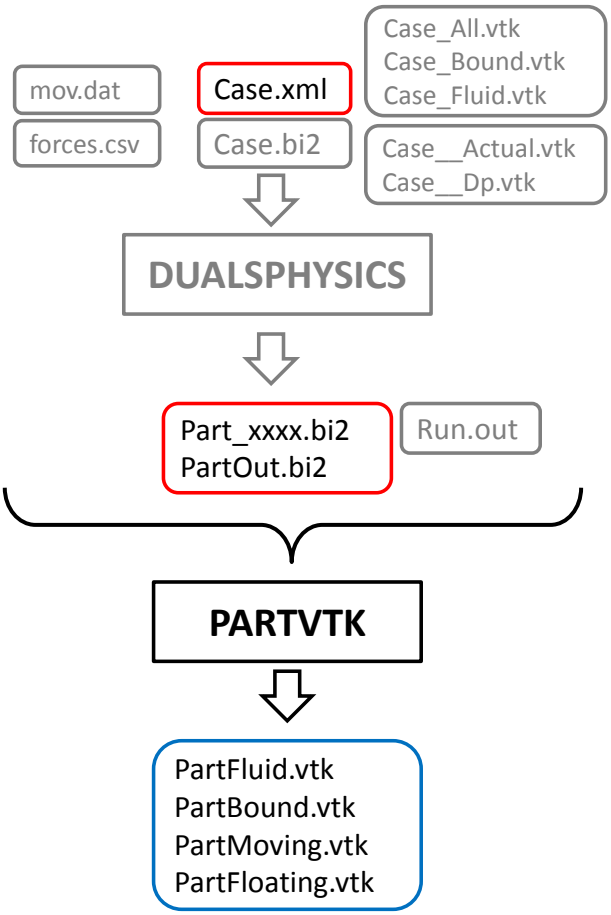


Figure 10-1. Input (red) and output (blue) files of PartVTK code.

The **output files** can be VTK-binary (**-savevtk**), CSV (**-savecsv**) or ASCII (**-saveascii**). In this way the results of the simulation can be plotted using Paraview gnuplot Octave etc…. For example; PartVtkBin_0000.vtk,... These files can be generated selecting a set of particles defined by **mk** (**-onlymk:**), by the **id** of the particles (**-onlyid:**) or by the type of the particle (**-onlytype:**) so we can check or uncheck all the particles (**+/-all**), the boundaries (**+/-bound**), the fixed boundaries (**+/-fixed**), the moving boundaries (**+/-moving**), the floating bodies (**+/-floating**), the fluid particles (**+/-fluid**) or the excluded fluid particles during the simulation (**+/-fluidout**). The output files can contain different particle data (**-vars:**); all the physical quantities (**+/-all**), velocity (**+/-vel**), density (**+/-rhop**), pressure (**+/-press**), mass (**+/-mass**), acceleration (**+/-ace**), vorticity (**+/-vor**), the id of the particle (**+/-id**),

the type (**+/-type:**) and mk (**+/-mk**). Acceleration values can be positive (AcePos) or negative (AceNeg) according to the direction of the axis.

TO RUN PARTVTK:
PartVTK -savevtk PartFluid.vtk -onlytype:-fluid [options]

Basic options:

-h
Shows information about parameters. Typing "PartVTK –h" in the command window generates a brief help manual (available in **HELP** folder).

-opt <file>
Loads configuration from a file.

Define input file:

-dirin <dir>
Indicates the directory with particle data.

-filein <file>
Loads file with particle data.

-filexml file.xml
Loads xml file with information of mk and type of particles, this is needed for the filter -onlymk and for the variable -vars:mk.

-filemove <file>
Loads file with particle displacement.

-first:<int>
Indicates the first file to be computed.

-last:<int>
Indicates the last file to be computed.

-jump:<int>
Indicates the number of files to be skipped.

-threads:<int>
Indicates the number of threads for parallel execution of the interpolation, it takes the number of cores of the device by default (or using zero value).

Define parameters for acceleration or vorticity calculation:

-acekercubic
Use of Cubic spline kernel.

-acekerwendland
Use of Wendland kernel.

-aceviscoart:<float>
Use of Artifitical viscosity [0-1] (only for acceleration).

-acegravity:<float:float:float>
Gravity value (only for acceleration).

Define output file:

-savevtk <file.vtk>
Generates vtk(polydata) files with particles according to given filters with the options onlymk, onlyid and onlytype.

-savecsv <file.csv>
Generates CSV files to use with spreadsheets (e.g. Excel).

-saveascii <file.asc>
Generates ASCII files without headers.

Configuration for each output file:

-onlypos:xmin:ymin:zmin:xmax:ymax:zmax
Indicates limits of particles.

-onlymk:<values>
Indicates the mk of selected particles.

-onlyid:<values>
Indicates the id of selected particles.

-onlytype:<values>
Indicates the type of selected particles:

- **+/-all:** To choose or reject all options
- **+/-bound:** Boundary particles (fixed, moving and floating)
- **+/-fixed:** Boundary fixed particles
- **+/-moving:** Boundary moving particles
- **+/-floating:** Floating body particles
- **+/-fluid:** Fluid particles (no excluded)
- **+/-fluidout:** Excluded fluid particles

(by default: all)

-vars:<values>
Indicates the stored variables of each particle:

- **+/-all:** To choose or reject all options
- **+/-vel:** Velocity
- **+/-rhop:** Density
- **+/-press:** Pressure
- **+/-mass:** Mass
- **+/-id:** Id of particle
- **+/-type:** Type (fixed,moving,floating,fluid,fluidout)
- **+/-mk:** Value of mk associated to the particles
- **+/-ace:** Acceleration
- **+/-vor:** Vorticity

(by default: id,vel,rhop,type)

10.2 Analysis of numerical measurements

To compare experimental and numerical values, a tool to analyse these numerical measurements is needed. The **MeasureTool** code allows different physical quantities at a set of given points to be computed. The binary files (.bi2) generated by DualSPHysics are the **input files** of the **MeasureTool** code and the **output files** are again VTK-binary or CSV or ASCII. The numerical values are computed by means of an SPH interpolation of the values of the neighbouring particles around a given position.

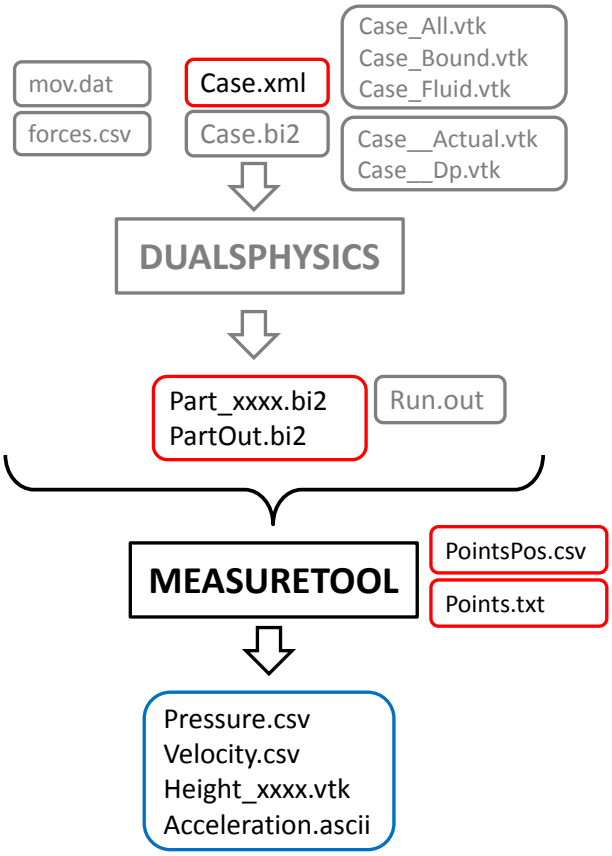


Figure 10-2. Input (red) and output (blue) files of MeasureTool code.

The interpolation can be computed using different kernels (**-kercubic**, **-kerwendland**). Kernel correction is also applied when the summation of the kernel values around the position is higher than a value (**-kclimit:**) defining a dummy value if the correction is not applied (**-kcdummy:**). The positions where the interpolation is performed are given in a text file (**-points <file>**) and the distance of interpolation can be 2*h* (the size of the kernel) or can be changed (**-distinter_2h:**, **-distinter:**). The computation can only be carried out by a selected set of particles, so the same commands for PartVTK can be used (**-onlymk:**, **-onlyid:**, **-onlytype:**). Different interpolated variables (**-vars**) can be calculated numerically; all available ones (**+/-all**), velocity (**+/-vel**), density (**+/-rhop**), pressure (**+/-press**), mass (**+/-mass**), vorticity (**+/-vor**), acceleration (**+/-ace**), the id of the particle (**+/-id**), the type (**+/-type:**) and mk (**+/-mk**). The maximum water depth can be also computed. Height values (**-height:**) are calculated according to the interpolated mass, if the nodal mass is higher

than a given reference mass, that Z-position will be considered as the maximum height. The reference value can be calculated in relation to the mass values of the selected particles (**-height:0.5**, half the mass by default in 3D and **-height:0.4** in 2D) or can be given in an absolute way (**-heightlimit:**).

TO RUN MEASURETOOL:
MeasureTool -points points.txt -vars:press -savecsv Press [options]

Basic options:

-h
Shows information about parameters. Typing "MeasureTool –h" in the command window generates a brief help manual (available in **HELP** folder).

-opt <file>
Loads configuration from a file.

Define input file:

-dirin <dir>
Indicates the directory with particle data.

-filein <file>
Loads file with particle data.

-filexml file.xml
Loads xml file with information of mk and type of particles, this is needed for the filter –onlymk and for the variable -vars:mk

-first:<int>
Indicates the first file to be computed.

-last:<int>
Indicates the last file to be computed.

-jump:<int>
Indicates the number of files to be skipped.

-threads:<int>
Indicates the number of threads for parallel execution of the interpolation, it takes the number of cores of the device by default (or using zero value).

Define parameters for acceleration or vorticity calculation:

-acekercubic
Use of Cubic spline kernel.

-acekerwendland
Use of Wendland kernel.

-aceviscoart:<float>
Use of Artifitical viscosity [0-1].

-acegravity:<float:float:float>
Gravity value.

Set the filters that are applied to particles:

-onlymk:<values>
Indicates the mk of selected particles for interpolation.

-onlyid:<values>
Indicates the id of selected particles.

-onlytype:<values>
Indicates the type of selected particles:

+/-all: To choose or reject all options
+/-bound: Boundary particles (fixed, moving and floating)
+/-fixed: Boundary fixed particles
+/-moving: Boundary moving particles
+/-floating: Floating body particles
+/-fluid: Fluid particles (no excluded)
+/-fluidout: Excluded fluid particles
(by default: all)

Set the configuration of interpolation:

-pointstemplate
Creates example file to be used with -points or –pointspos.

