MANCHESTER 1824

The University of Manchester

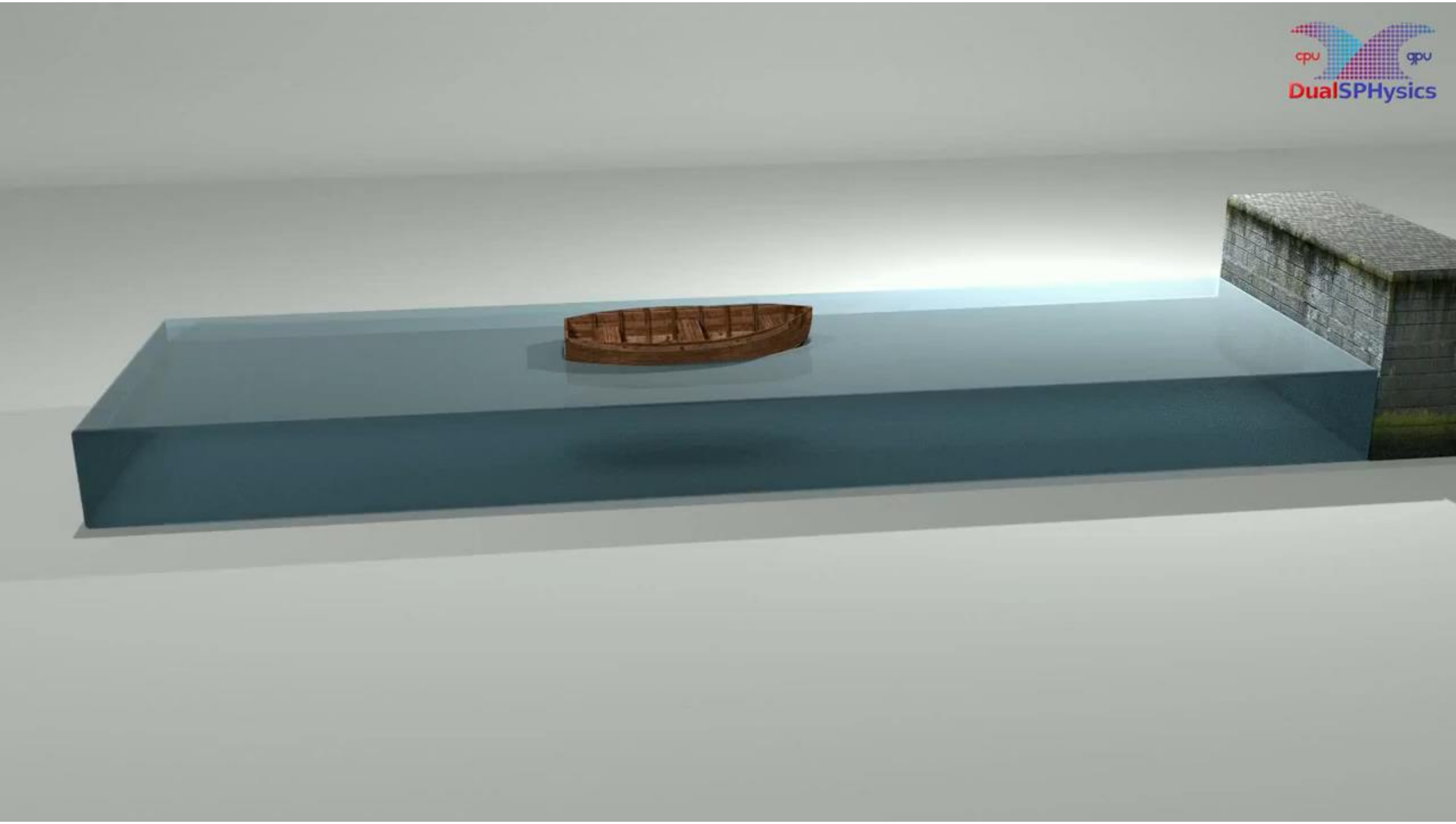
DualSPHysics code
New version 4.0

Dr José Domínguez
University of Vigo, SPAIN

Dr Georgios Fourtakas
The University of Manchester, UK

Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0


cpu
gpu
DualSPHysics

Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0

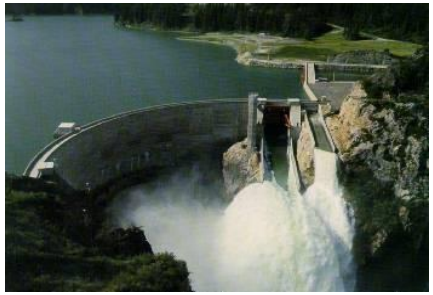
1.1. Origin of DualSPHysics

SPH method was invented for astrophysics during the seventies, but now it is applied in many different fields including fluid dynamics and solid mechanics.

Fluid is represented using particles which move according to the governing dynamics.

SPH is particularly suited to describe a variety of **free-surface flows**:

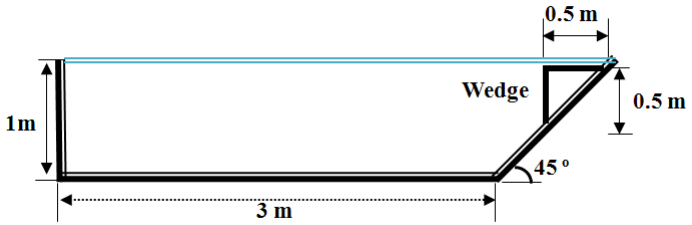
- Wave propagation over a beach.
- Plunging breakers.
- Wave-structure interactions.
- Solid bodies impacting on water surface.
- Dam breaks.

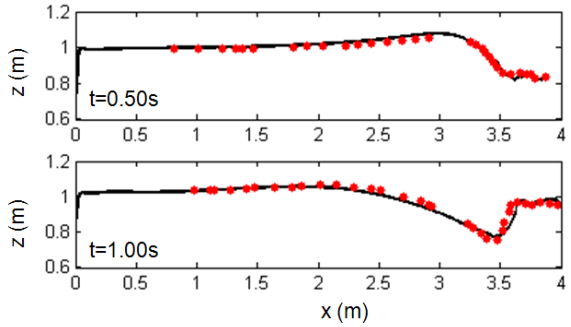


1.1. Origin of DualSPHysics

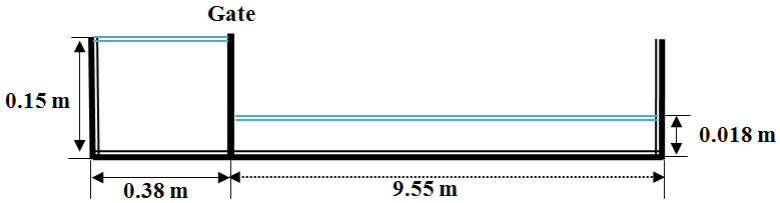
The SPHysics group focused its research mainly on wave propagation and interaction with coastal structures, in 2D and 3D.

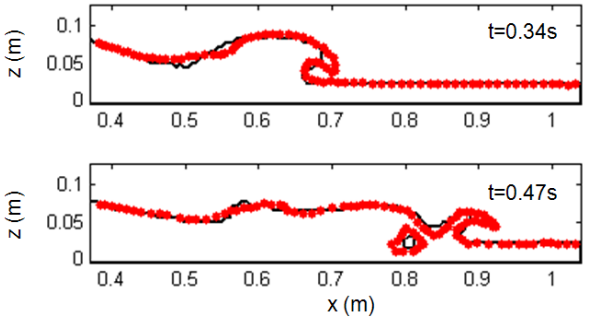
Waves generated by landslides



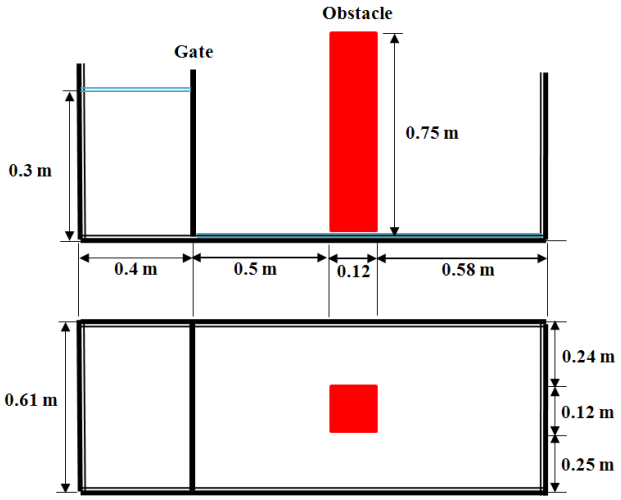


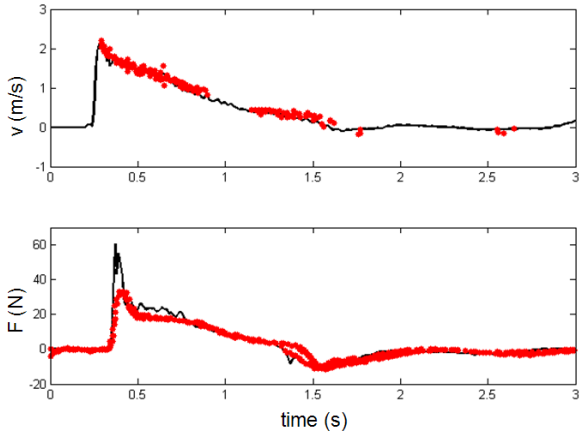
Dam-break propagation over wet beds





3D Wave-structure interaction.





1.1. Origin of DualSPHysics

Drawbacks of SPH:

SPH presents a **high computational cost** that increases when increasing the number of particles.

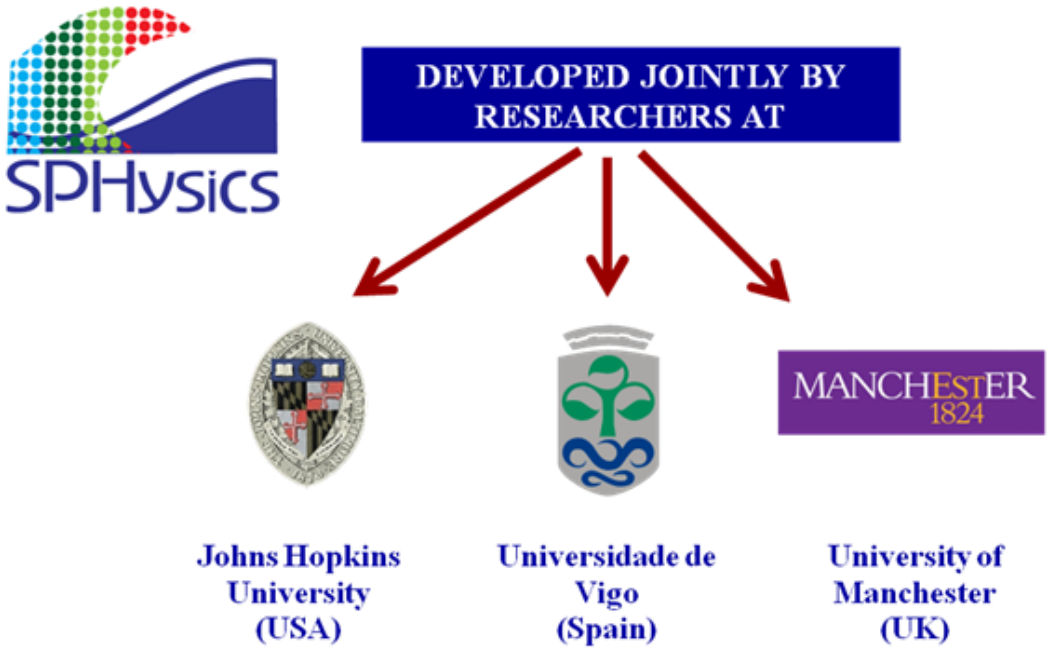
The simulation of **real problems** requires a high resolution which implies simulating **millions of particles**.

The **time required** to simulate a few seconds is **too large**. One second of physical time can take several days of calculation.

IT IS NECESSARY TO USE HPC TECHNIQUES TO REDUCE THESE COMPUTATION TIMES.

1.1. Origin of DualSPHysics

The DualSPHysics code was created starting from SPHysics www.sphysics.org.



SPHysics is a numerical model SPH developed for the study of free-surface problems.

It is a code written in Fortran90 with numerous options (different kernels, several boundary conditions,…), which had already demonstrated high accuracy in several validations with experimental results… but it is too slow to apply to large domains.

1.1. Origin of DualSPHysics

The problem:

- SPH PRESENTS A HIGH COMPUTATIONAL COST THAT INCREASES WHEN INCREASING THE NUMBER OF PARTICLES
- THE SIMULATION OF REAL PROBLEMS REQUIRES A HIGH RESOLUTION WHICH IMPLIES SIMULATING MILLIONS OF PARTICLES

IT WAS NECESSARY TO INCREASE THE SPEED OF THE CODE BY A FACTOR 100x

Classic options:

- **OpenMP:** Distribute the workload among all CPU cores (≈8x)
- **MPI:** Combines the power of multiple machines connected via network (high cost).