-points <file>
Defines the points where interpolated data will be computed (each value separated by an space or a new line).

-pointspos <file>
Defines the position of points where interpolated data will be computed (each line contains the points for one part file, points are defined as x0;y0;z0;x1;y1;z1...).

-particlesmk:<values>
Indicates the points where interpolated data will be computed using the positions of the particles with the given mk.

-kercubic
Uses Cubic spline kernel for interpolation.

-kerwendland
Uses Wendland kernel for interpolation (by default).

-kclimit:<float>
Defines the minimum value of sum_wab_vol to apply the Kernel Correction (0.5 by default).

-kcdummy:<float>
Defines the dummy for the interpolated quantity if Kernel Correction is not applied (0 by default).

-kcusedummy:<0/1>
Defines whether or not to use the dummy value (1 by default).

-distinter_2h:<float>
Defines the maximum distance for the interaction among particles depending on 2h (1 by default).

-distinter:<float>
Defines the maximum distance for the interaction among particles in an absolute way.

Set the values to be calculated:

-vars:<values>
Defines the variables or magnitudes that are going to be computed as an interpolation of the selected particles around a given position:

- **+/-all:** To choose or reject all options
- **+/-vel:** Velocity
- **+/-rhop:** Density
- **+/-press:** Pressure
- **+/-mass:** Mass
- **+/-id:** Id of particle
- **+/-type:** Type (fixed,moving,floating,fluid,fluidout)
- **+/-mk:** Value of mk associated to the particles
- **+/-ace:** Acceleration
- **+/-vor:** Vorticity

(by default: id,vel,rhop,type)

-height[:<float>]
Height value is calculated starting from mass values for each point x,y. The reference mass to obtain the height is calculated according to the mass values of the selected particles; by default 0.5 in 3D (half the mass) and 0.4 in 2D.

-heightlimit:<float>
The same than -height but the reference mass is given in an absolute way

Define output files format:

-savevtk <file.vtk>
Generates VTK (polydata) with the given interpolation points.

-savecsv <file.csv>
Generates one CSV file with the time history of the obtained values.

-saveascii <file.asc>
Generates one ASCII file without headers with the time history of the obtained values.

10.3 Visualization of boundaries

In order to visualise the boundary shapes formed by the boundary particles, different geometry files can be generated using the **BoundaryVTK code**. The code creates triangles or planes to represent the boundaries.

As **input data**, shapes can be loaded from a VTK file (**-loadvtk**), a PLY file (**-loadply**) or an STL file (**-loadstl**) while boundary movement can be imported from an XML file (**-loadxml file.xml**) using the timing of the simulation (**-motiontime**) or with the exact instants of output data (**-motiondatatime**). The movement of the boundaries can also be determined starting from the particle positions (**-motiondata**). The **output files** consist of VTK files (**-savevtk**), PLY files (**-saveply**) or STL files (**-savestl**) with the loaded information and the moving boundary positions at different instants. For example the output files can be named motion_0000.vtk, motion_0001.vtk, motion_0002.vtk... These files can be also generated by only a selected object defined by **mk** (**-onlymk:**), by the **id** of the particles (**-onlyid:**).

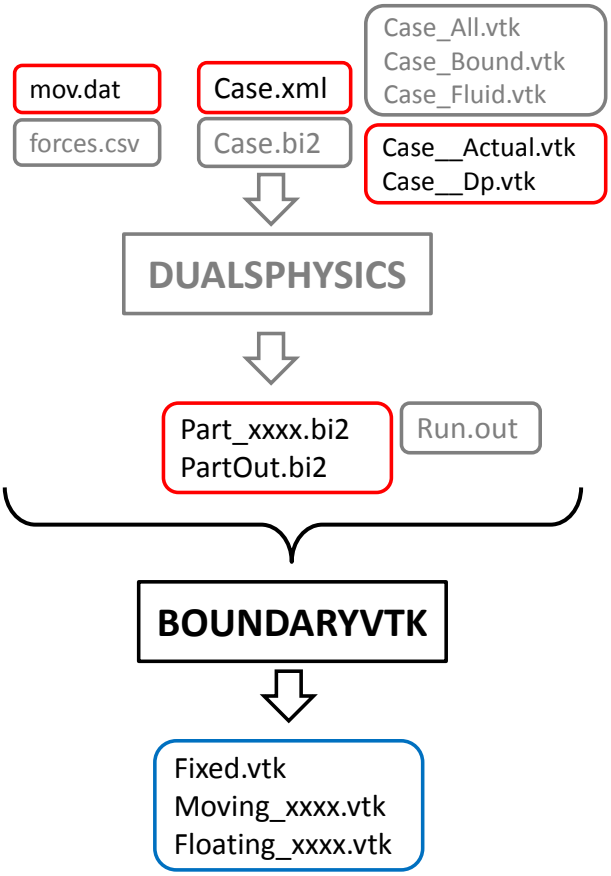


Figure 10-3. Input (red) and output (blue) files of BoundaryVTK code.

TO RUN BOUNDARYVTK:
BoundaryVTK -loadvtk bound.vtk -savevtk box.vtk -onlymk:10 [options]

Basic options:

-h
Shows information about parameters. Typing "BoundaryVTK –h" in the command window generates a brief help manual (available in **HELP** folder).

-opt <file>
Loads configuration from a file.

-info
Shows information of loaded data.

Load shapes:

-loadvtk <file.vtk>
Load shapes from vtk files (PolyData).

-onlymk:<values>
Indicates the mk of the shapes to be loaded, affects to the previous -loadvtk and cannot be used with –onlyid.

-onlyid:<values>
Indicates the code of object of the shapes to be loaded, only affects to the previous –loadvtk.

-changemk:<value>
Changes the mk of the loaded shapes to a given value, only affects to the previous –loadvtk.

-loadply:<mk> <file.ply>
Load shapes from ply files with a given mk.

-loadstl:<mk> <file.stl>
Load shapes from stl files with a given mk.

Load configuration for a mobile boundary:

-filexml file.xml
Loads xml file with information of the movement and kind of particles.

-motiontime:<time>:<step>
Configures the duration and time step to simulate the movement defined on the xml file, cannot be used with -motiondatatime and –motiondata.

-motiondatatime <dir>
Indicates the directory where real times of the simulation are loaded for the mobile boundary cannot be used with -motiontime and –motiondata.

-motiondata <dir>
Indicates the directory where position of particles are loaded to generate the mobile boundary cannot be used with -motiontime and –motiondatatime.

Define output files:

-savevtk <file.vtk>
Generates vtk(polydata) files with the loaded shapes.

-saveply <file.ply>
Generates ply file with the loaded information.

-savestl <file.stl>
Generates stl file with the loaded information.

-savevtkdata <file.vtk>
Generates vtk(polydata) file with the loaded shapes including mk and shape code.

-onlymk:<values>
Indicates the *mk* value of the shapes to be stored, affects to the previous out option and cannot be used with –onlyid.

-onlyid:<values>
Indicates the code of the object of the shapes to be stored, affects to the previous out option.

-filemove <file>
Loads file with displacement.

10.4 Surface representation

Using a large number of particles, the visualization of the simulation can be improved by representing surfaces instead of particles. To create the surfaces, the *marching cubes* algorithm is used [Lorensen and Cline, 1987]. This computer graphics technique extracts a polygonal mesh (set of triangles) of an isosurface from a 3-D scalar field.

Figure 10-4, represents a 3D dam-break simulation using 300,000 particles. The first snapshot shows the particle representation. Values of mass are interpolated at the nodes of a 3-D Cartesian mesh that covers the entire domain using an SPH interpolation. Thus a 3-D mesh vertex that belongs to the free surface can be identified. The triangles of this surface (generated by means of the marching cubes algorithm) are represented in the second frame of the figure. The last snapshots correspond to the surface representation where the colour corresponds to the interpolated velocity value of the triangles.

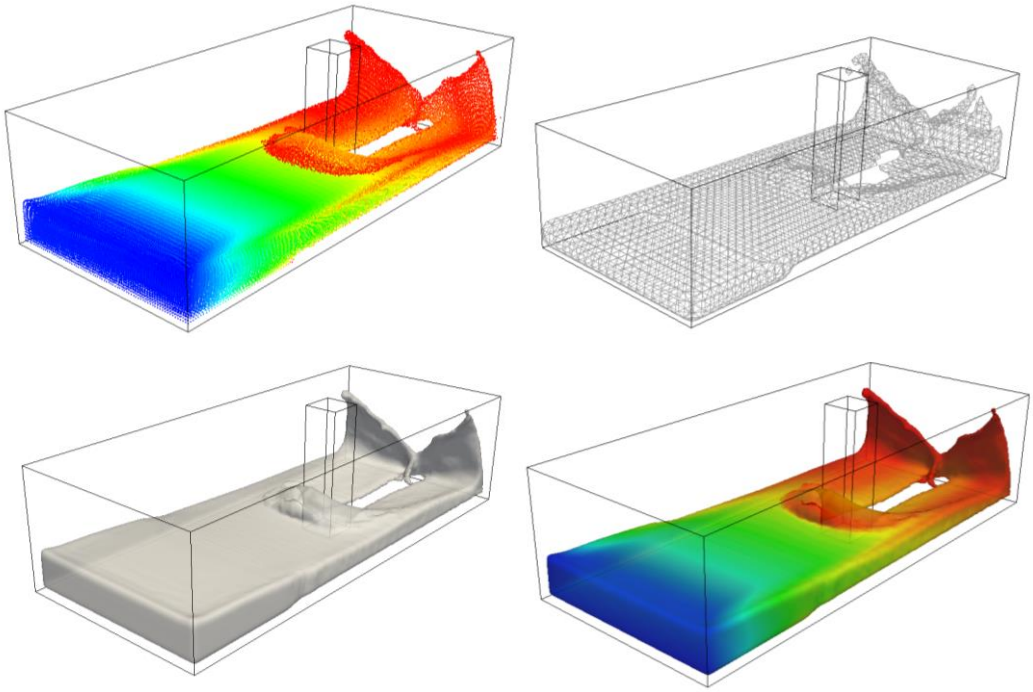


Figure 10-4. Conversion of points to surfaces.

The output binary files of DualSPHysics are the **input files** of the IsoSurface code and the **output files** are VTK files (**-saveiso[:<var>]**) with the isosurfaces calculated using a variable (**<var>**) or can be structured points (**-savegrid**) with data obtained after the interpolation. The Cartesian mesh size can be defined by specifying the discretisation size using either the particle size dp (**-distnode_dp:**) or by specifying an absolute internode distance (**-distnode:**). On the other hand, the maximum distance for the interaction between particles to interpolate values on the nodes can be also defined depending on 2h (**-distinter_dp:**) or in an absolute way (**-distinter:**). The limits of mesh can be calculated starting from particle data of all part files (**-domain_static**) or adjusted to the selected particles of each part file (**-domain_dynamic**).