New option:

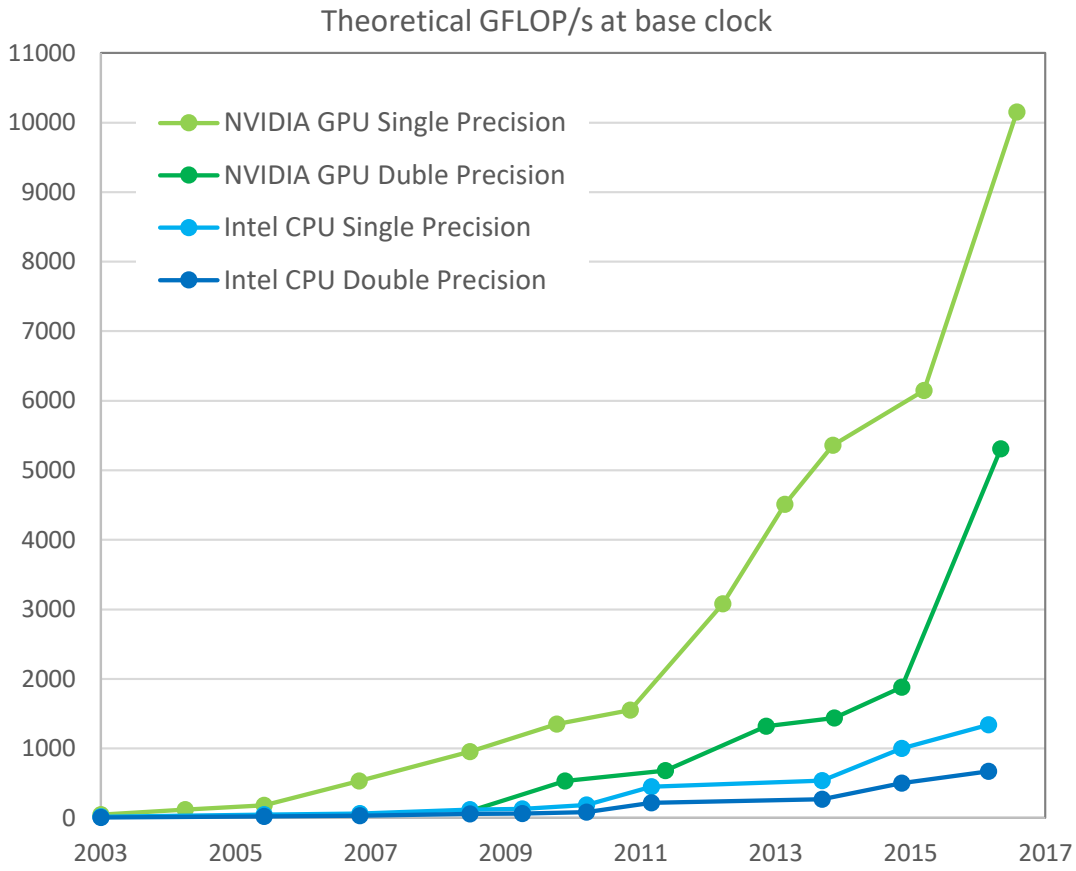
- **GPU:** Graphics cards with a high parallel computing power (cheap and accessible).

1.2. Why GPUs?

Graphics Processing Units (GPUs) are powerful parallel processors originally designed for graphics rendering.

Due to the development of the video games market and multimedia, their computing power has increased much faster than CPUs.

Now, GPUs can be used for general purpose applications, achieving speedups of 100x or more.



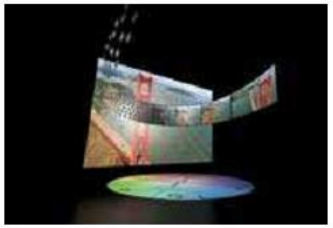
Advantages: GPUs provide the necessary power with very low cost and without expensive infrastructures.

Drawbacks: An efficient and full use of the capabilities of the GPUs is not straightforward.

1.2. Why GPUs?

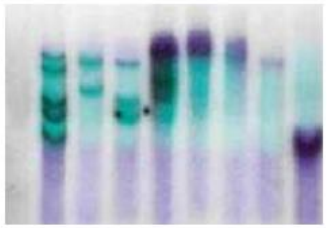
GPUs are an accessible tool to accelerate SPH, all numerical methods in CFD and any computational method

5X
Digital Content Creation
Adobe

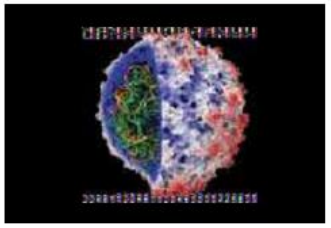


18X
Video Transcoding
Elemental Technologies

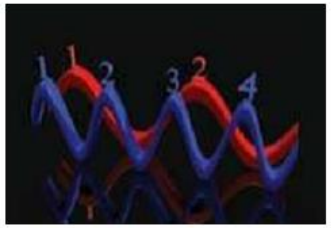
20X
3D Ultrasound
TechniScan



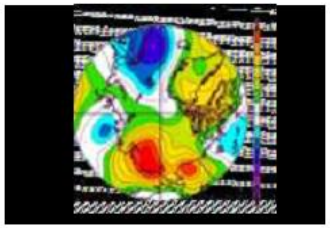
30X
Gene Sequencing
U of Maryland



36X
Molecular Dynamics
U of Illinois, Urbana-Champaign



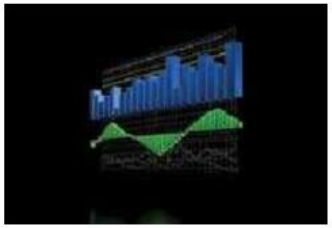
50X
MATLAB Computing
AccelerEyes



80X
Weather Modeling
Tokyo Institute of Technology

100X
Astrophysics
RIKEN

146X
Medical Imaging
U of Utah



149X
Financial Simulation
Oxford University

http://www.nvidia.com

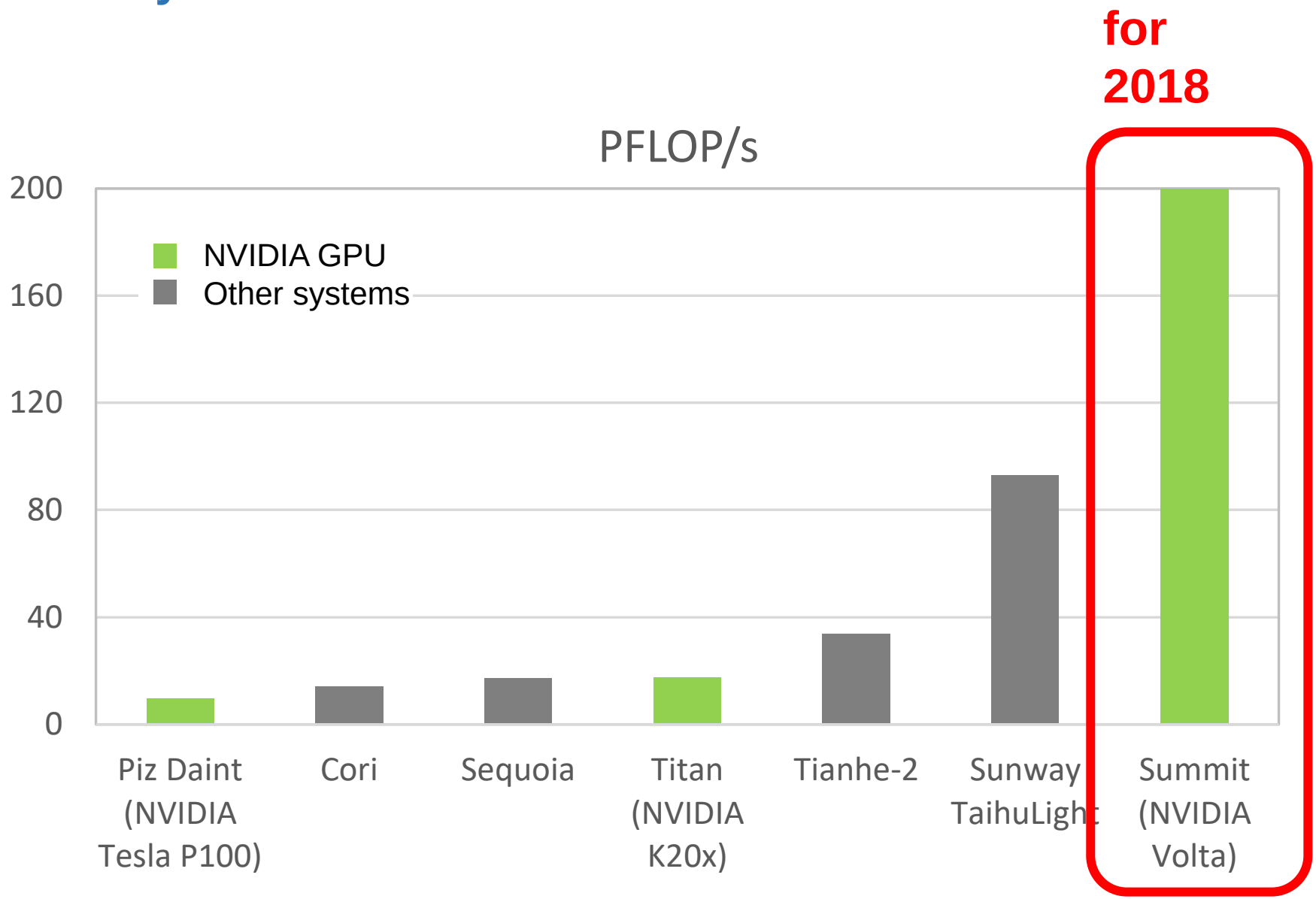
1.2. Why GPUs?

https://www.top500.org/lists/2016/11/

TOP500 LIST – NOVEMEBER 2016

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
5	DOE/SC/LBNL/NERSC United States	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	622,336	14,014.7	27,880.7	3,939
8	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	206,720	9,779.0	15,988.0	1,312

1.2. Why GPUs?


for 2018
PFLOP/s
200
160
120
80
40
0
NVIDIA GPU
Other systems
Piz Daint (NVIDIA Tesla P100)
Cori
Sequoia
Titan (NVIDIA K20x)
Tianhe-2
Sunway TaihuLight
Summit (NVIDIA Volta)

1.2. Why GPUs?

https://www.top500.org/green500/lists/2016/06/

GREEN500 LIST – JUNE 2016

Green500 Rank	MFLOPS/W	Site	System	Total Power(kW)
1	6673.8	Advanced Center for Computing and Communication, RIKEN	ZettaScaler-1.6, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband FDR, PEZY-SCnp	150.0
2	6195.2	Computational Astrophysics Laboratory, RIKEN	ZettaScaler-1.6, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband FDR, PEZY-SCnp	46.9
3	6051.3	National Supercomputing Center in Wuxi	Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway	15371
4	5272.1	GSI Helmholtz Center	ASUS ESC4000 FDR/G2S, Intel Xeon E5-2690v2 10C 3GHz, Infiniband FDR, AMD FirePro S9150	57.2
5	4778.5	Institute of Modern Physics (IMP), Chinese Academy of Sciences	Sugon Cluster W780I, Xeon E5-2640v3 8C 2.6GHz, Infiniband QDR, NVIDIA Tesla K80	65
6	4112.1	Stanford Research Computing Center	Cray CS-Storm, Intel Xeon E5-2680v2 10C 2.8GHz, Infiniband FDR, Nvidia K80	190
7	3775.5	Internet Service (B)	Inspur TS10000 HPC Server, Intel Xeon E5-2620v2 6C 2.1GHz, 10G Ethernet, NVIDIA Tesla K40	110
8	3775.5	Internet Service (B)	Inspur TS10000 HPC Server, Intel Xeon E5-2620v2 6C 2.1GHz, 10G Ethernet, NVIDIA Tesla K40	110
9	3775.5	Internet Service (B)	Inspur TS10000 HPC Server, Intel Xeon E5-2620v2 6C 2.1GHz, 10G Ethernet, NVIDIA Tesla K40	110
10	3775.5	Internet Service (B)	Inspur TS10000 HPC Server, Intel Xeon E5-2620v2 6C 2.1GHz, 10G Ethernet, NVIDIA Tesla K40	110

1.3. DualSPHysics project

First version in late 2009.

It includes **two implementations**:

- **CPU**: C++ and OpenMP.
- **GPU**: CUDA.

Both options optimized for the best performance of each architecture.

Why two implementations?

This code can be used on machines with GPU and without GPU.

It allows us to make a fair and realistic comparison between CPU and GPU.

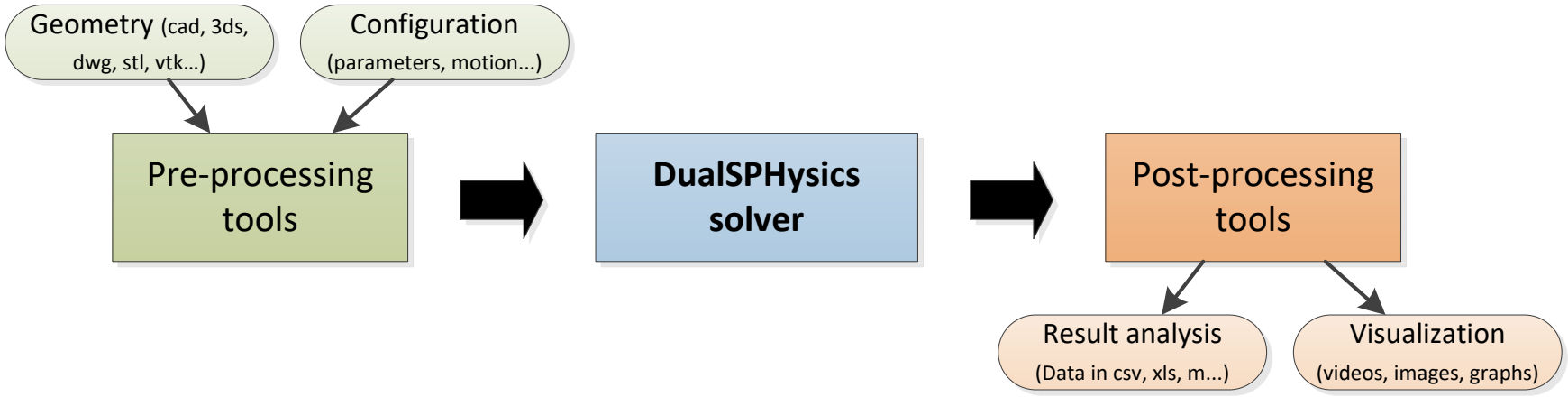
Some algorithms are complex and it is easy to make errors difficult to detect. So they are implemented twice and we can compare results.

It is easier to understand the code in CUDA when you can see the same code in C++.

Drawback: It is necessary to implement and to maintain two different codes.

1.3. DualSPHysics project

DSPH project includes:



Pre-processing tools:

- Converts geometry into particles.
- Provides configuration for simulation.

DualSPHysics solver:

- Runs simulation using SPH particles.
- Obtains data simulation for time intervals.

Post-processing tools:

- Calculates magnitudes using particle data.
- Generates images and videos starting form SPH particles.