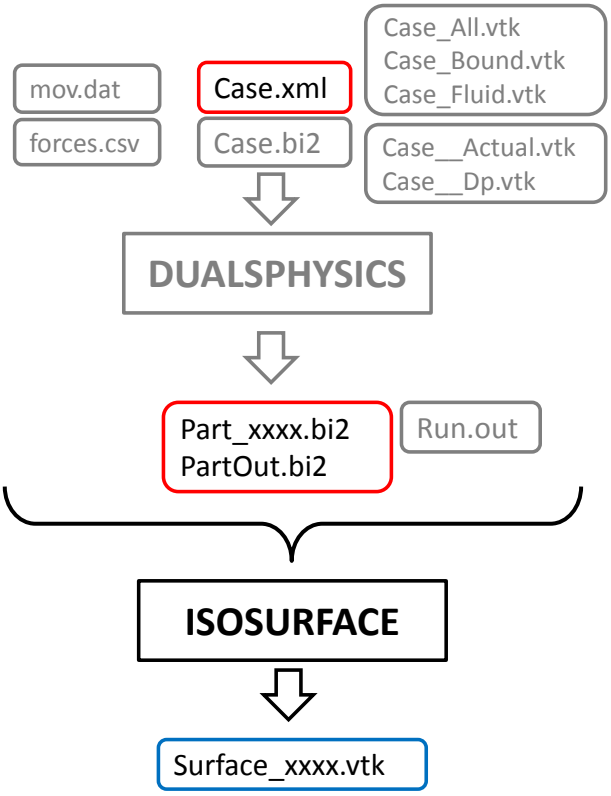


Figure 10-5. Input (red) and output (blue) files of IsoSurface code.

TO RUN ISOSURFACE:
IsoSurface -saveiso:0.04,0.045,0.05 fileiso [options]

Basic options:

-h
Shows information about parameters. Typing "IsoSurface –h" in the command window generates a brief help manual (available in **HELP** folder).

-opt <file>
Loads configuration from a file.

Define input files:

-dirin <dir>
Indicates the directory with particle data.

-filein <file>
Loads file with particle data.

-first:<int>
Indicates the first file to be computed.

-last:<int>
Indicates the last file to be computed.

Set the configuration of interpolation:

-domain_dynamic
Mesh limits are adjusted to the selected particles of each part file (option by default).

-domain_limits:xmin:ymin:zmin:xmax:ymax:zmax
Mesh limits are adjusted to the selected particles within the given limits.

-iso_limits:xmin:ymin:zmin:xmax:ymax:zmax
Isosurface limits are adjusted to the given limits.

-distinter_2h:<float>
Defines the maximum distance for the interaction between particles depending on 2*h* (1 by default).

-distinter:<float>
Defines the maximum distance for the interaction between particles in an absolute way.

-distnode_dp:<float>
Defines the distance between nodes by multiplying dp and the given value (option by default).

-distnode:<float>
Defines the distance between nodes.

-threads:<int>
Indicates the number of threads for parallel execution of the interpolation, it takes the number of cores of the device by default (or using zero value).

Define output files format:

-saveiso[:<values>] <file.vtk>
Generates vtk files (polydata) with the isosurface calculated starting from a variable mass and using several limit values. The variable by default is mass. When limit values are not given the intermediate mass value of the fluid is considered.

11. Testcases

Some demonstration cases are included in the DualSPHysics package (Figure 11.1); such as a dam break interacting with an obstacle in a tank, waves in a beach created by a piston or the real geometry of a pump mechanism that translates water within a storage tank.

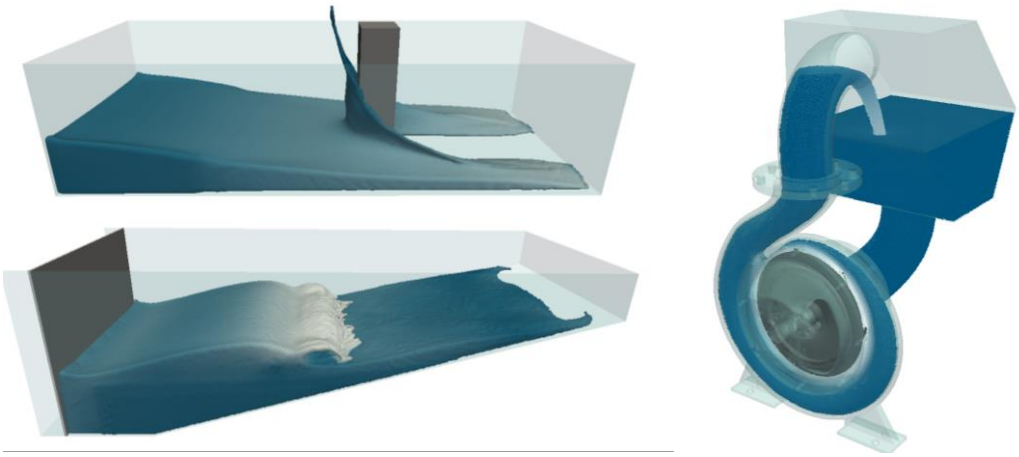


Figure 11-1. Test cases: dam break, wave maker, and pump (d).

The different test cases are described in the following sections. The scripts (*.bat* in Windows and *.sh* in Linux) are presented and the different command lines to execute the different programs are described. The XML files are also explained in detail to address the execution parameters and the different labels to plot boundaries, fluid volumes, movements, etc.

For each case there are 4 options;
Case_win_CPU.bat (windows on CPU)
Case_win_GPU.bat (windows on GPU)
Case_linux_CPU.sh (linux on CPU)
Case_linux_GPU.sh (linux on GPU)

Since many 3-D applications can be reduced to a 2-D problem, the executable of DualSPHysics can also perform 2-D simulations, however this option is not fully optimized. A 2-D simulation can be achieved easily by imposing the same values in the Y-direction (**<pointmin>** = **<pointmax>** in the input XML file). Thus, the initial configuration is a 2-D system and DualSPHysics will be automatically detected that the simulation must be carried out using the 2-D formulation.

11.1 CASEDAMBREAK

The first test case consists of a dam-break flow impacting on a structure inside a numerical tank. No moving boundary particles are involved in this simulation and no movement is defined. *CaseDambreak.bat* (*CaseDambreak.sh* in linux) file summarises the tasks to be carried out using the different codes in a given order:

```
CaseDambreak.bat

GenCase.exe CaseDambreak_Def CaseDambreak_out/CaseDambreak

DualSPHysics.exe CaseDambreak_out/CaseDambreak -svres -gpu

PartVTK.exe    -dirin CaseDambreak_out
       -savevtk CaseDambreak_out/PartFluid -onlytype:-all,+fluid

MeasureTool.exe -dirin CaseDambreak_out
      -points PointsVelocity.txt
       -onlytype:-all,+fluid -vars:-all,+vel
       -savevtk CaseDambreak_out/PointsVelocity
       -savecsv CaseDambreak_out/PointsVelocity
```

Generate Geometry for CaseDamBreak using GenCase

Run DualSPHysics on GPU

Postprocess output data for Paraview

Extract velocity data using MeasureTool

Different instants of the CaseDambreak simulation are shown in Figure 11-2 where the *PartFluid_xxxx.vtk* of the fluid particles, the *CaseDambreak_Box_Dp.vtk* and *CaseDambreak_Building_Dp.vtk* are depicted.

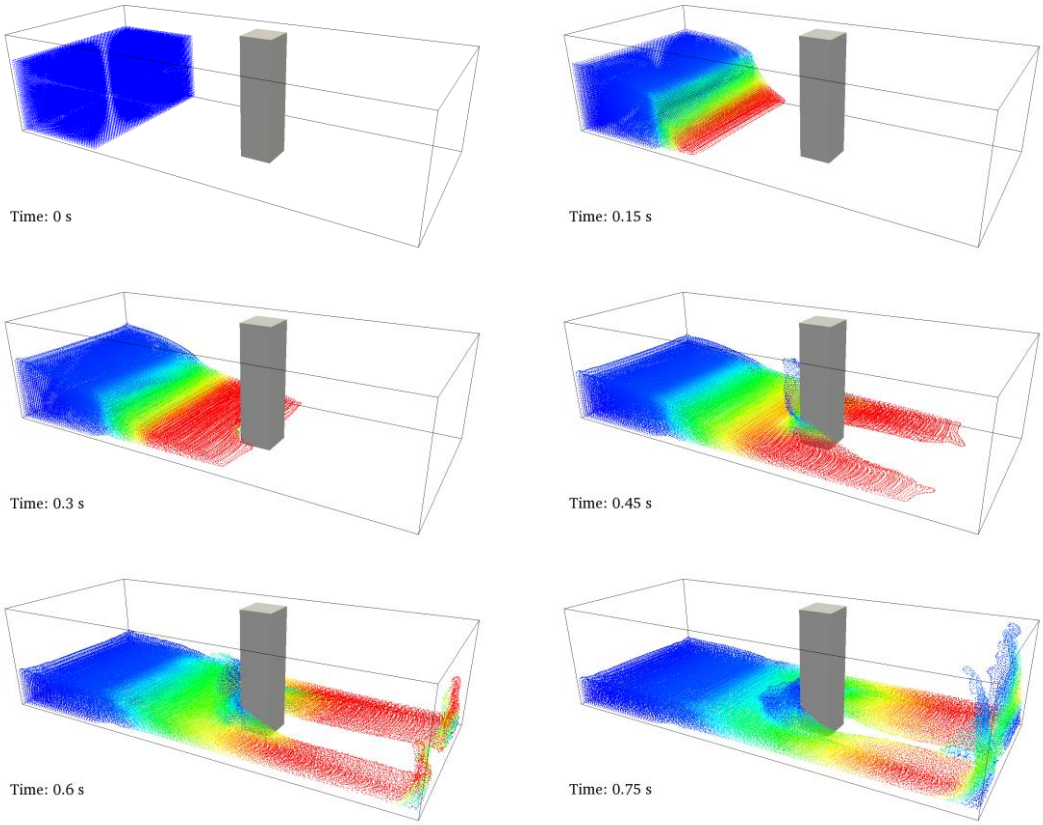


Figure 11-2. Instants of the CaseDambreak simulation.
Colour represents velocity of the particles.

Using MeasureTool code, numerical velocities have been computed in the list of points described in the file *PointsVelocity.txt*, which are (0.754,0.31,0.02) , (0.754,0.31,0.04) and (0.754,0.31,0.06). Numerical values of *u*-velocity in the *x*-direction are depicted against time in Figure 11-3 for these three points.