1.3. DualSPHysics project

DualSPHysics

FAQ References Downloads Validation Animations SPHysics GPU Computing
Developers Contact News Forums

Universidade Vigo

MANCHESTER 1824
The University of Manchester

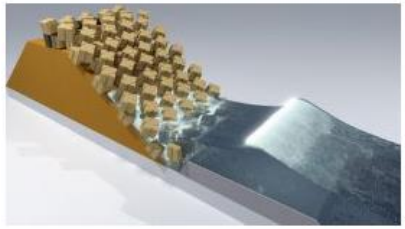
DualSPHysics is based on the Smoothed Particle Hydrodynamics model named SPHysics (www.sphysics.org).

The code is developed to study free-surface flow phenomena where Eulerian methods can be difficult to apply, such as waves or impact of dam-breaks on off-shore structures. **DualSPHysics** is a set of C++, CUDA and Java codes designed to deal with real-life engineering problems.

Contact E-Mail: dualsphysics@gmail.com

Youtube Channel: www.youtube.com/user/DualSPHysics

Twitter Account: @DualSPHysics



www.dual.sphysics.org

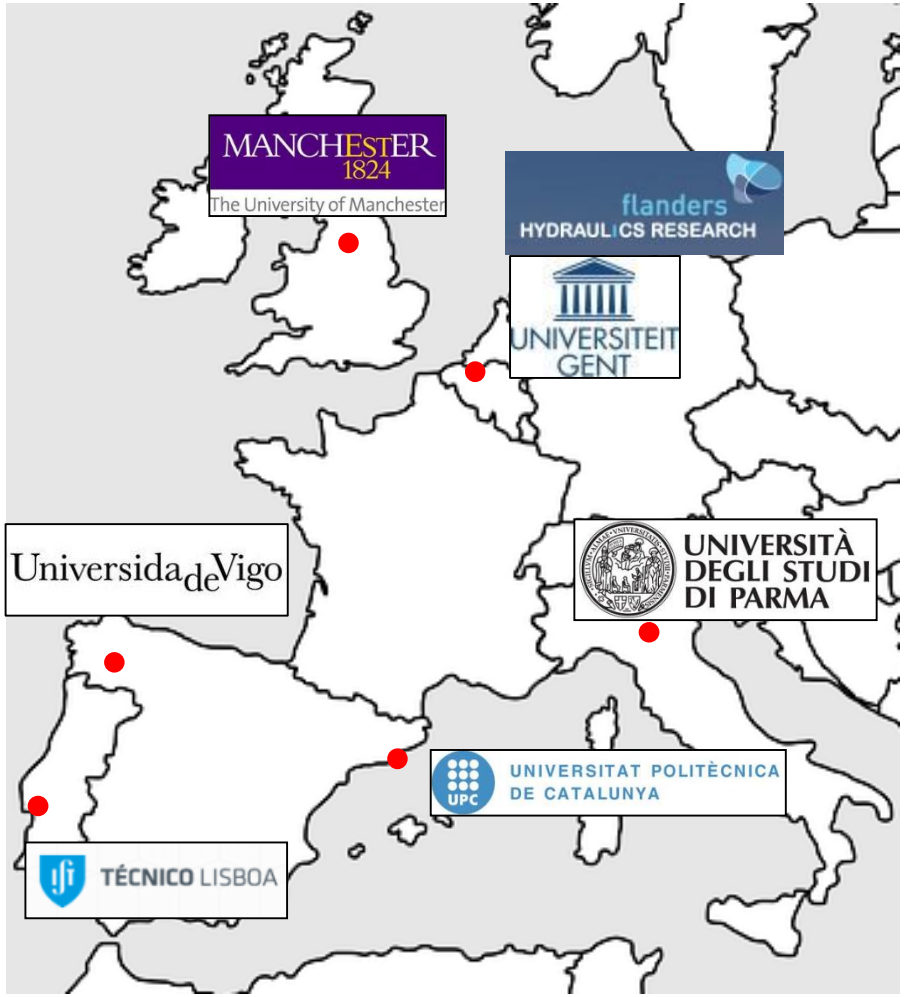
1.3. DualSPHysics project

People working on DualSPHysics project:

Dr Benedict D. Rogers
Dr Athanasios Mokos
Dr Georgios Fourtakas
Dr Steve Lind
Prof. Peter Stansby
Abouzied Nasar
Alex Chow
Annelie Baines
Aaron English

Prof. Moncho Gómez Gesteira
Dr Alejandro J.C. Crespo
Dr Jose M. Domínguez
Dr José González-Cao
Orlando G. Feal

Prof. Rui Ferreira
Dr Ricardo Canelas



Dr Corrado Altomare
Dr Tomohiro Suzuki

Dr Renato Vacondio
Prof. Paolo Mignosa

Dr Xavier Gironella
Andrea Marzeddu

1.3. DualSPHysics project

Developers v4.0:

The University of Manchester, UK
Dr Benedict D. Rogers
Dr Georgios Fourtakas
Dr Athanasios Mokos

Science & Technology Facilities Council, UK
Dr Stephen Longshaw

Università degli studi di Parma, Italy
Dr Renato Vacondio

Universiteit Gent –
Flanders Hydraulics Research, Belgium
Dr Corrado Altomare

Universidade de Vigo, Spain
Dr José M. Domínguez
Dr Alejandro J.C. Crespo
Dr Anxo Barreiro
Professor Moncho Gómez Gesteira

EPHYTECH SL, Spain
Orlando G. Feal

Instituto Superior Tecnico, Lisbon, Portugal
Dr Ricardo Canelas

Contributors:

Imperial College London, UK. Mashy D Green
Universidad Politécnica de Madrid, Spain. Dr. Jose Luis Cercós Pita.
Universidad de Guanajuato, Mexico. Carlos Enrique Alvarado Rodríguez.
NYU Tandon School of Engineering, USA. Dr. Angelo Tafuni

1.3. DualSPHysics project

Developers v4.0:

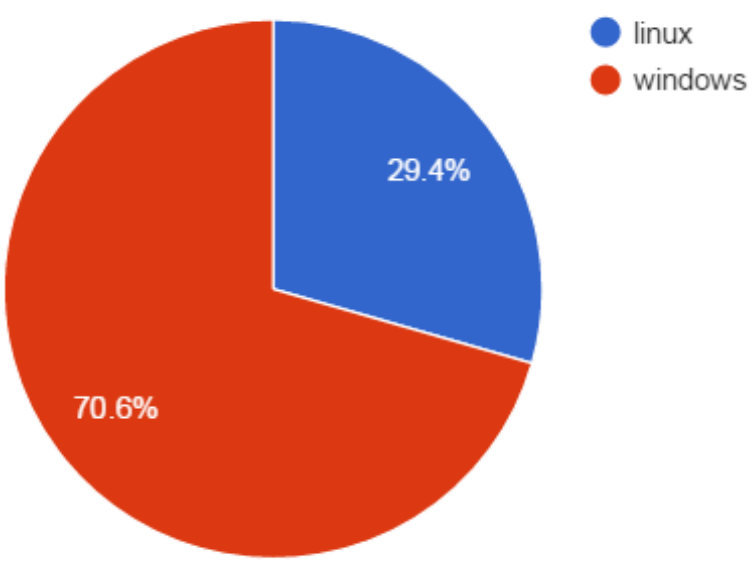
Contributors:

1.3. DualSPHysics project

NUMBER OF DOWNLOADS

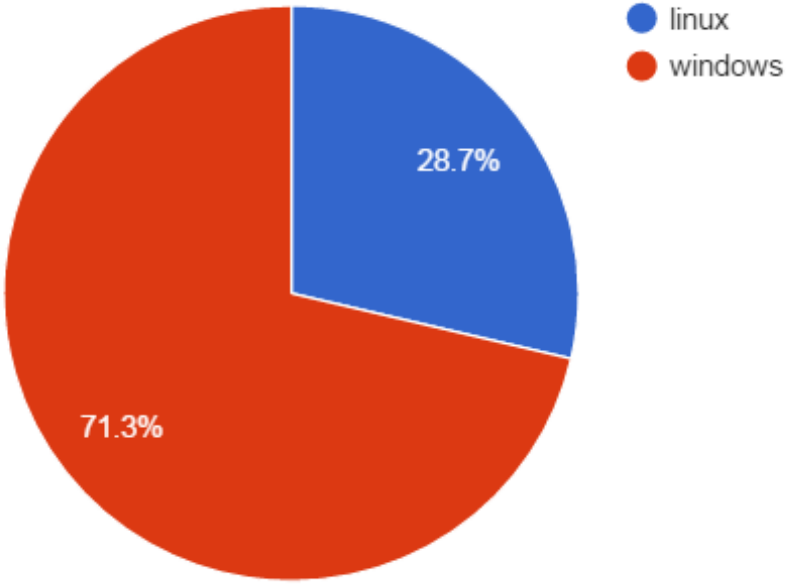
DUALSPHYSICS V4.0

linux	906	29%
windows	2,171	71%
TOTAL	3,077	



DUALSPHYSICS ALL VERSIONS

linux	4,773	28.65%
windows	11,885	71.35%
TOTAL	**16,658**	



1.3. DualSPHysics project

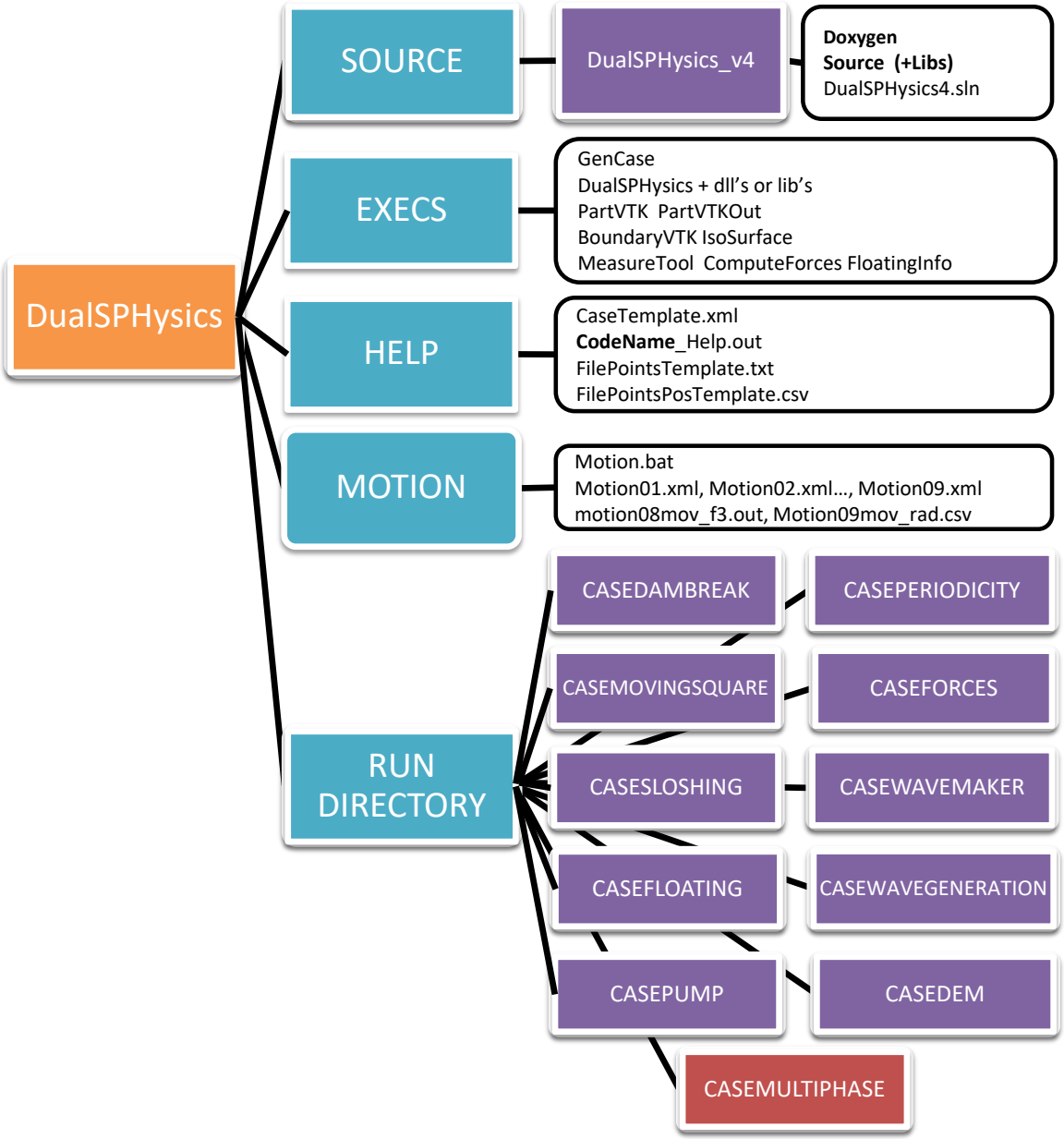
LAST RELEASE OF DUALSPHYSICS CODE:

http://dual.sphysics.org/index.php/downloads/

- **DUALSPHYSICS DOCUMENTATION:**
  - **DualSPHysics_v4.0_GUIDE.pdf**
  - **XML_GUIDE_v4.0.pdf**
  - **ExternalModelsConversion_GUIDE.pdf**
  - **PostprocessingCalculations.pdf**

- **DUALSPHYSICS PACKAGE:**
  - **DualSPHysics_v4.0_Linux_x64.zip**
  - **DualSPHysics_v4.0_Windows_x64.zip**

1.3. DualSPHysics project



Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0

2. SPH formulation: DualSPHysics v4

- 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 treatment:
  - 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.