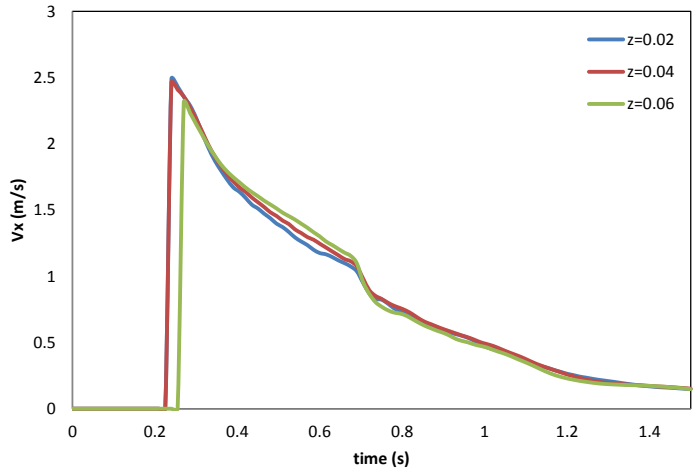


Figure 11-3. Numerical velocity computed at three different locations.

In addition, a validation in 2D is also included for a different dam break (Figure 11-4) where the experimental data of Koshizuka and Oka, 1996 can be reproduced numerically. Please note that, in this test case there are no obstacles. The file *EXP_X-DamTipPosition_Koshizula&Oka1996.txt* includes the X-position of the tip of the dam for several instants (before the impact with the other wall).

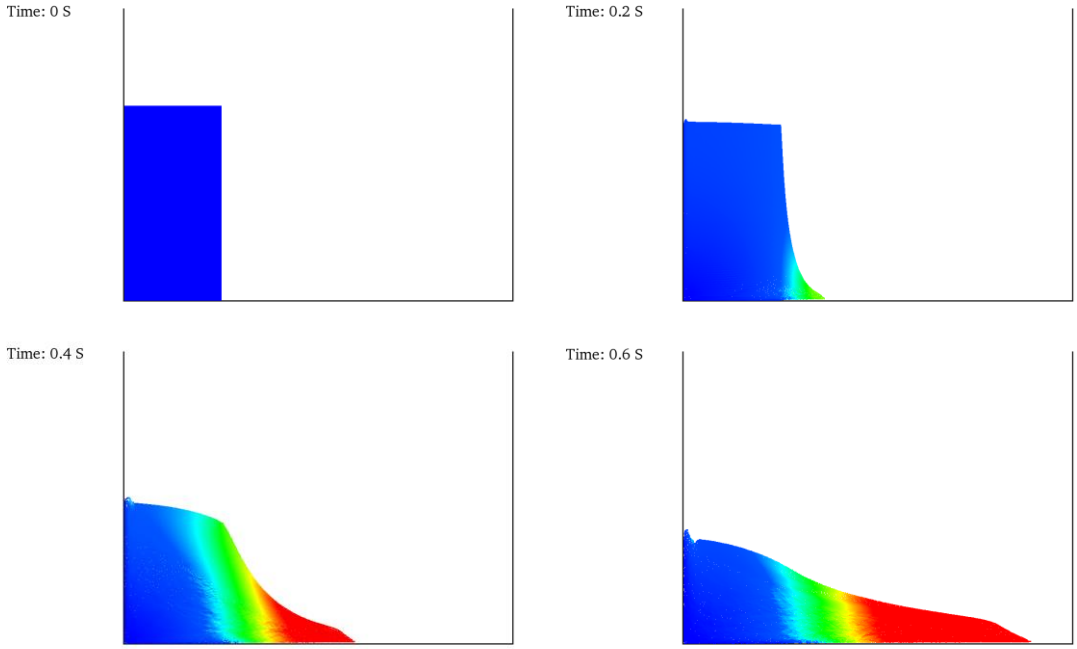


Figure 11-4. Instants of the CaseDambreakVal2D simulation.
Colour represents velocity of the particles.

11.2 CASEFLOATING

In this case, a floating box moves due to waves generated with a piston. *CaseFloating.bat* (*CaseFloating.sh* in linux) file summarises the tasks to be carried out using the different codes:

```
CaseFloating.bat

GenCase.exe CaseFloating_Def CaseFloating_out/CaseFloating

DualSPHysics.exe CaseFloating_out/CaseFloating -svres -gpu

PartVTK.exe   -dirin CaseFloating_out
              -savevtk CaseFloating_out/PartFluid -onlytype:-all,+fluid
              -savevtk CaseFloating_out/PartFloating -onlytype:-all,+floating

BoundaryVTK.exe -loadvtk CaseFloating_out/CaseFloating__Actual.vtk
              -filexml CaseFloating_out/CaseFloating.xml
              -motiondata CaseFloating_out
              -savevtkdata CaseFloating_out/Box.vtk -onlymk:31
              -savevtkdata CaseFloating_out/MotionFloating -onlymk:61
              -savevtkdata CaseFloating_out/MotionPiston -onlymk:21

IsoSurface.exe -dirin CaseFloating_out
              -saveiso CaseFloating_out/Surface -onlytype:-all,+fluid
```

Figure 11-5 shows different instants of the simulation of this test case. The floating object is represented using VTK files generated using the BAT file provided that can help users for their own post-processing:

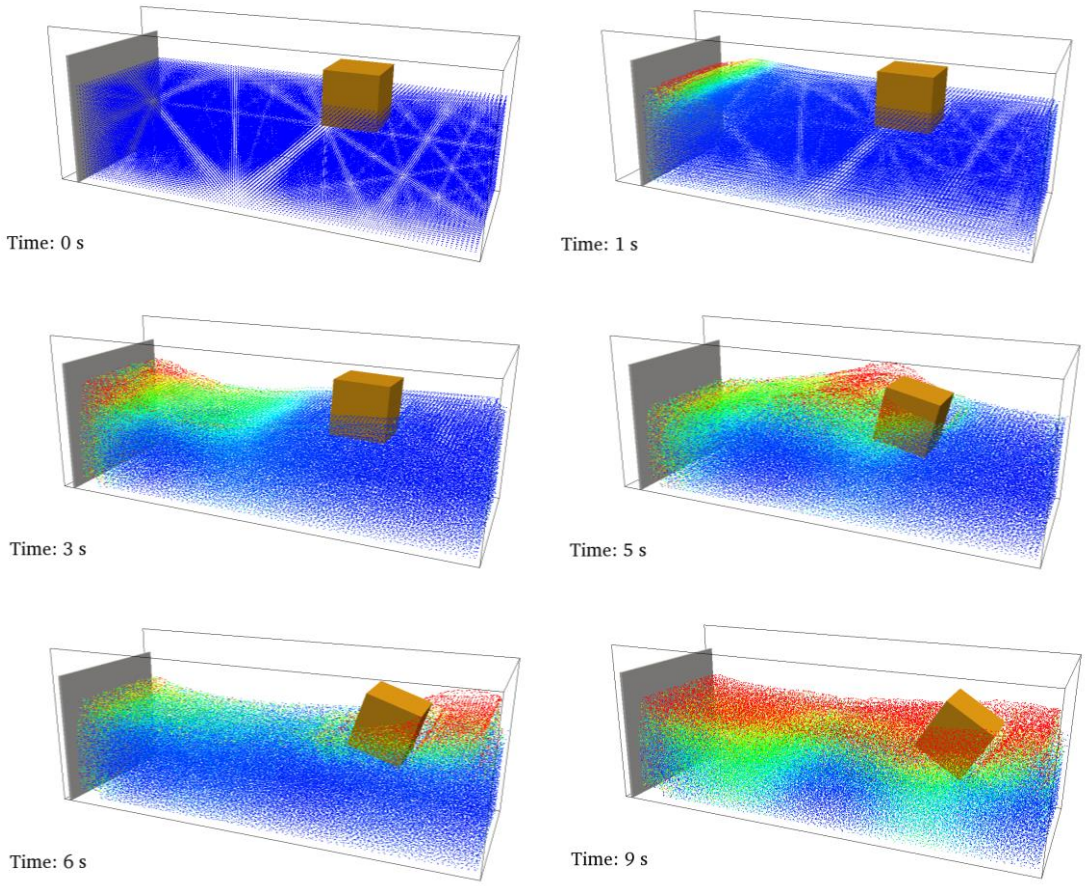


Figure 11-5. Different instants of the CaseFloating simulation.
Colour represents velocity of the particles.

The 2-D case of fluid-structure interaction is also provided (Figure 11-6). The text files included in the folder contains the experimental displacement and velocity of experiments in Fekken, 2004 and Moyo and Greenhow, 2000.

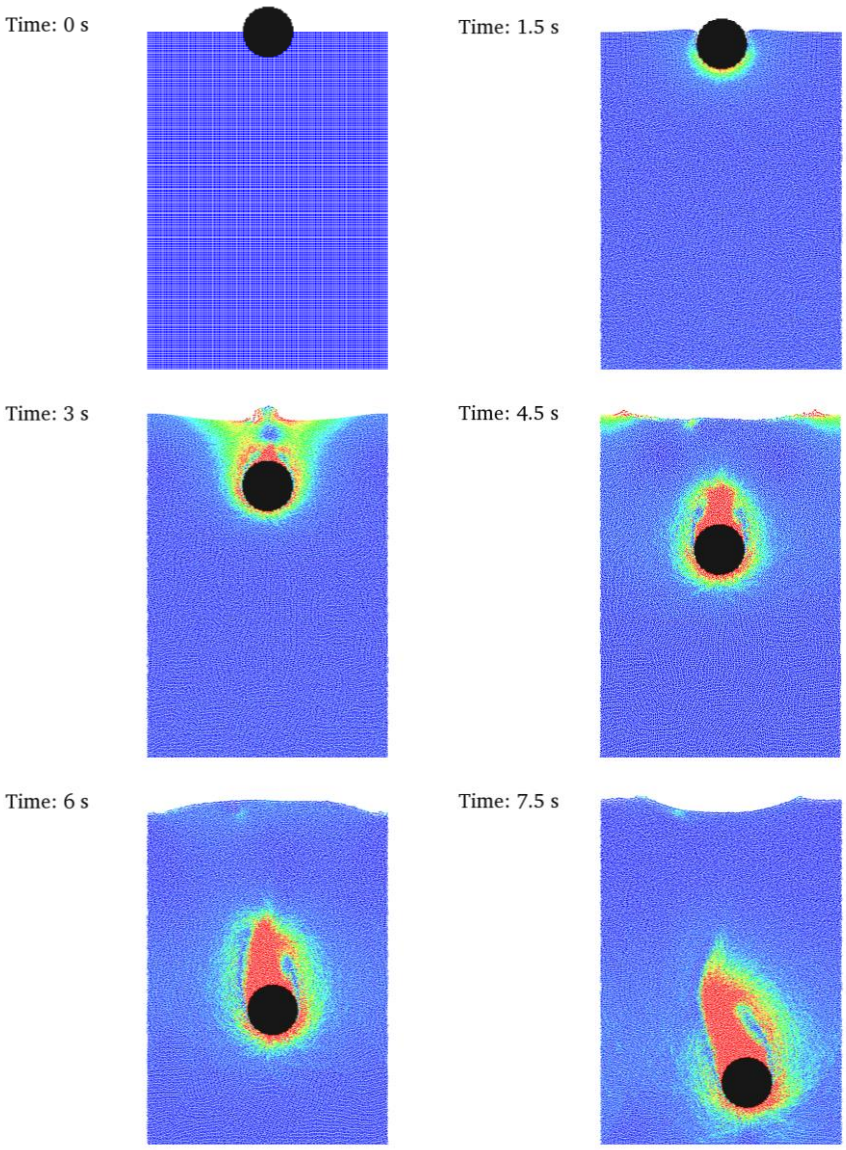


Figure 11-6. Different instants of the CaseFloatingVal2D simulation. Colour represents velocity of the particles.