2. SPH formulation: DualSPHysics v4

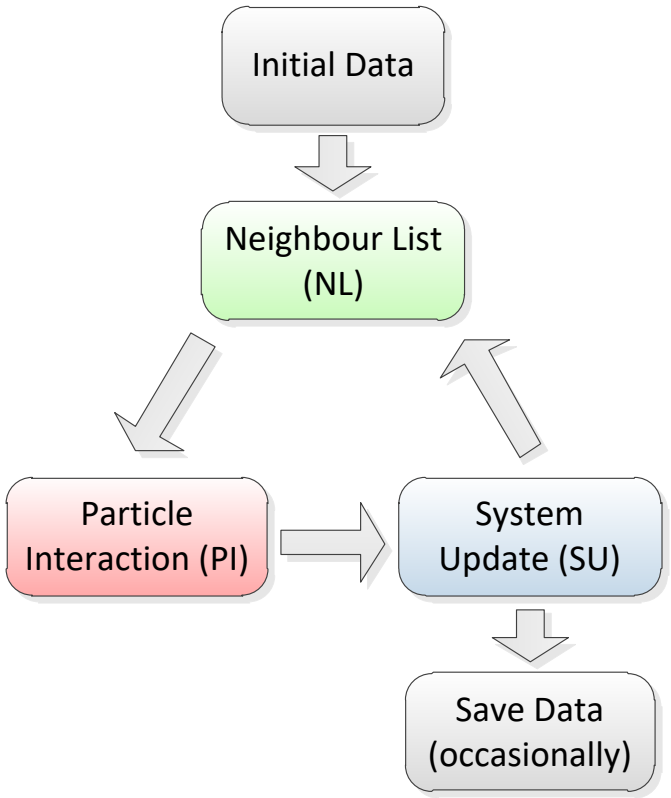
- Shifting algorithm (Lind et al., 2012).
- Dynamic boundary conditions (Crespo et al., 2007).
- Floating objects (Monaghan et al., 2003).
- Periodic open boundaries (Gómez-Gesteira et al., 2012a)
- Coupling with Discrete Element Method (Canelas et al., 2016).
- External body forces (Longshaw and Rogers, 2015).
- Double precision (Domínguez et al., 2013c).
- Wave generation (Biesel and Suquet, 1951; Madsen, 1971; Liu and Frigaard, 2001).
- Multi-phase (soil-water) (Fourtakas and Rogers, 2016) – executable only.

Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0

3.1. Steps of simulation

For the implementation of SPH, the code is organised in **3 main steps** that are repeated each time step till the end of the simulation.



Neighbour list (NL):
Particles are grouped in cells and reordered to optimise the next step.

Particle interactions (PI):
Forces between particles are computed, solving momentum and continuity equations.
This step takes **more than 95%** of execution time.

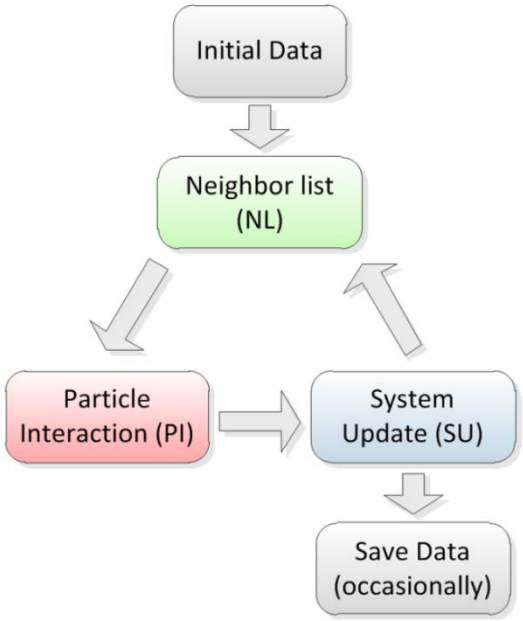
System update (SU):
Starting from the values of computed forces, the magnitudes of the particles are updated for the next instant of the simulation.

3.1. Steps of simulation

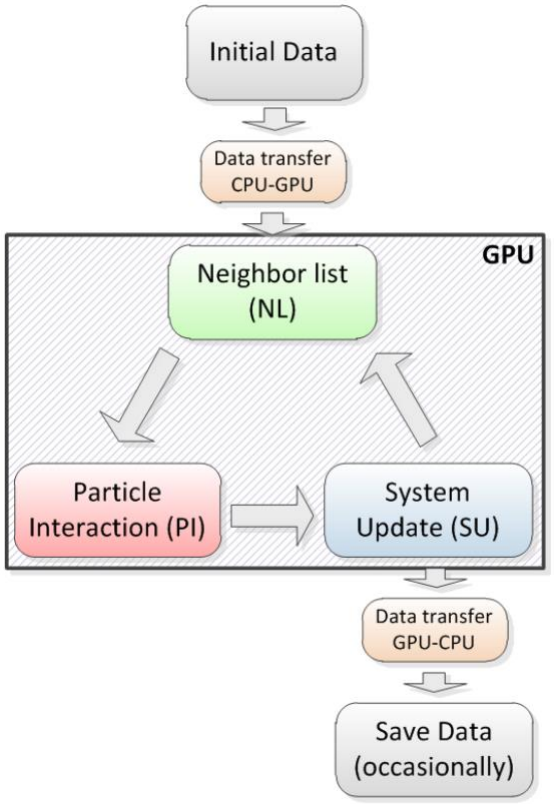
Full GPU implementation

- GPU is used in all steps (Neighbour List, Particle Interaction and System Update).
- All particle data is kept in GPU memory to avoid transfers CPU-GPU at each time step.

CPU Implementation



GPU implementation



3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp)	main.cpp	
FunctionsMath (.h .cpp)	JCfgRun (.h .cpp)	
JBinaryData (.h .cpp)	JPartsLoad4 (.h .cpp)	
JException (.h .cpp)	JPartsOut (.h .cpp)	
JLog2 (.h .cpp)	JSaveDt.cpp (.h .cpp)	
JMatrix4.h	JSph (.h .cpp)	
JMeanValues (.h .cpp)	JSphAccInput (.h .cpp)	
JObject (.h .cpp)	JSphDtFixed (.h .cpp)	
JObjectGpu (.h .cpp)	JSphVisco (.h .cpp)	
JPartDataBi4 (.h .cpp)	JTimerClock.h	
JPartFloatBi4 (.h .cpp)	JTimeOut (.h .cpp)	
JPartOutBi4Save (.h .cpp)	Types.h	
JRadixSort (.h .cpp)		
JRangeFilter (.h .cpp)	JSphCpu (.h .cpp)	JSphGpu (.h .cpp)
JReadDatafile (.h .cpp)		JSphGpu_ker (.h .cu)
JSpaceCtes (.h .cpp)	JSphCpuSingle (.h .cpp)	JSphGpuSingle (.h .cpp)
JSpaceEParms (.h .cpp)		
JSpaceParts (.h .cpp)	JSphTimersCpu.h	JSphTimersGpu.h
JSpaceProperties (.h .cpp)		
JTimer.h	JCellDivCpu (.h .cpp)	JCellDivGpu (.h .cpp)
JTimerCuda.h		JCellDivGpu_ker (.h .cu)
TypesDef.h		
	JCellDivCpuSingle (.h .cpp)	JCellDivGpuSingle (.h .cpp)
JFormatFiles2.h		JCellDivGpuSingle_ker (.h .cu)
JSphMotion.h		
JXml.h	JArraysCpu (.h .cpp)	JArraysGpu (.h .cpp)
JWaveGen.h		JBlockSizeAuto (.h .cpp)

3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp)	main.cpp	
FunctionsMath (.h .cpp)	JCfgRun (.h .cpp)	
JBinaryData (.h .cpp)	JPartsLoad4 (.h .cpp)	
JException (.h .cpp)	JPartsOut (.h .cpp)	
JLog2 (.h .cpp)	JSaveDt.cpp (.h .cpp)	
JMatrix4.h	JSph (.h .cpp)	
JMeanValues (.h .cpp)	JSphAccInput (.h .cpp)	
JObject (.h .cpp)	JSphDtFixed (.h .cpp)	
JObjectGpu (.h .cpp)	JSphVisco (.h .cpp)	
JPartDataBi4 (.h .cpp)	JTimerClock.h	
JPartFloatBi4 (.h .cpp)	JTimeOut (.h .cpp)	
JPartOutBi4Save (.h .cpp)	Types.h	
JRadixSort (.h .cpp)		
JRangeFilter (.h .cpp)	JSphCpu (.h .cpp)	JSphGpu (.h .cpp)
JReadDatafile (.h .cpp)		JSphGpu_ker (.h .cu)
JSpaceCtes (.h .cpp)	JSphCpuSingle (.h .cpp)	JSphGpuSingle (.h .cpp)
JSpaceEParms (.h .cpp)		
JSpaceParts (.h .cpp)	JSphTimersCpu.h	JSphTimersGpu.h
JSpaceProperties (.h .cpp)		
JTimer.h	JCellDivCpu (.h .cpp)	JCellDivGpu (.h .cpp)
JTimerCuda.h		JCellDivGpu_ker (.h .cu)
TypesDef.h		
	JCellDivCpuSingle (.h .cpp)	JCellDivGpuSingle (.h .cpp)
JFormatFiles2.h		JCellDivGpuSingle_ker (.h .cu)
JSphMotion.h		
JXml.h	JArraysCpu (.h .cpp)	JArraysGpu (.h .cpp)
JWaveGen.h		JBlockSizeAuto (.h .cpp)

Files:

49 C++ headers

36 C++ codes

3 CUDA headers

3 CUDA codes

3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp) FunctionsMath (.h .cpp) JBinaryData (.h .cpp) JException (.h .cpp) JLog2 (.h .cpp) JMatrix4.h JMeanValues (.h .cpp) JObject (.h .cpp) JObjectGpu (.h .cpp) JPartDataBi4 (.h .cpp) JPartFloatBi4 (.h .cpp) JPartOutBi4Save (.h .cpp) JRadixSort (.h .cpp) JRangeFilter (.h .cpp) JReadDatafile (.h .cpp) JSpaceCtes (.h .cpp) JSpaceEParms (.h .cpp) JSpaceParts (.h .cpp) JSpaceProperties (.h .cpp) JTimer.h JTimerCuda.h TypesDef.h	JSphDtFixed (.h .cpp) JSphVisco (.h .cpp) JTimerClock.h JTimeOut (.h .cpp) Types.h	
	JSphCpu (.h .cpp) JSphCpuSingle (.h .cpp) JSphTimersCpu.h JCellDivCpu (.h .cpp) JCellDivCpuSingle (.h .cpp)	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)
JFormatFiles2.h JSphMotion.h JXml.h JWaveGen.h	JArraysCpu (.h .cpp)	JArraysGpu (.h .cpp) JBlockSizeAuto (.h .cpp)

Common files used in several pre-processing and post-processing applications.

They provide basic functionalities.

Users do not need to modify them.

3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp) FunctionsMath (.h .cpp) JBinaryData (.h .cpp) JException (.h .cpp) JLog2 (.h .cpp) JMatrix4.h JMeanValues (.h .cpp) JObject (.h .cpp) JObjectGpu (.h .cpp) JPartDataBi4 (.h .cpp) JPartFloatBi4 (.h .cpp) JPartOutBi4Save (.h .cpp) JRadixSort (.h .cpp) JRangeFilter (.h .cpp) JReadDatafile (.h .cpp) JSpaceCtes (.h .cpp) JSpaceEParms (.h .cpp) JSpaceParts (.h .cpp) JSpaceProperties (.h .cpp) JTimer.h JTimerCuda.h TypesDef.h	main.cpp JCfgRun (.h .cpp) JPartsLoad4 (.h .cpp) JPartsOut (.h .cpp) JSaveDt.cpp (.h .cpp) JSph (.h .cpp) JSphAccInput (.h .cpp) JSphDtFixed (.h .cpp) JSphVisco (.h .cpp) JTimerClock.h JTimeOut (.h .cpp) Types.h	
	JSphCpu (.h .cpp) JSphCpuSingle (.h .cpp) JSphTimersCpu.h JCellDivC JC llDivC JArraysCpu (.h .cpp)	JSphGpu (.h .cpp) JSphGpu_ker (.h .cu) JSphGpuSingle (.h .cpp) JSphTimersGpu.h JArraysGpu (.h .cpp) JBlockSizeAuto (.h .cpp)

JFormatFiles2.h
JSphMotion.h
JXml.h
JWaveGen.h

Precompiled code in libraries.

Common files used in several pre-processing and post-processing applications.

They provide basic functionalities.

3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp)	main.cpp	
FunctionsMath (.h .cpp)	JCfgRun (.h .cpp)	
JBinaryData (.h .cpp)	JPartsLoad4 (.h .cpp)	
	JPartsOut (.h .cpp)	
	JSaveDt.cpp (.h .cpp)	
	JSph (.h .cpp)	
	JSphAccInput (.h .cpp)	
	JSphDtFixed (.h .cpp)	
	JSphVisco (.h .cpp)	
JPartDataBi4 (.h .cpp)	JTimerClock.h	
JPartFloatBi4 (.h .cpp)	JTimeOut (.h .cpp)	
JPartOutBi4Save (.h .cpp)	Types.h	
JRadixSort (.h .cpp)		
JRangeFilter (.h .cpp)	JSphCpu (.h .cpp)	JSphGpu (.h .cpp)
JReadDatafile (.h .cpp)		JSphGpu_ker (.h .cu)
JSpaceCtes (.h .cpp)	JSphCpuSingle (.h .cpp)	JSphGpuSingle (.h .cpp)
JSpaceEParms (.h .cpp)		
JSpaceParts (.h .cpp)	JSphTimersCpu.h	JSphTimersGpu.h
JSpaceProperties (.h .cpp)		
JTimer.h	JCellDivCpu (.h .cpp)	JCellDivGpu (.h .cpp)
JTimerCuda.h		JCellDivGpu_ker (.h .cu)
TypesDef.h		
	JCellDivCpuSingle (.h .cpp)	JCellDivGpuSingle (.h .cpp)
JFormatFiles2.h		JCellDivGpuSingle_ker (.h .cu)
JSphMotion.h		
JXml.h	JArraysCpu (.h .cpp)	JArraysGpu (.h .cpp)
JWaveGen.h		JBlockSizeAuto (.h .cpp)

Exclusive source code files of DualSPHysics that implements the SPH solver.

3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp)	main.cpp	
FunctionsMath (.h .cpp)	JCfgRun (.h .cpp)	
JMatrix4.h	JPartsLoad4 (.h .cpp)	
JMeanValues (.h .cpp)	JPartsOut (.h .cpp)	
JObject (.h .cpp)	JSaveDt.cpp (.h .cpp)	
JObjectGpu (.h .cpp)	JSph (.h .cpp)	
JPartDataBi4 (.h .cpp)	JSphAccInput (.h .cpp)	
JPartFloatBi4 (.h .cpp)	JSphDtFixed (.h .cpp)	
JPartOutBi4Save (.h .cpp)	JSphVisco (.h .cpp)	
JRadixSort (.h .cpp)	JTimerClock.h	
JRangeFilter (.h .cpp)	JTimeOut (.h .cpp)	
JSpaceProperties (.h .cpp)	Types.h	
JTimer.h	JSphCpu (.h .cpp)	JSphGpu (.h .cpp)
JTimerCuda.h		JSphGpu_ker (.h .cu)
TypesDef.h	JSphCpuSingle (.h .cpp)	JSphGpuSingle (.h .cpp)
	JSphTimersCpu.h	JSphTimersGpu.h
JFormatFiles2.h	JCellDivCpu (.h .cpp)	JCellDivGpu (.h .cpp)
JSphMotion.h		JCellDivGpu_ker (.h .cu)
JXml.h	JCellDivCpuSingle (.h .cpp)	JCellDivGpuSingle (.h .cpp)
JWaveGen.h		JCellDivGpuSingle_ker (.h .cu)
	JArraysCpu (.h .cpp)	JArraysGpu (.h .cpp)
		JBlockSizeAuto (.h .cpp)

Code used in CPU and GPU implementation.

Exclusive code for CPU implementation.

Exclusive code for GPU.

3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp)	main.cpp	
FunctionsMath (.h .cpp)	JCfgRun (.h .cpp)	
JMatrix4.h	JPartsLoad4 (.h .cpp)	
JMeanValues (.h .cpp)	JPartsOut (.h .cpp)	
JObject (.h .cpp)	JSaveDt.cpp (.h .cpp)	
JObjectGpu (.h .cpp)	**JSph (.h .cpp)**	
JPartDataBi4 (.h .cpp)	JSphAccInput (.h .cpp)	
JPartFloatBi4 (.h .cpp)	JSphDtFixed (.h .cpp)	
JPartOutBi4Save (.h .cpp)	JSphVisco (.h .cpp)	
JRadixSort (.h .cpp)	JTimerClock.h	
JRangeFilter (.h .cpp)	JTimeOut (.h .cpp)	
	Types.h	
JSpaceProperties (.h .cpp)	**JSphCpu (.h .cpp)**	**JSphGpu (.h .cpp)**
JTimer.h		**JSphGpu_ker (.h .cu)**
JTimerCuda.h	**JSphCpuSingle (.h .cpp)**	**JSphGpuSingle (.h .cpp)**
TypesDef.h		
	JSphTimersCpu.h	JSphTimersGpu.h
JFormatFiles2.h	**JCellDivCpu (.h .cpp)**	**JCellDivGpu (.h .cpp)**
JSphMotion.h		**JCellDivGpu_ker (.h .cu)**
JXml.h	**JCellDivCpuSingle (.h .cpp)**	**JCellDivGpuSingle (.h .cpp)**
JWaveGen.h		**JCellDivGpuSingle_ker (.h .cu)**
	JArraysCpu (.h .cpp)	JArraysGpu (.h .cpp)
		JBlockSizeAuto (.h .cpp)

Code used in CPU and GPU implementation.

Exclusive code for CPU implementation.

Exclusive code for GPU.

3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp)		main.cpp
FunctionsMath (.h .cpp)		JCfgRun (.h .cpp)
JBinaryData (.h .cpp)		JPartsLoad4 (.h .cpp)
JException (.h .cpp)		JPartsOut (.h .cpp)
JLog2 (.h .cpp)		JSaveDt.cpp (.h .cpp)
JMatrix4.h		JSph (.h .cpp)
JMeanValues (.h .cpp)		JSphAccInput (.h .cpp)
JObject (.h .cpp)		JSphDtFixed (.h .cpp)
JObjectGpu (.h .cpp)		JSphVisco (.h .cpp)
JPartDataBi4 (.h .cpp)		
JPartFloatBi4 (.h .cpp)		
JPartOutBi4Save (.h .cpp)		
JRadixSort (.h .cpp)		
JRangeFilter (.h .cpp)	JSph	
JReadDatafile (.h .cpp)		
JSpaceCtes (.h .cpp)	JSphCpuSingle (.h .cpp)	JSphGpuSingle (.h .cpp)
JSpaceEParms (.h .cpp)		
JSpaceParts (.h .cpp)	JSphTimersCpu.h	JSphTimersGpu.h
JSpaceProperties (.h .cpp)		
JTimer.h	JCellDivCpu (.h .cpp)	JCellDivGpu (.h .cpp)
JTimerCuda.h		JCellDivGpu_ker (.h .cu)
TypesDef.h		
	JCellDivCpuSingle (.h	JCellDivGpuSingle (.h .cpp)
JFormatFiles2.h	.cpp)	JCellDivGpuSingle_ker (.h .cu)
JSphMotion.h		
JXml.h		JArraysGpu (.h .cpp)
JWaveGen.h	JArraysCpu (.h .cpp)	JBlockSizeAuto (.h .cpp)

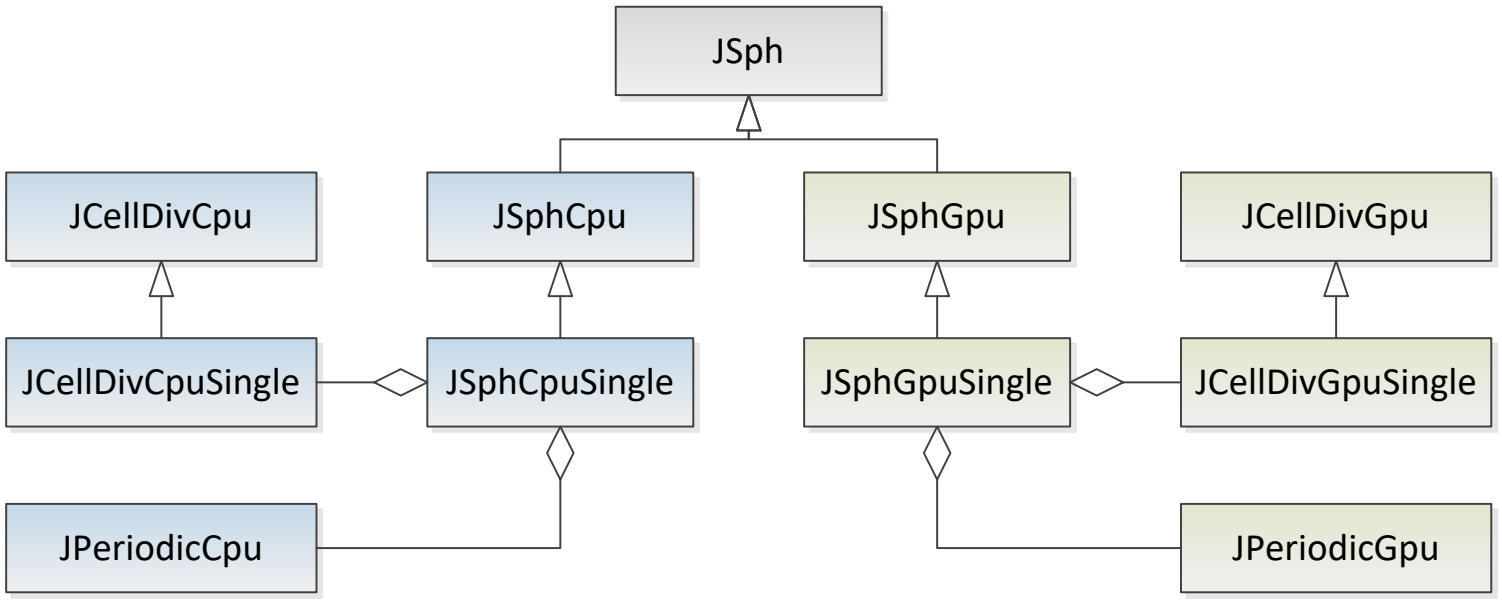
Complete description of each file in DualSPHysics Guide.

3.3. Object-Oriented Programming

DualSPHysics uses Object-Oriented Programming (OOP) to organise the code. However, the particles data is stored in arrays to improve the performance.

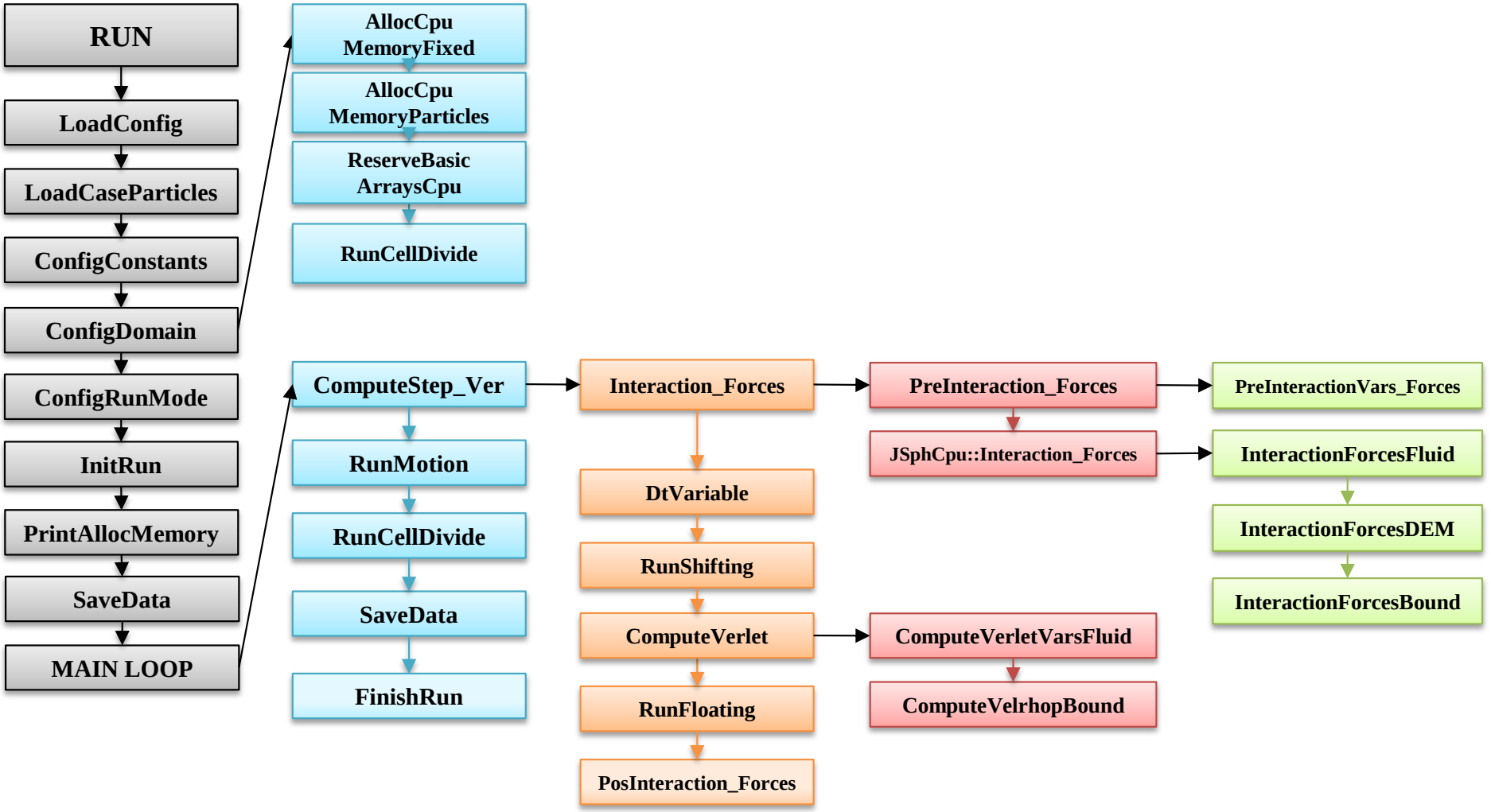
Class name matches the file name.

Diagram main classes



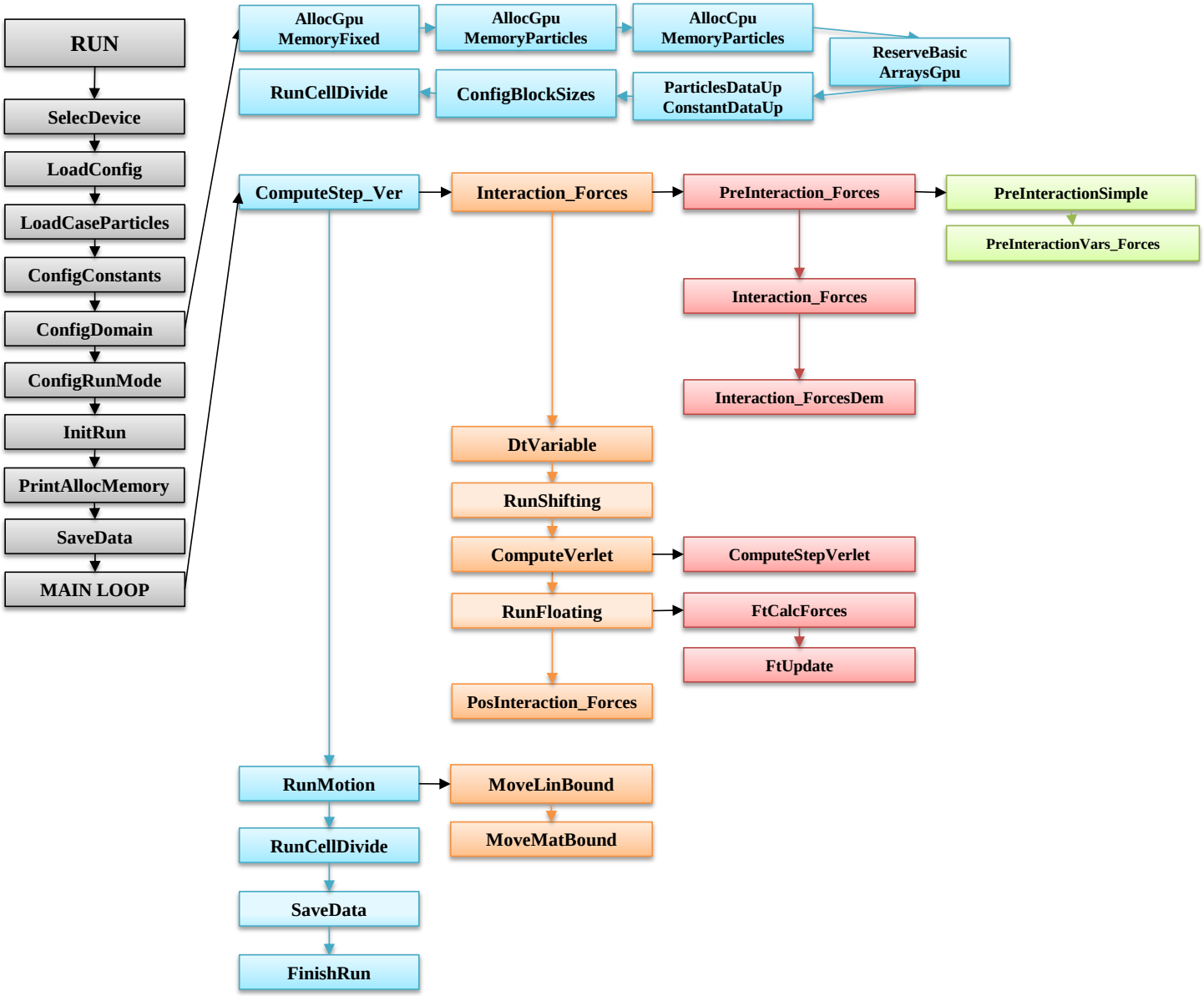
3.4. Execution diagram: CPU

CPU execution using **Verlet** algorithm (JSphCpuSingle::Run() in JSphCpuSingle.cpp).