11.3 CASEFORCES

This is a new testcase where external forces are applied to the system. The external forces can be loaded from the files *varAccInputFile_0.csv* and *varAccInputFile_1.csv*. *CaseForces.bat* (*CaseForces.sh* in linux) file summarises the tasks to be carried out using the different codes:

```
CaseForces.bat

GenCase.exe CaseForces_Def CaseForces_out/CaseForces

DualSPHysics.exe CaseForces_out/CaseForces -svres -gpu

PartVTK.exe  -dirin CaseForces_out
             -filexml CaseForces_out/CaseForces_out.xml
             -savevtk CaseForces_out/PartFluid
             -onlytype:-all,+fluid -vars:+mk,+press
```

Figure 11-7 shows the external forces that will be applied to all the fluid particles inside the numerical tank. This mimics the effect of a box being accelerated in one direction, or rotating around the y-axis.

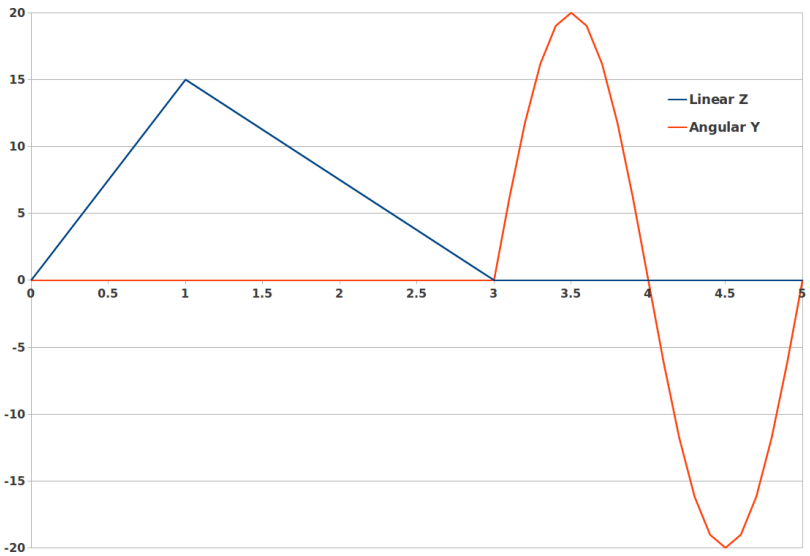


Figure 11-7. External forces applied to the fluid particles in CaseForces.

Different instants of the simulation can be observed in Figure 11-8 where the colours of the fluid particles correspond to the two different MK values.

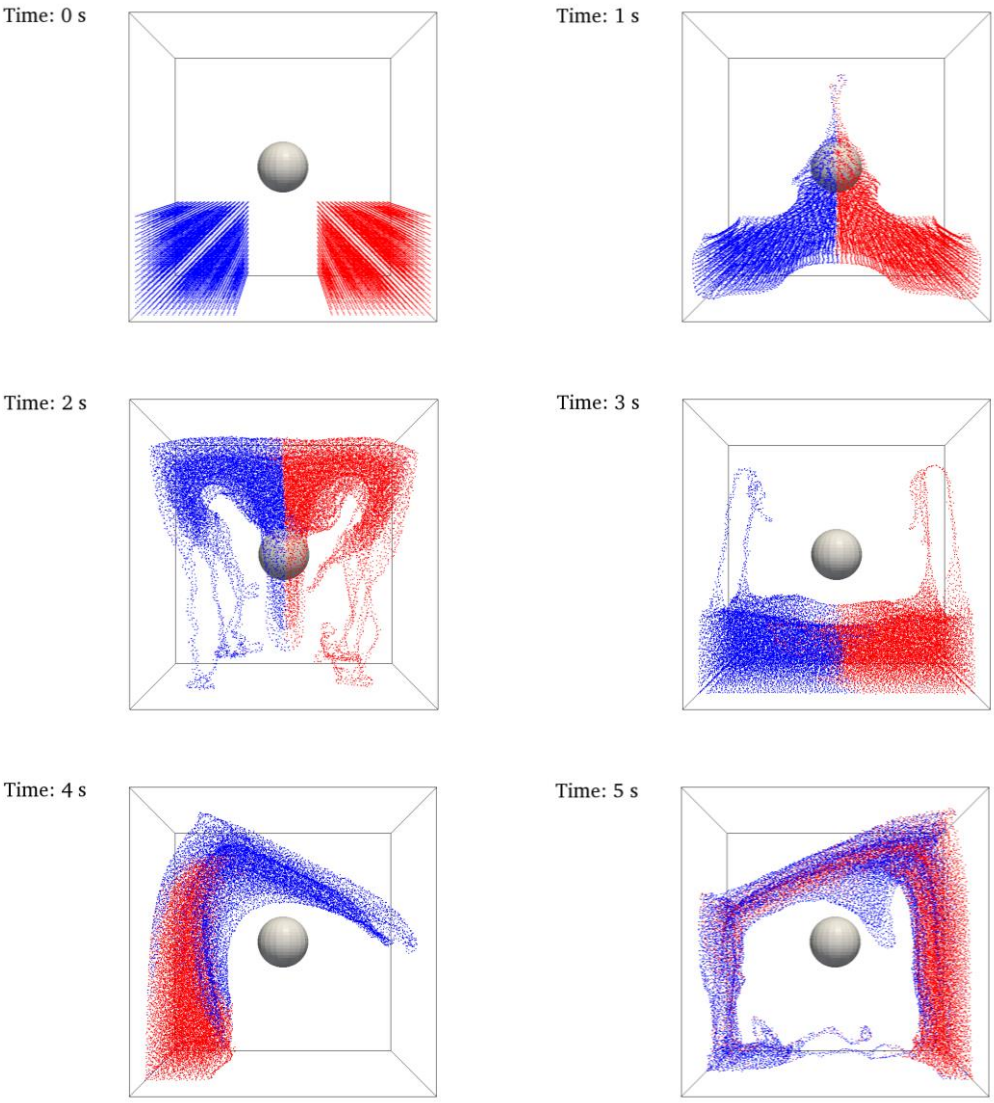


Figure 11-8. Different instants of the CaseForces simulation.
Colour represents MK value of the fluid particles.

11.4 CASEPERIODICITY

Periodic boundary conditions are now implemented in DualSPHysics v3.0. This testcase is an example of periodicity applied to 2D case where particles that leave the domain through the right wall are introduced through the left wall with the same properties but where the vertical position can change (with an increase of +0.3 in Z-position).
CasePeriodicity.bat (*CasePeriodicity.sh* in linux) file summarises the tasks to be carried out using the different codes:

```
CasePeriodicity.bat

GenCase.exe CasePeriodicity_Def CasePeriodicity_out/CasePeriodicity

DualSPHysics.exe CasePeriodicity_out/CasePeriodicity -svres -gpu

PartVTK.exe  -dirin CasePeriodicity_out
             -savevtk CaseForces_out/PartFluid
             -onlytype:-all,+fluid -vars:-all,+vel,+rhop,+press,+vor
```

Different instants of the CasePeriodicity simulation are shown in Figure 11-9 where the *PartFluid_xxxx.vtk* of the fluid particles and the *CasePeriodicity_Bound.vtk* are depicted.

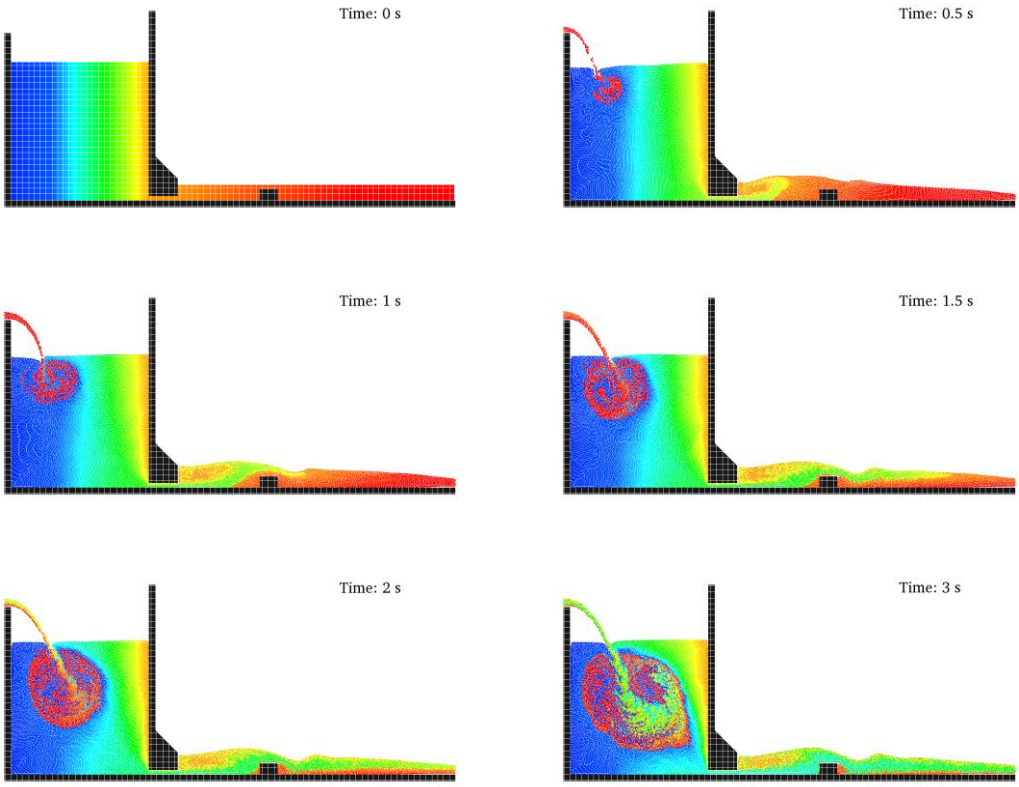


Figure 11-9. Different instants of the CasePeriodicity simulation.
Colour represents Id of the fluid particles.

11.5 CASEPUMP

This 3D testcase loads an external model of a pump with a fixed (*pump_fixed.vtk*) and a moving part (*pump_moving.vtk*). The moving part describes a rotational movement and the reservoir is pre-filled with fluid particles. *CasePump.bat* (*CasePump.sh* in linux) file summarises the tasks to be carried out using the different codes:

```
CasePump.bat

GenCase.exe CasePump_Def CasePump_out/CasePump

DualSPHysics.exe CasePump_out/CasePump -svres -gpu

PartVTK.exe  -dirin CasePump_out
             -savevtk CasePump_out/PartFluid -onlytype:-all,+fluid

BoundaryVTK.exe -loadvtk CasePump_out/CasePump__Actual.vtk
             -filexml CasePump_out/CasePump.xml
             -motiondata CasePump_out
             -savevtkdata CasePump_out/MotionPump -onlymk:13
             -savevtkdata CasePump_out/Pumpfixed.vtk -onlymk:11
```

In Figure 11-10, the fixed *Pumpfixed.vtk*, the different *MotionPump.vtk* and the *PartFluid_xxxx.vtk* files of the fluid particles are represented at different instants of the simulation.

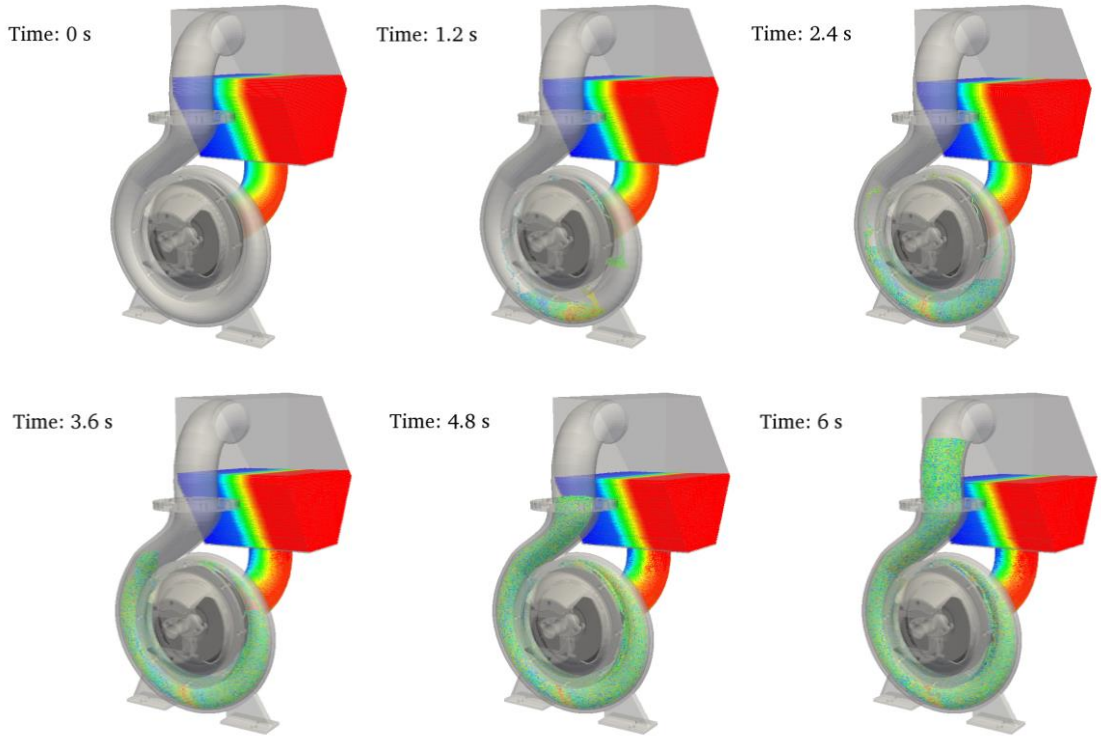


Figure 11-10. Different instants of the CasePump simulation.
Colour represents Id of the fluid particles.

11.6 CASEWAVEMAKER

CaseWavemaker simulates several waves breaking on a numerical beach. A wavemaker is performed to generate and propagate the waves. In this testcase, a sinusoidal movement is imposed to the boundary particles of the wavemaker. *CaseWavemaker.bat* (*CaseWavemaker.sh* in linux) file describes the execution task to be carried out in this directory:

```
CaseWavemaker.bat

GenCase.exe CaseWavemaker_Def  CaseWavemaker_out/CaseWavemaker

DualSPHysics.exe CaseWavemaker_out/CaseWavemaker -svres -gpu

BoundaryVTK.exe -loadvtk CaseWavemaker_out/CaseWavemaker__Actual.vtk
                -filexml CaseWavemaker_out/CaseWavemaker.xml
                -motiondatatime CaseWavemaker_out
                -savevtkdata CaseWavemaker_out/MotionPiston -onlymk:21
                -savevtkdata CaseWavemaker_out/Box.vtk -onlymk:11

PartVTK.exe     -dirin CaseWavemaker_out
                -savevtk CaseWavemaker_out/PartFluid -onlytype:-all,+fluid

MeasureTool.exe -dirin CaseWavemaker_out
                -points PointsHeights.txt
                -onlytype:-all,+fluid -height
                -savevtk CaseWavemaker_out/PointsHeights
                -savecsv CaseWavemaker_out/PointsHeights

IsoSurface.exe -dirin CaseWavemaker_out
               -saveiso CaseWavemaker_out/Surface -onlytype:-all,+fluid
```

The files *MotionPiston_xxxx.vtk* containing the position of the wavemaker at different instants and *Surface_xxxx.vtk* containing the surface representation are used to display the results in Figure 11-11:

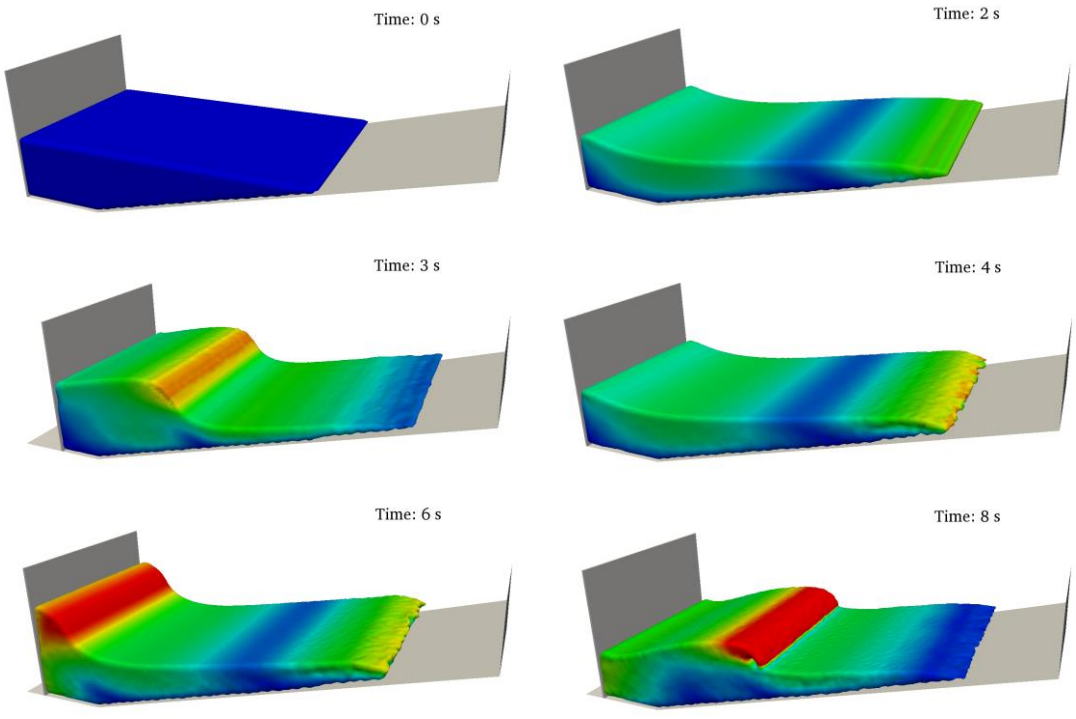


Figure 11-11. Different instants of the CaseWavemaker simulation. Colour represents velocity.

Using MeasureTool code, numerical wave elevations have been computed in the list of points described in the file *PointsHeights.txt*. So that, *PointsHeights_h_xxxx.vtk* are now depicted in Figure 11-12.

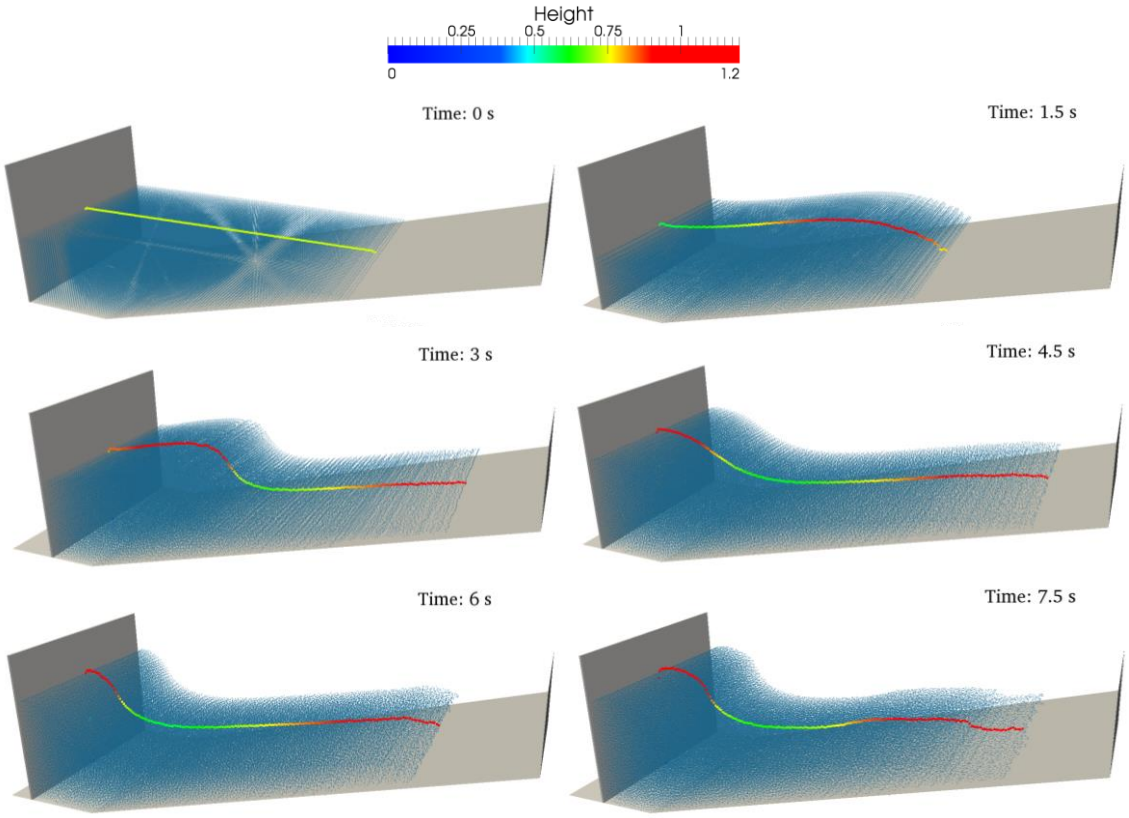


Figure 11-12. Different instants of the CaseWavemaker simulation.
Colour represents wave elevation.