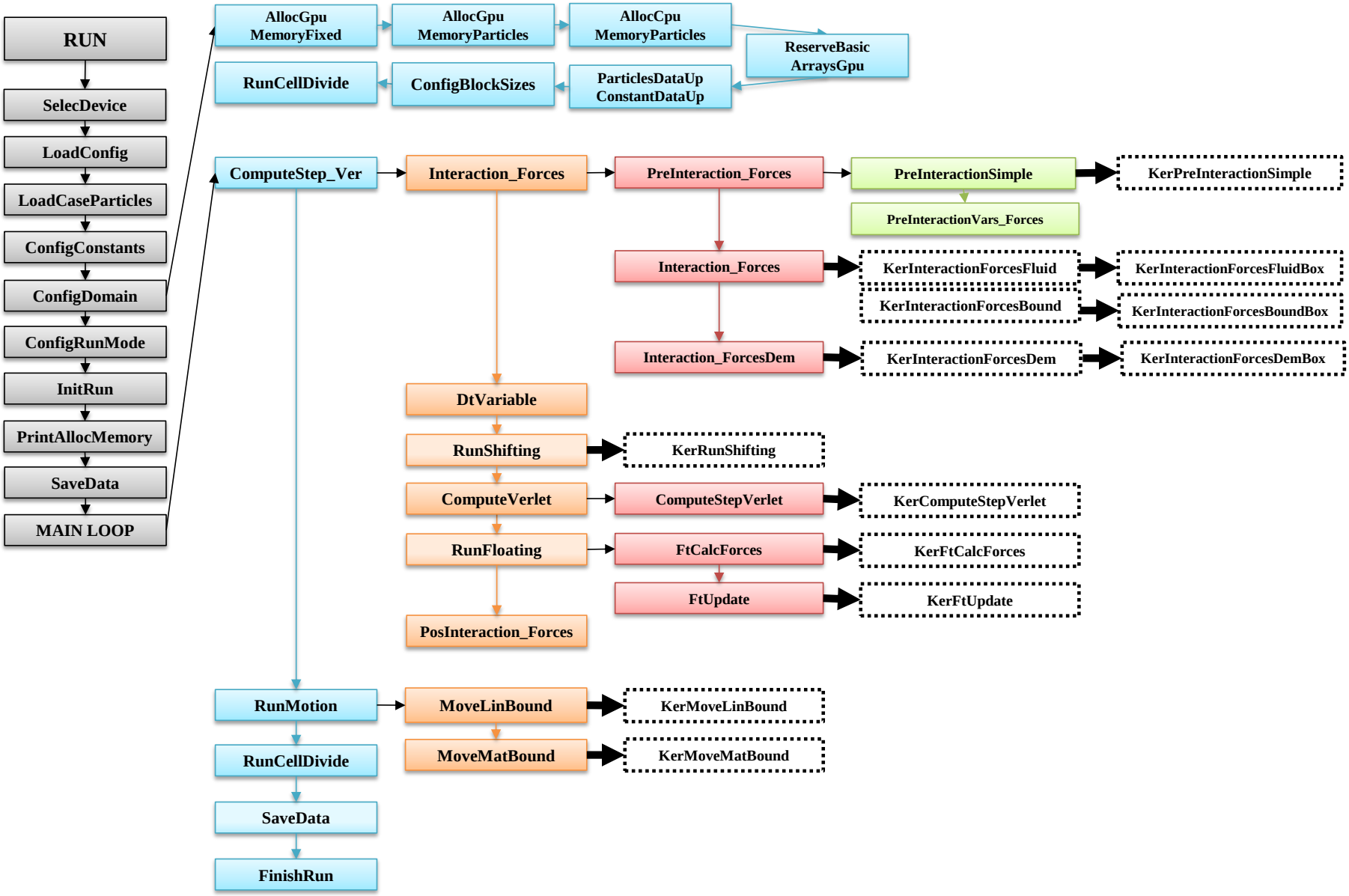
3.4. Execution diagram: GPU with CUDA kernels

GPU execution using **Verlet** algorithm (similar to CPU execution).



3.4. Execution diagram: GPU with CUDA kernels

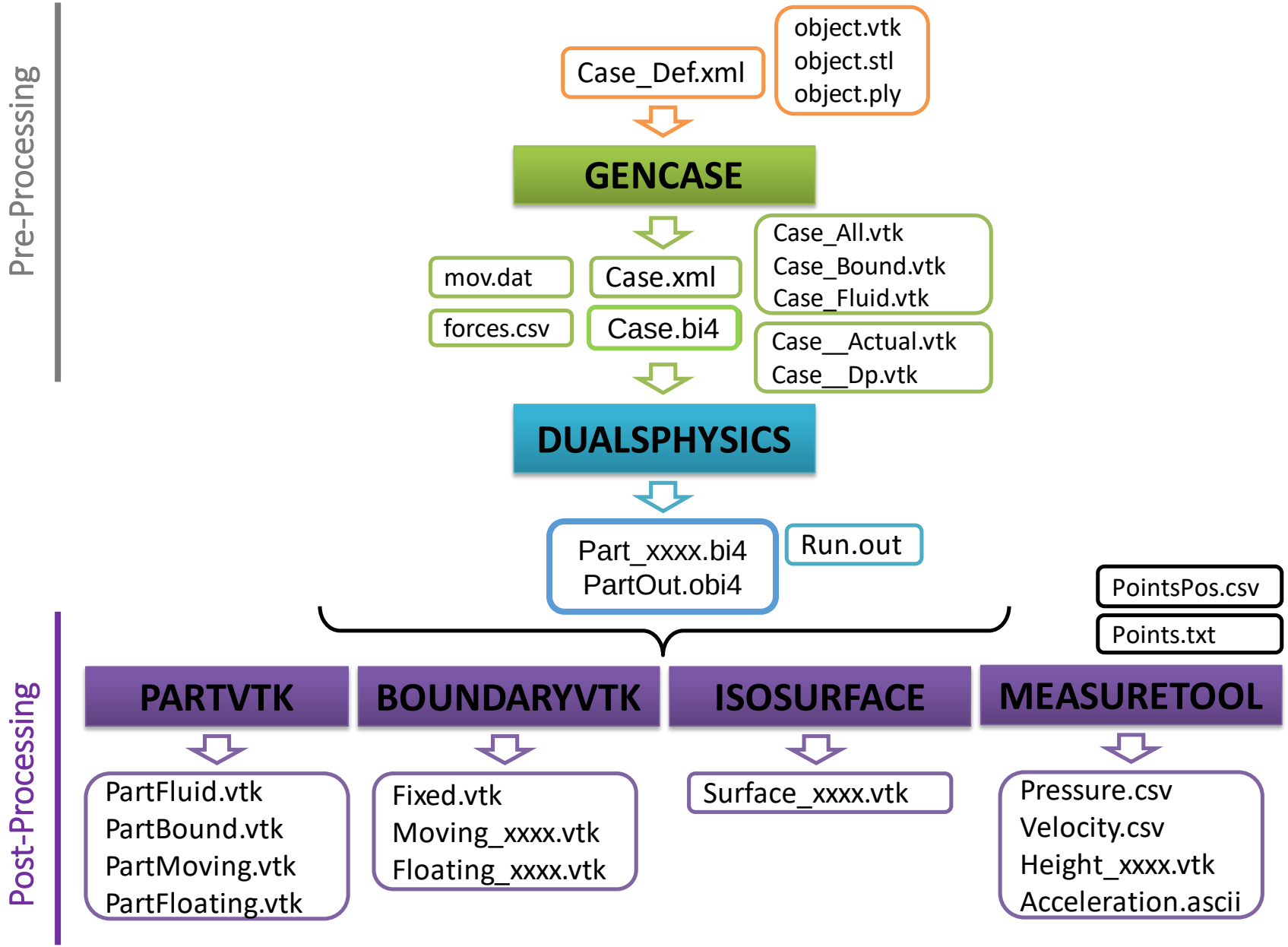
The C++ methods call CUDA kernels to execute each task on GPU.



Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0

4. Input & output files



4. Input & output files

Case_Def.xml

Case.xml

XML File

- The eXtensible Markup Language is textual data format compatible with any hardware and software.
- Information is structured and organised by using labels.
- They can be easily edited using any text editor.

Case.bi4

Part_xxxx.bi4
PartOut.obi4

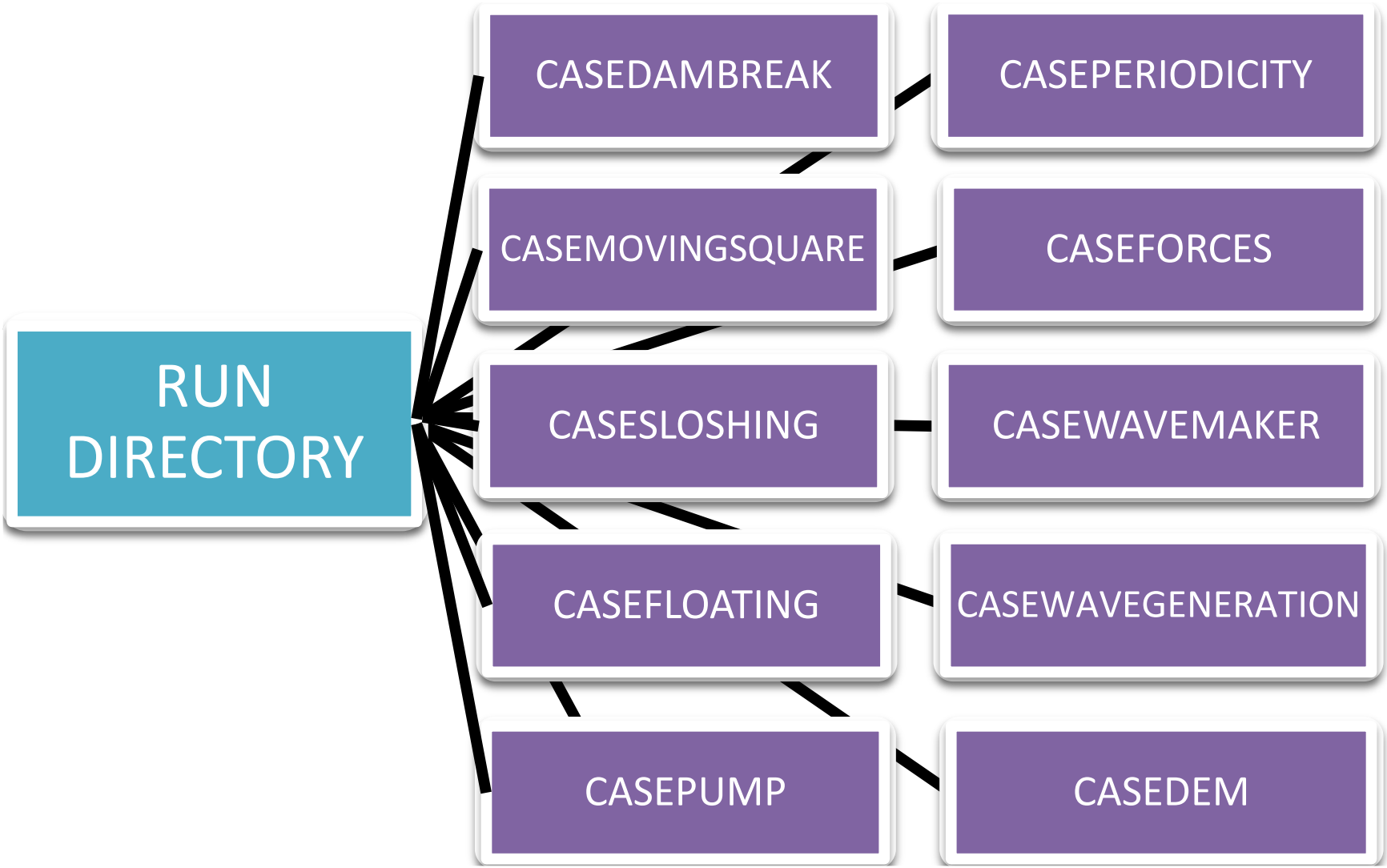
BINARY File

- Binary format consumes **six times less memory** than text format.
- Reading or writing is **several times f**aster using a binary format.
- A special code is required to read the data (JPartDataBi4.cpp/.h).
- “.bi4” is the new binary format that also includes double precision.
- The user can also define new arrays that post-processing tools can automatic manage.

Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0

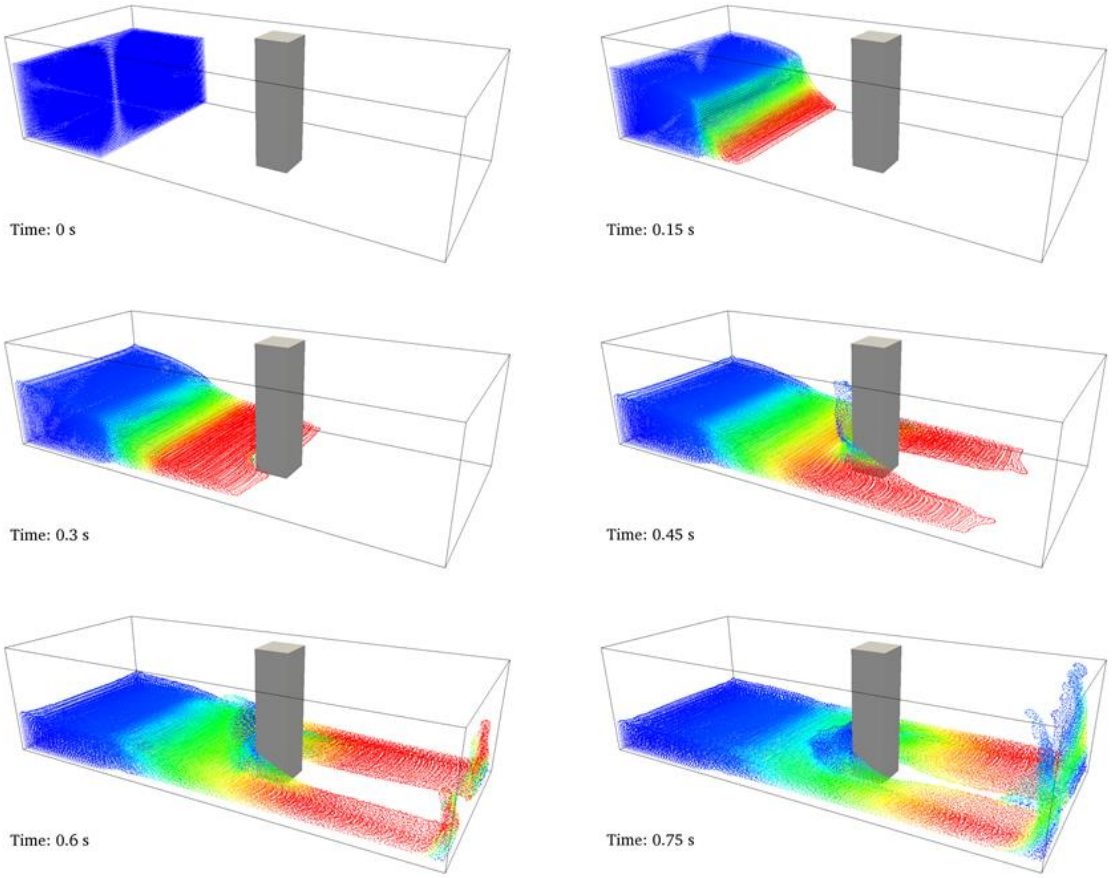
5. Test cases online



5. Test cases online

CASEDAMBREAK (3-D Dam break):

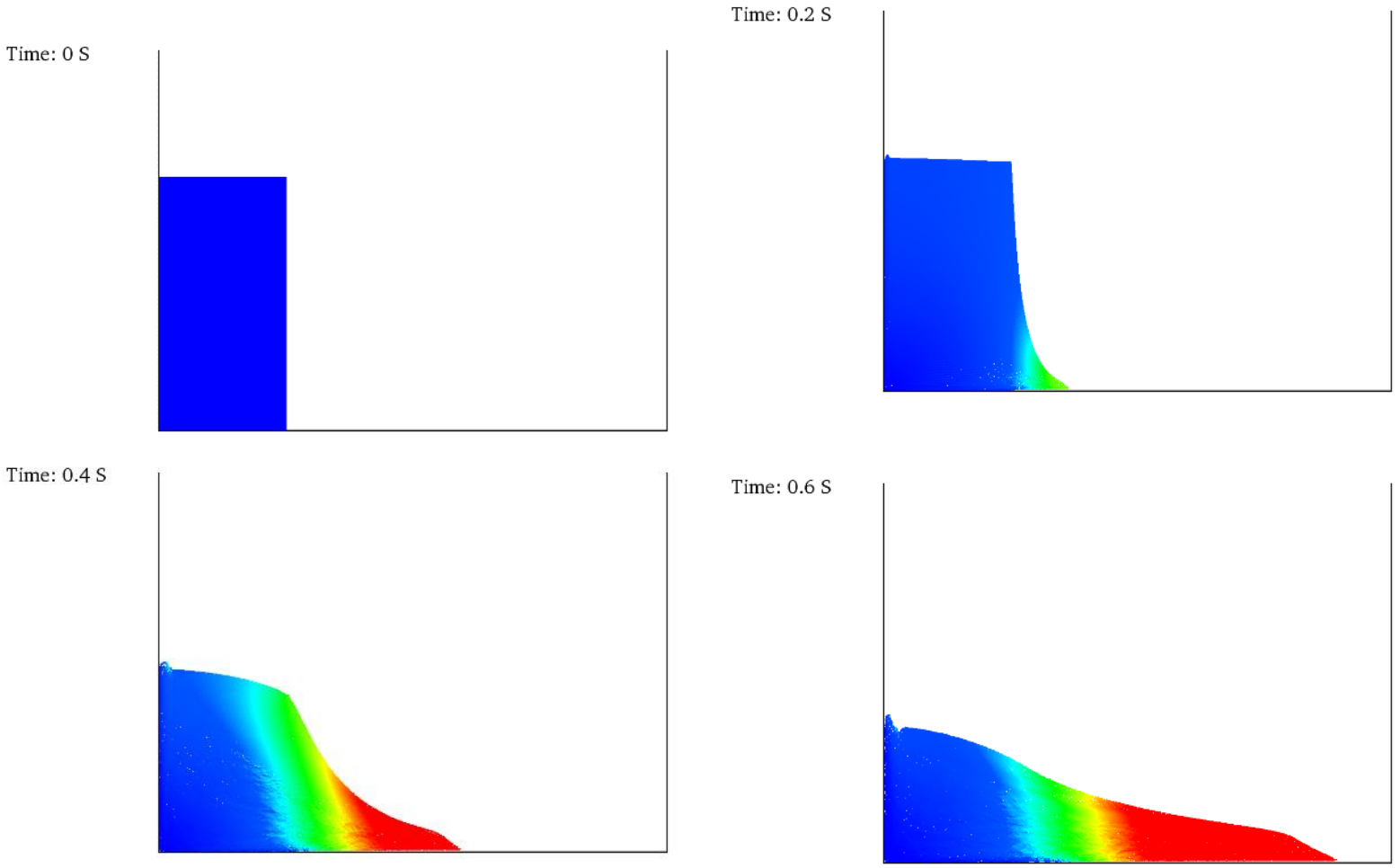
- 3-D dam break flow impacting on a structure (for validation visit Crespo et al, 2011).
- Numerical velocity, pressure and force are computed.



5. Test cases online

CASEDAMBREAK (2-D Dam break):

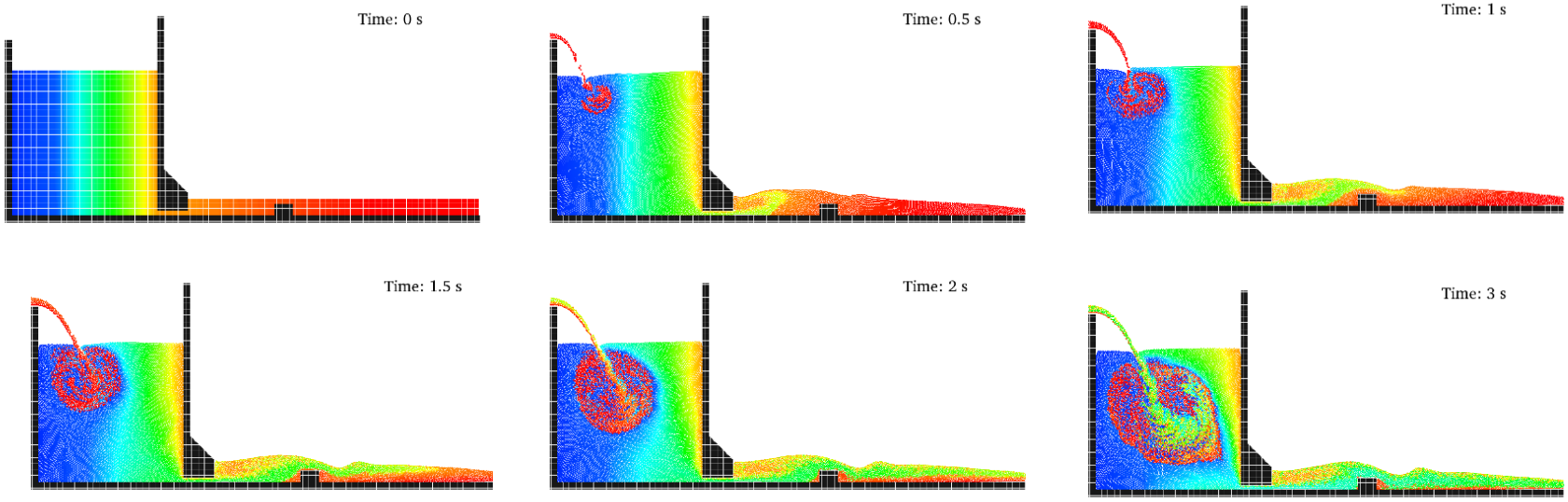
- 2-D dam break and **validation** data from (Koshizula and Oka, 1996) experiment.



5. Test cases online

CASEPERIODICITY (Periodicity in DualSPHysics):

- 2-D case periodic example.
- Particles leave the domain through the right.
- and are introduced through the left with the same properties but where the vertical position can change (with an increase of +0.3 in Z-position).

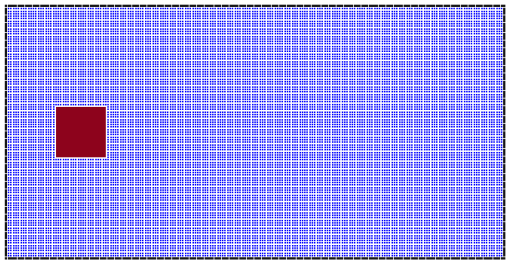


5. Test cases online

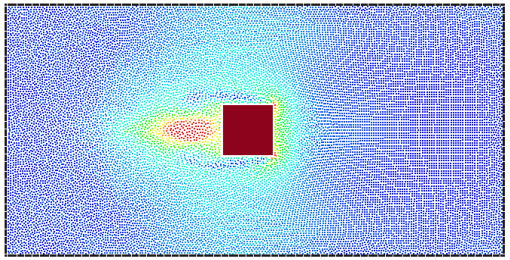
CASEMOVINGSQUARE (Motion in DualSPHysics):

- 2-D case with square that moves with rectilinear motion.
- Shifting is used for this internal flow (no need to detect free surface).

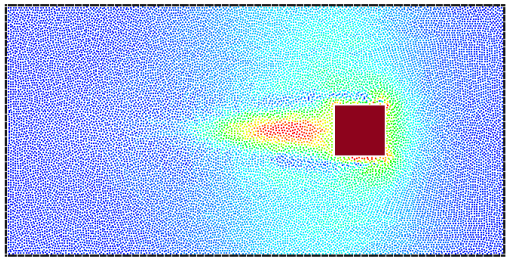
Time: 0.00 s



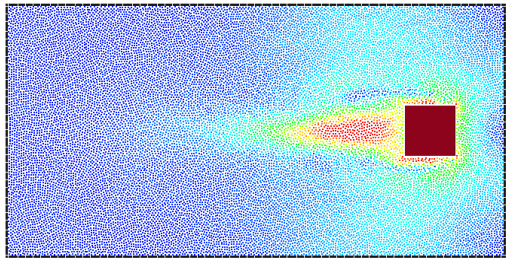
Time: 1.20 s



Time: 2.00 s



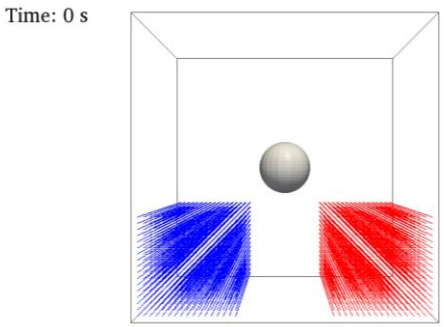
Time: 2.50 s

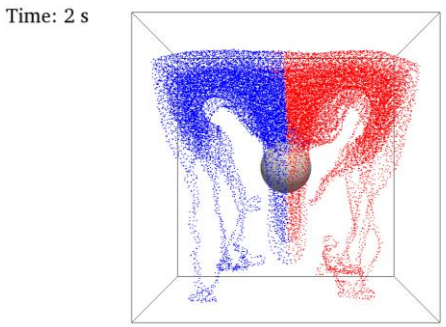


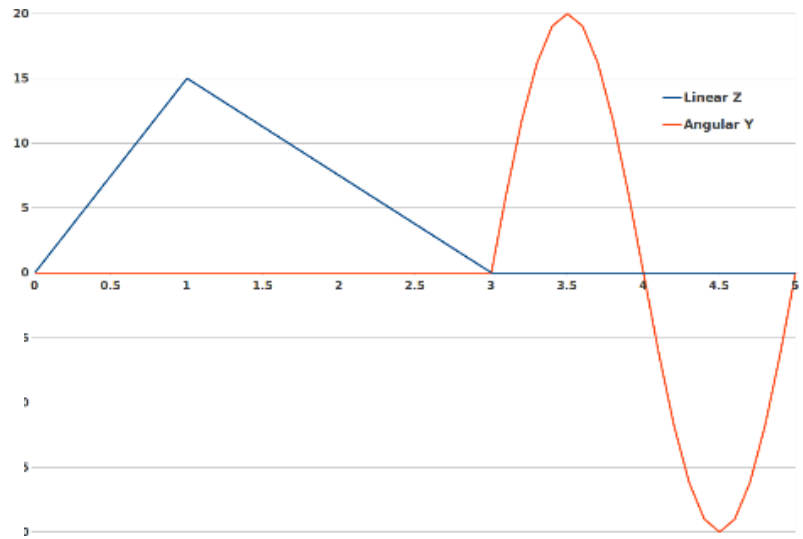
5. Test cases online

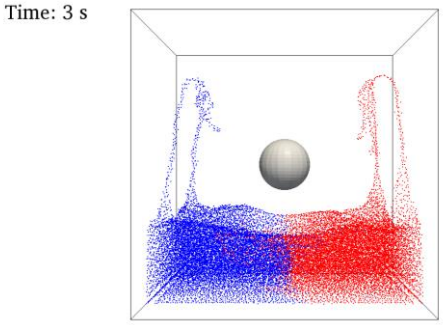
CASEFORCES (External acceleration applied to fluid):

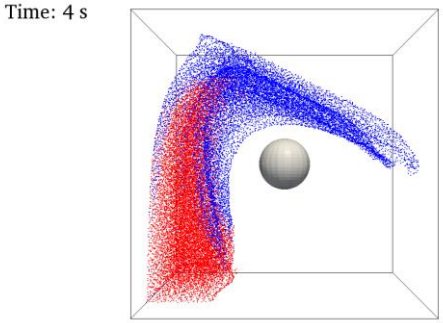
- External acceleration is loaded from a file and applied to different volumes of fluid (linear and angular).