MeasureTool is also used to compute the wave elevation at one wave gauge located at x=1, y=1 (wg0.txt) and the output result can be analysed in *WG0_Height.csv* as shown in Figure 11-13.

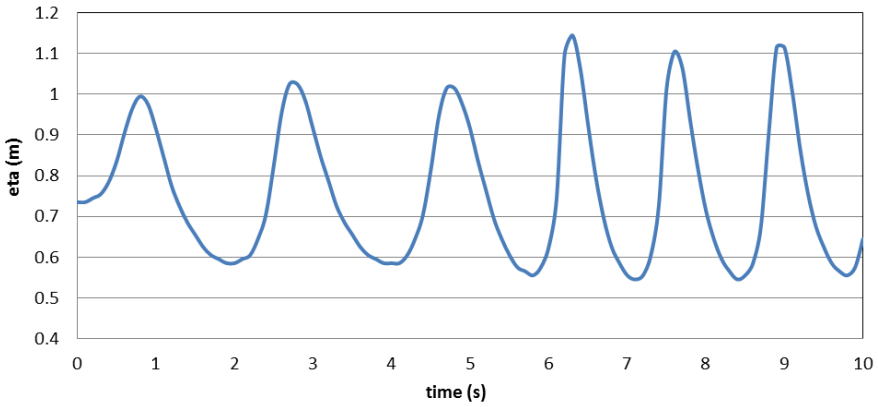


Figure 11-13. Numerical wave elevation at wg0 in CaseWavemaker.

12. How to modify DualSPHysics for your application

Creating new cases

More info is provided to help users to create their own cases of study:

- A guide presenting all the commands that can be used in the XML file. *GenCase_XML_GUIDE.pdf.*
- A guide describing how to convert the file format of any external geometry of a 3D model to VTK, PLY or STL using open-source codes. *ExternalModelsConversion_GUIDE.pdf.*
- More *XML examples* of some simulations from www.youtube/dualsphysics.

Source files of the SPH solver

To add new formulation or changes, the following files are the ones that require your attention:

CPU and GPU executions:
main.cpp
JCfgRun (.h .cpp)
JSph (.h .cpp)
JPartsOut (.h .cpp)
JSphVarAcc (.h .cpp)
Types.h

For only CPU executions:
JSphCpu (.h .cpp)
JSphCpu (.h .cpp)
JSphCpuSingle (.h .cpp)
JSphTimersCpu.h
JCellDivCpu (.h .cpp)
JCellDivCpuSingle (.h .cpp)
JPeriodicCpu (.h .cpp)

For only GPU executions:
JSphGpu (.h .cpp)
JSphGpu_ker (.h .cu)
JSphGpuSingle (.h .cpp)
JSphTimersGpu.h
JCellDivGpu (.h .cpp)
JCellDivGpu_ker (.h .cu)
JCellDivGpuSingle (.h .cpp)
JCellDivGpuSingle_ker (.h .cu)
JPeriodicGpu (.h .cpp)
JPeriodicGpu_ker (.h .cu)
JGpuArrays (.h .cpp)

Please read Section 5 for a complete description of these files.

Example code “ToVtk”

The new release includes not only the source files of DualSPHysics v3.0 but also the source files of a code named “ToVtk”. This code is provided to show how to load and interpret particle data, how to read .bi2 files and how to store .vtk files.

13. FAQ: Frequently asked questions about DualSPHysics

1. **What do I need to use DualSPHysics? What are the hardware and software requirements?**
2. **Why cannot I run DualSPHysics binary?**
3. **How can I compile the code with different environments/compilers?**
4. **How many particles can I simulate with the GPU code?**
5. **How should I start looking at the source code?**
6. **How can I create my own geometry?**
7. **How can I contribute to the project?**
8. **How does the code define the limits of the domain?**
9. **How can I define the movement of boundaries?**
10. **How do I prevent the boundary particles from going outside of the domain limits when applying motion?**
11. **Why do I observe a gap between boundaries, floating bodies and fluid in the solution?**
12. **When fluid particles are excluded from the simulation?**
13. **How do I create a 2-D simulation?**
14. **How must I cite the use of the code in my paper?**

1. What do I need to use DualSPHysics? What are the hardware and software requirements?

DualSPHysics can be executed either on CPU or GPU. In order to use DualSPHysics code on a GPU, you need a CUDA-enabled Nvidia GPU card on your machine (http://developer.nvidia.com/cuda-gpus). If you want to run GPU simulations (i.e. not develop the source code) the latest version of the driver for your graphics card must be installed. If no source code development is required, there is no need to install any compiler to run the binary of the code, only the driver must be updated. If you also want to compile the code you must install the *nvcc* compiler and a C++ compiler. The *nvcc* compiler is included in the CUDA Toolkit that can be downloaded from the Nvidia website and must be installed on your machine. CUDA versions 4.0, 4.1, 4.2, 5.0, and 5.5 can be used to compile.

2. Why cannot I run DualSPHysics binary?

If you are trying to run the executable GPU version on a CUDA-enabled Nvidia GPU card, the error message

Exception (JSphGpuSingle::SelecDevice)
Text: Failed getting devices info. (CUDA error: CUDA driver version is insufficient for CUDA runtime version)

can be solved by installing the latest version of the driver for the GPU card.

3. How can I compile the code with different environments/compilers?

The provided source files in this release can be compiled for linux using a 'makefile' along with *gcc* and *nvcc* compilers, and for windows using a project for Visual Studio

VS2010. In case you use another compiler or other environment you can adjust the contents of the makefile.

4. How many particles can I simulate with the GPU code?
The amount of particles that can be simulated depends on (i) the memory space of the GPU card and (ii) the options of the simulation.

5. How should I start looking at the source code?
Section 5 of this guide introduces the source files including some call graphs for a better understanding and it is also highly recommended that you read the documentation generated with Doxygen (www.doxygen.org).

6. How can I create my own geometry?
Using the Cases provided as a guide or starting point, to help users to create their own geometry the input XML files can be modified following *GenCase_XML_GUIDE.pdf* and different input formats of real geometries can be converted using *ExternalModelsConversion_GUIDE.pdf*. This manuscript also describes in detail the input files of the different test cases.

7. How can I contribute to the project?
You can contribute to the DualSPHysics project by reporting bugs, suggesting new improvements, citing DualSPHysics [See the answer to question 14] in your paper if you use it, submitting your modified codes together with examples.

8. How does the code define the limits of the domain?
In the input XML file, the parameters *pointmin* and *pointmax* only define the domain to create particles so out of these limits fluid or boundary particles will not be created. The limits of the computational domain are computed at the beginning of the DualSPHysics simulation and use the initial minimum and maximum positions of the particles that were already created with GenCase. In order to modify the limits automatically computed by DualSPHysics, different execution parameters can be used:

-domain_particles[:xmin,ymin,zmin,xmax,ymax,zmax]
-domain_particles_prc:xmin,ymin,zmin,xmax,ymax,zmax
-domain_fixed:xmin,ymin,zmin,xmax,ymax,zmax

so that the limits can be specified instead of using initial particle positions

9. How can I define the movement of boundaries?
Examples of the different type of movements that can be described with DualSPHysics are addressed in directory MOTION. Different kind of movements can be defined such as rectilinear, rotational, sinusoidal or circular motion and they can be uniform or accelerated, with pauses or with hierarchy of movements. And a last useful option is loading the movement from an external file with a prescribed movement (info of time, X-position, Y-position, Z-position and velocities) that will be interpolated at each time step during the simulation.

10. How do I prevent the boundary particles from going outside of the domain limits when applying motion?

As explained in the previous question, the limits of the computational domain are computed starting from the initial minimum and maximum positions of the particles. Since these values use the initial configuration, any movement of boundaries that implies positions beyond these limits will give us the error '*boundary particles out the domain'*. The solutions to solve this problem and to avoid creating larger tanks or domains are:

(i) defining boundary points <*drawpoint*> at the minimum and maximum positions that the particles are expected to reach during the simulation. The option <*drawpoint*> will create a particle at the given location *x,y,z*.

(ii) using the parameters of DualSPHysics execution mentioned in the answer to question 8.

11. Why do I observe a gap between boundaries, floating bodies and fluid in the solution?

The gap is a result of pressure overestimation across density discontinuities. It is inherent to the boundary formulation used in DualSPHysics. New boundary conditions are being developed and should be available in future releases.

12. When fluid particles are excluded from the simulation?

Fluid particles are excluded during the simulation; (i) if their positions are outside of the limits of the domain and (ii) when density values are out of a given range.

13. How do I create a 2-D simulation?

DualSPHysics can also perform 2-D simulations. To generate a 2-D configuration you only have to change the XML file; imposing the same values along Y-direction that define the limits of the domain where particles are created (**pointmin.y=1** and **pointmax.y=1**).

14. How must I cite the use of the code in my paper?

Please refer to the code if you use it in a paper with references [Crespo et al., 2011] and [Gómez-Gesteira et al., 2012a, 2012b].

14. New in DualSPHysics v3.0

The new version DualSPHysics v3.0 includes important improvements:

• Clearer structure of the CPU and GPU code.
This new implementation helps to follow the common structure of the CPU and GPU code. Now only CUDA kernels are implemented in the CUDA files (*.cu*).

• Improved behaviour of floating bodies.
The mass of the individual particles that form the floating bodies were corrected and now the proper behaviour of the object such as buoyancy can be simulated. In addition, a new testcase can be executed including experimental data for validation.

• Periodic open boundaries.
This new periodicity can be defined in any direction, which enables new possibilities. Periodicity enables simulation with feedback of particles that leave the end of the domain and are then re-injected creating a continuous flow such as shown in the new testcase. This provides a convenient approach to avoid wall friction when propagating waves in a 3D tank.

• Delta-SPH formulation.
Changes in fluid density are computed by solving the conservation of mass or continuity equation. In addition, the SPH scheme with numerical diffusive terms, named delta-SPH [Molteni and Colagrossi, 2009] can now be applied. This formulation improves the stability and accuracy of the code.

• Possibility of adding external forces.
Some cases can require external forces to be imparted on the fluid that vary both spatially and temporally (i.e. sloshing or varying gravity). This capability now exists by way of an external file that defines both linear and angular forces through time. A new test case is provided to demonstrate this functionality where extra forces are loaded from external files and applied to different sets of fluid particles.

• Friendly interface for pre-processing.
In order to make the process of defining a case XML for the GenCase application, a new Java based tool called the DualSPHysics Pre-processing Interface (DPI) has been created. This allows the creation/editing of XML files that will be used by GenCase using a more intuitive graphical user interface that includes in-depth textual feedback about the meaning of parameters that need to be set, it also provides instantaneous visual feedback as to the boundaries of the geometry of a case, decreasing the amount of time and effort needed to ensure objects are placed correctly within a simulation.

• More powerful GenCase.
With the new GenCase, larger cases can be rapidly created using OpenMP. In addition, new features facilitate the design of more complex cases.

• New properties of the particles defined by users.
New properties can be defined for different sets of particles. This encourages users to implement new options since it will be useful to define, for example, different phases (gas, solids) or objects with different properties (weight, centre...). More information can be found in the PDF file *GenCase_XML_GUIDE.pdf*.

• Example code ToVtk.
The new release includes not only the source files of DualSPHysics v3.0 but also the source files of a code named "ToVtk". This code is provided to show how to load and interpret particle data, how to read .bi2 files and how to store .vtk files.

• New options in MeasureTool.
Now the MeasureTool code can compute magnitudes at locations/points that change position with time. These locations can be the positions of moving particles or can be loaded from an external file.

15.DualSPHysics future

The new version **DualSPHysics v4.0** that will be released in 2014 will include:

- Multi-GPU implementation [Domínguez et al., 2013b].
- Double precision [Domínguez et al., 2013c].

Other features that are planned to be integrated into the DualSPHysics solver are:

- SPH-ALE with Riemann Solver.
- Primitive-variable Riemann Solver.
- Variable particle resolution [Vacondio et al., 2013].
- Multiphase (gas-soil-water) [Fourtakas et al., 2013a, Mokos et al., 2013].
- Inlet/outlet flow conditions.
- Extensions to the Modified virtual boundary conditions [Fourtakas et al., 2013b].
- Coupling with Discrete Element Method [Canelas et al., 2013].
- Coupling with Mass Point Lattice Spring Model [Longshaw et al., 2013].
- Coupling with SWASH Wave Propagation Model [Altomare et al., 2013].

16. References

1. Altomare C, Crespo AJC, Rogers BD, Domínguez JM, Gironella X, Gómez-Gesteira M. 2014. Numerical modelling of armour block sea breakwater with Smoothed Particle Hydrodynamics. Computers and Structures, 130, 34-45 doi:10.1016/j.compstruc.2013.10.011.

2. Altomare C, Suzuki T, Domínguez JM, Crespo AJC, Gómez-Gesteira M. 2013. Coupling SPH-SWASH. Oral communication in Iberian Workshop Advances on Smoothed Particle Hydrodynamics.

3. Barreiro A, Crespo AJC, Domínguez JM and Gómez-Gesteira M. 2013. Smoothed Particle Hydrodynamics for coastal engineering problems. Computers and Structures, 120(15), 96-106. doi:10.1016/j.compstruc.2013.02.010

4. Canelas R, Crespo AJC, Domínguez JM, Ferreira RML. 2013. A generalized SPH-DEM discretization for the modelling of complex multiphasic free surface flows. 8th SPHERIC.

5. Crespo AJC, Gómez-Gesteira M and Dalrymple RA. 2007. Boundary conditions generated by dynamic particles in SPH methods. Computers, Materials & Continua, 5, 173-184.

6. Crespo AJC, Gómez-Gesteira M and Dalrymple RA. 2008. Modeling Dam Break Behavior over a Wet Bed by a SPH Technique. Journal of Waterway, Port, Coastal, and Ocean Engineering, 134(6), 313-320.

7. Crespo AJC, Domínguez JM, Barreiro A, Gómez-Gesteira M and Rogers BD. 2011. GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods. PLoS ONE 6 (6), e20685, doi:10.1371/journal.pone.0020685.

8. Dalrymple RA and Rogers BD. 2006. Numerical modeling of water waves with the SPH method. Coastal Engineering, 53, 141–147.

9. Domínguez JM, Crespo AJC, Gómez-Gesteira M and Marongiu JC. 2011. Neighbour lists in Smoothed Particle Hydrodynamics. International Journal for Numerical Methods in Fluids, 67, 2026-2042, doi: 10.1002/fld.2481

10. Domínguez JM, Crespo AJC and Gómez-Gesteira M. 2013a. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. Computer Physics Communications, 184(3), 617-627. doi:10.1016/j.cpc.2012.10.015

11. Domínguez JM, Crespo AJC, Valdez-Balderas D, Rogers BD. and Gómez-Gesteira M. 2013b. New multi-GPU implementation for Smoothed Particle Hydrodynamics on heterogeneous clusters. Computer Physics Communications, 184, 1848-1860. doi: 10.1016/j.cpc.2013.03.008

12. Domínguez JM, Crespo AJC, Gómez-Gesteira M. 2013c. Simulating more than 1 billion SPH particles using GPU hardware acceleration. 8th SPHERIC. ISBN 9788876170195.

13. Fekken G. 2004. Numerical simulation of free surface flow with moving rigid bodies. Ph. D. thesis, University of Groningen.

14. Fourtakas G, Rogers BD, Laurence D. 2013a. Modelling sediment suspension in industrial tanks using SPH, La Houille Blanche, 2, 39-45, DOI: 10.1051/lhb/2013014.

15. Fourtakas G, Vacondio R, Rogers BD. 2013b. SPH approximate Zeroth and First-order consistent boundary conditions for irregular boundaries. 8th SPHERIC. ISBN 9788876170195.

16. Gómez-Gesteira M and Dalrymple R (2004) Using a 3D SPH method for wave impact on a tall structure. Journal of Waterway, Port, Coastal, and Ocean Engineering, 130(2), 63-69.

17. Gómez-Gesteira M, Rogers BD, Dalrymple RA and Crespo AJC. 2010a. State-of-the-art of classical SPH for free-surface flows. Journal of Hydraulic Research 48 Extra Issue, 6–27, doi:10.3826/jhr.2010.0012.

18. Gómez-Gesteira M, Rogers BD, Crespo AJC, Dalrymple RA, Narayanaswamy M and Domínguez JM. 2012a. SPHysics - development of a free-surface fluid solver- Part 1: Theory and Formulations. Computers & Geosciences, 48, 289-299. doi:10.1016/j.cageo.2012.02.029.

19. Gómez-Gesteira M, Crespo AJC, Rogers BD, Dalrymple RA, Domínguez JM and Barreiro A. 2012b. SPHysics - development of a free-surface fluid solver- Part 2: Efficiency and test cases. Computers & Geosciences, 48, 300-307. doi:10.1016/j.cageo.2012.02.028.

20. Koshizuka and Oka 1996 Koshizuka S. and Oka Y. 1996. Moving particle semi-implicit method for fragmentation of compressible ffuid. Nuclear Science Engineering, 123, 421-434.

21. Leimkuhler BJ, Reich S, Skeel RD. 1996. Integration Methods for Molecular dynamic IMA Volume in Mathematics and its application. Springer.

22. Longshaw SM, Rogers BD, Stansby PK. 2013. Integration Of Spring Physics With The SPH Method For Quasi-Solid To Fluid Interaction Using GPGPU Programming. 8th SPHERIC. ISBN 9788876170195.

23. Lorensen WE, Cline HE. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm, Computer Graphics SIGGRAPH 87 Proceedings, 21, 4, 163-170.

24. Mokos A, Rogers BD, Stansby PK, Domínguez JM. 2013. GPU Acceleration of 3-D Multi-phase SPH Simulations for Violent Hydrodynamics. 8th SPHERIC. ISBN 9788876170195.

25. Molteni, D, Colagrossi A. 2009. A simple procedure to improve the pressure evaluation in hydrodynamic context using the SPH, Comput. Phys. Comm., 180, 6, 861–872.

26. Monaghan JJ. 1992. Smoothed particle hydrodynamics. Annual Review of Astronomy and Astrophysics 30, 543- 574.

27. Monaghan JJ, and Lattanzio JC. 1985. A refined method for astrophysical problems. Astron. Astrophys. 149, 135–143.

28. Monaghan JJ and Kos A. 1999. Solitary waves on a Cretan beach. Journal of Waterway, Port, Coastal and Ocean Engineering 125, 145-154.

29. Monaghan JJ, Kos A and Issa N. 2003. Fluid motion generated by impact. Journal of Waterway, Port, Coastal and Ocean Engineering 129, 250-259.

30. Moyo S. and Greenhow M. 2000. Free motion of a cylinder moving below and through a free surface. Applied Ocean Research 22, 31–44.

31. Panizzo A. 2004. Physical and Numerical Modelling of Subaerial Landslide Generated Waves. PhD thesis, Universita degli Studi di L'Aquila.

32. Rogers BD, Dalrymple RA, Stansby PK. 2010. Simulation of caisson breakwater movement using SPH. Journal of Hydraulic Research, 48, 135-141, doi:10.3826/jhr.2010.0013.

33. Vacondio R, Rogers B D, Stansby P K, Mignosa P, Feldman J. 2013. Variable resolution for SPH: a dynamic particle coalescing and splitting scheme. Computer Methods in Applied Mechanics and Engineering, 256, 132-148.

34. Verlet L. 1967. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. Physical Review, 159, 98-103.

35. Wendland H. 1995. Piecewiese polynomial, positive definite and compactly supported radial functions of minimal degree. Advances in Computational Mathematics 4, 389-396.

17. Licenses

GPL License

The source code for DualSPHysics is freely redistributable under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation.
Simply put, the GPL says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it. By distributing the complete source code for GNU DualSPHysics under the terms of the GPL, we guarantee that you and all other users will have the freedom to redistribute and change DualSPHysics.
Releasing the source code for DualSPHysics has another benefit as well. By having access to all of the source code for a mathematical system like SPHysics, you have the ability to see exactly how each and every computation is performed.
Although enhancements to DualSPHysics are not required to be redistributed under the terms of the GPL, we encourage you to release your enhancements to SPHysics under the same terms for the benefit of all users. We also encourage you to submit your changes for inclusion in future versions of DualSPHysics.

BSD License

DualSPHysics is an international collaboration between University of Vigo (Spain) and University of Manchester (UK).

TinyXml License

www.sourceforge.net/projects/tinyxml

Original code (2.0 and earlier) copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.