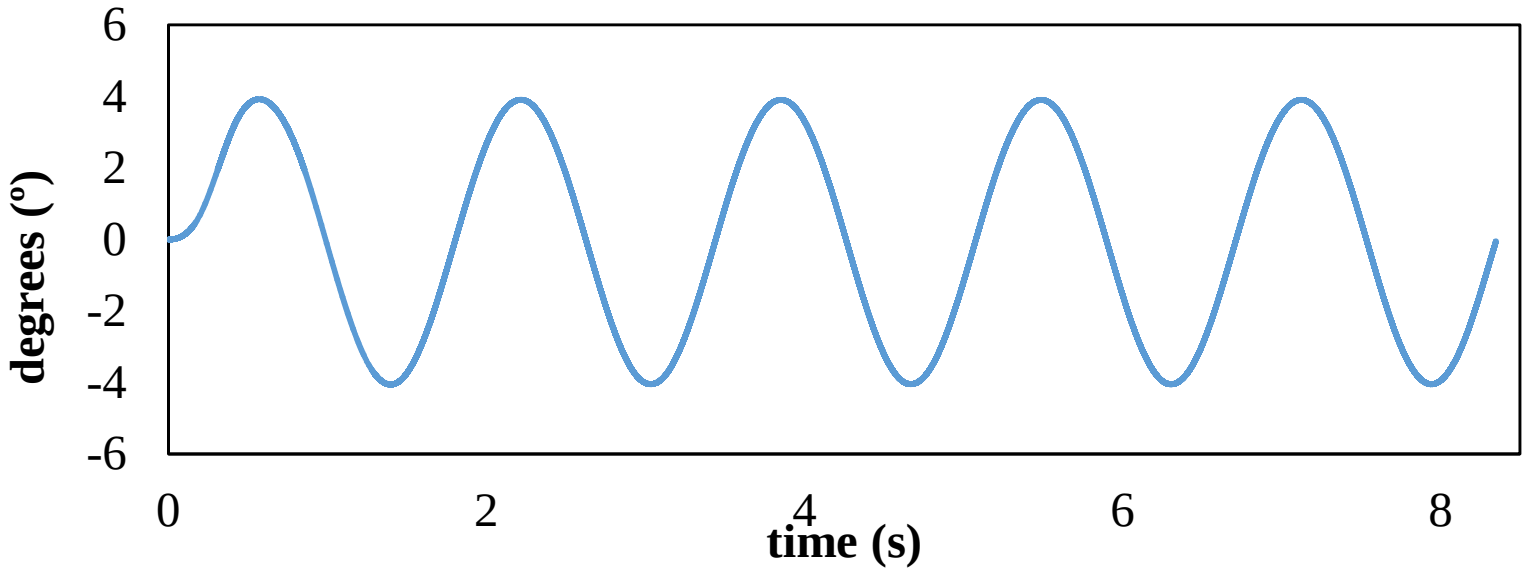
5. Test cases online

CASESLOSHING (Sloshing tank):

A 2-D sloshing tank is simulated in two different ways:

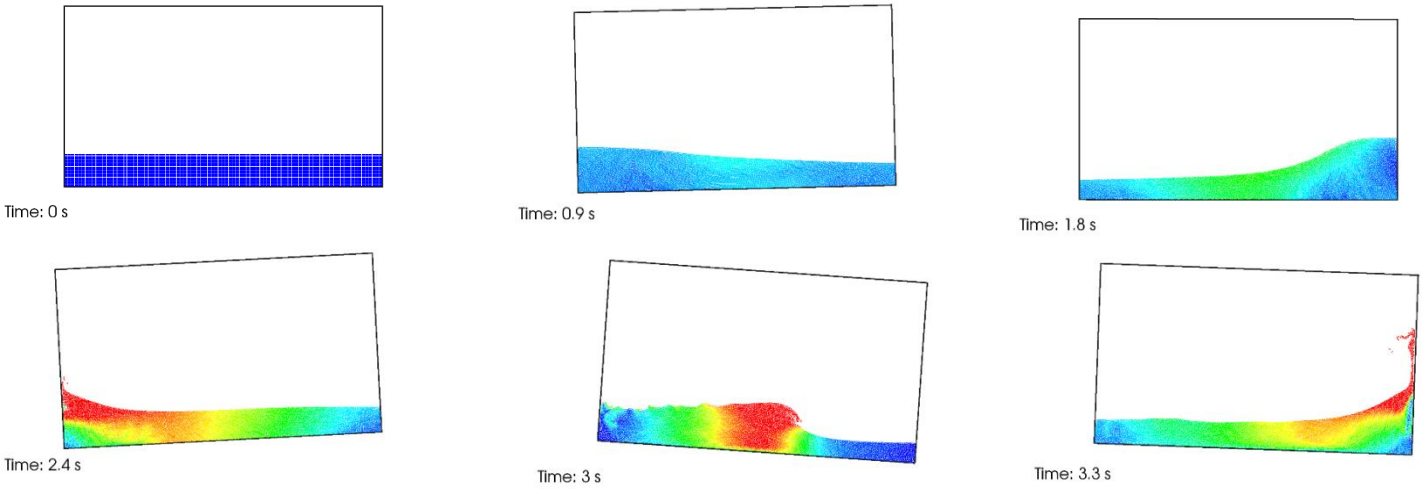
a) External acceleration is loaded from the CSV file *CaseSloshingAccData.csv* and applied to the fluid particles.

b) Rotational movement is applied to the tank using prescribed rotation defined in the text file *CaseSloshingMotionData.dat*.

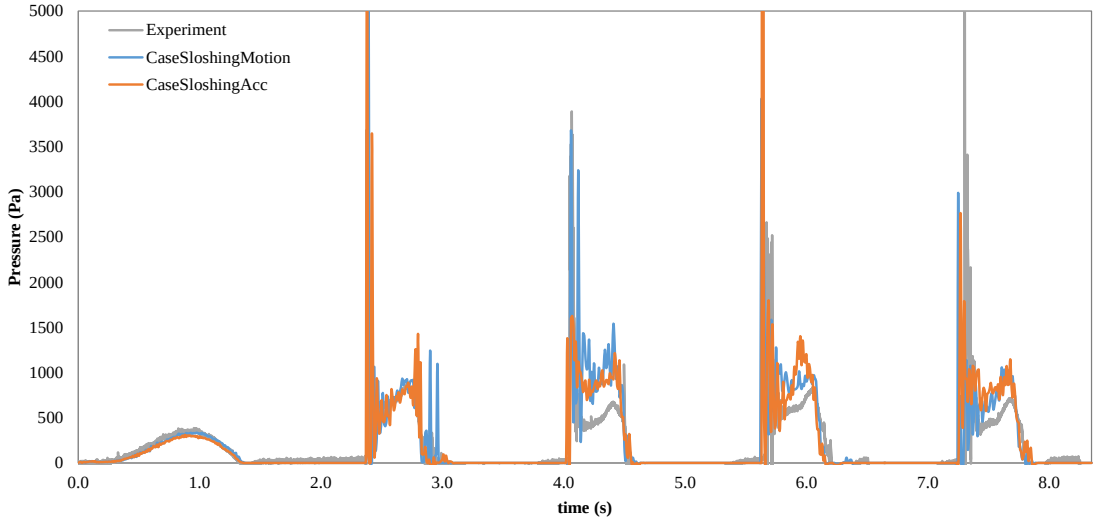


5. Test cases online

CASESLOSHING (Sloshing tank):



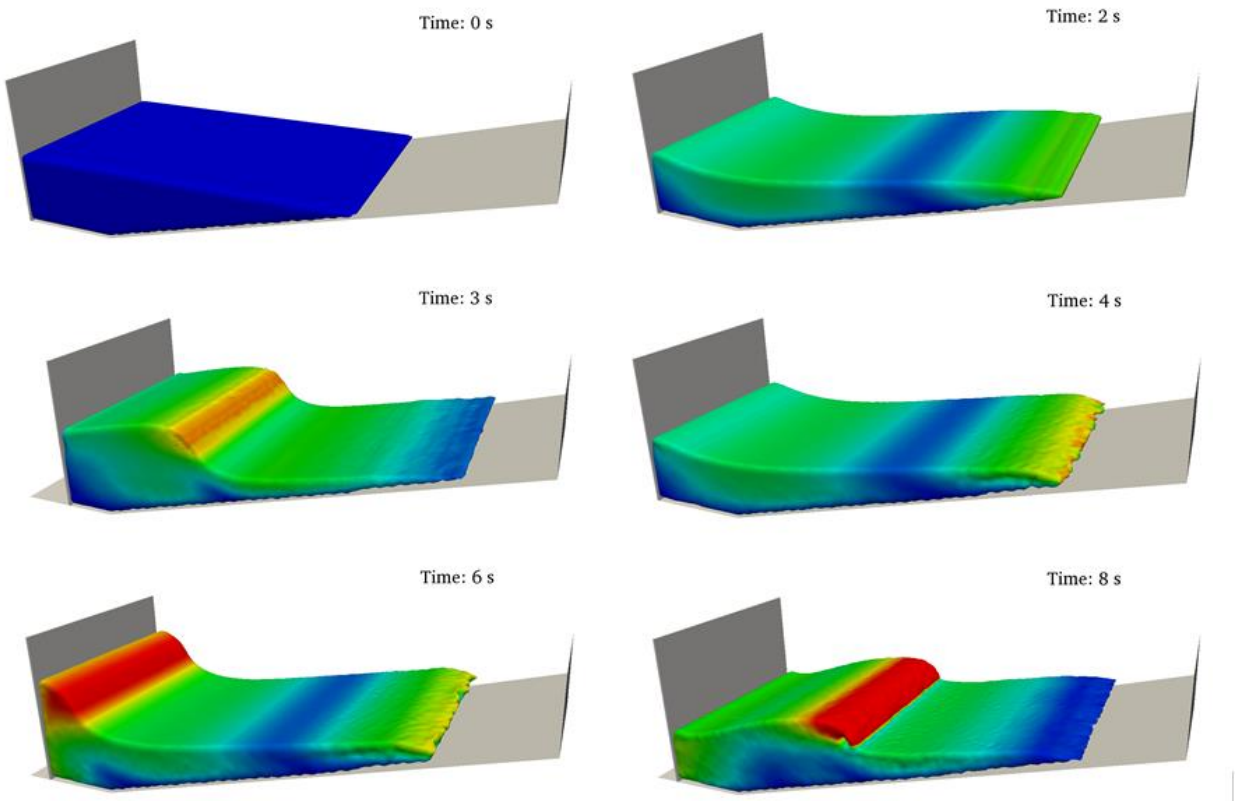
Validation with SPHERIC Benchmark #10 where pressure is computed.



5. Test cases online

CASEWAVEMAKER (3-D Wavemaker):

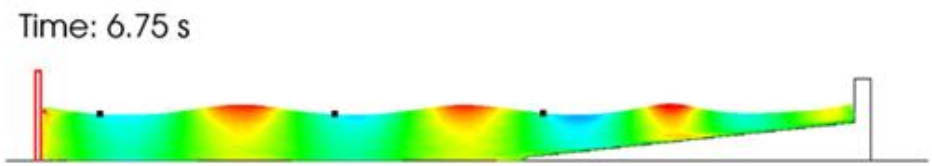
- 3-D tank with Periodicity in Y direction and piston with sinusoidal movement.
- Delta-SPH and Shifting have been used.

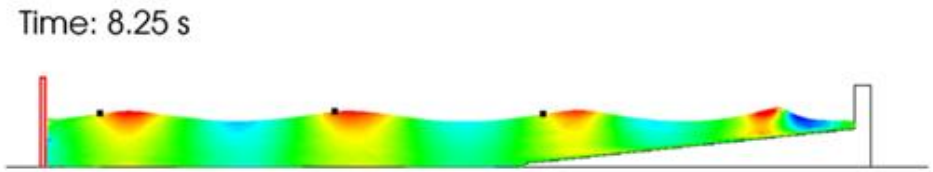


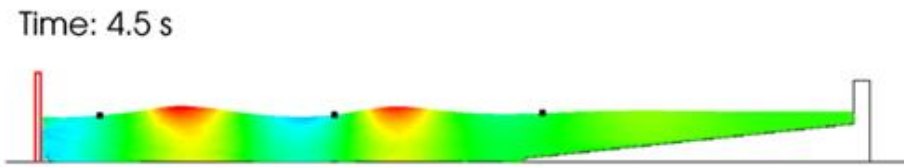
5. Test cases online

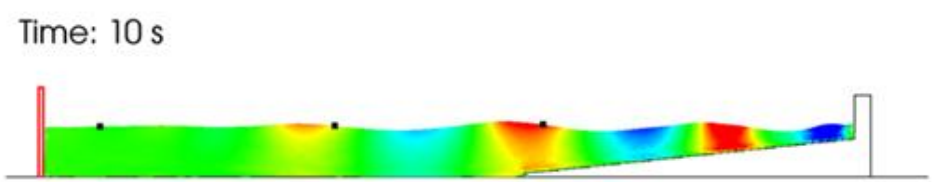
CASEWAVEMAKER (2-D Wavemaker):

- 2-D tank with piston motion loaded from external file and external structure (STL).
- Validation data from CIEMito experiment: numerical computation of wave surface elevation and force exerted onto the wall.





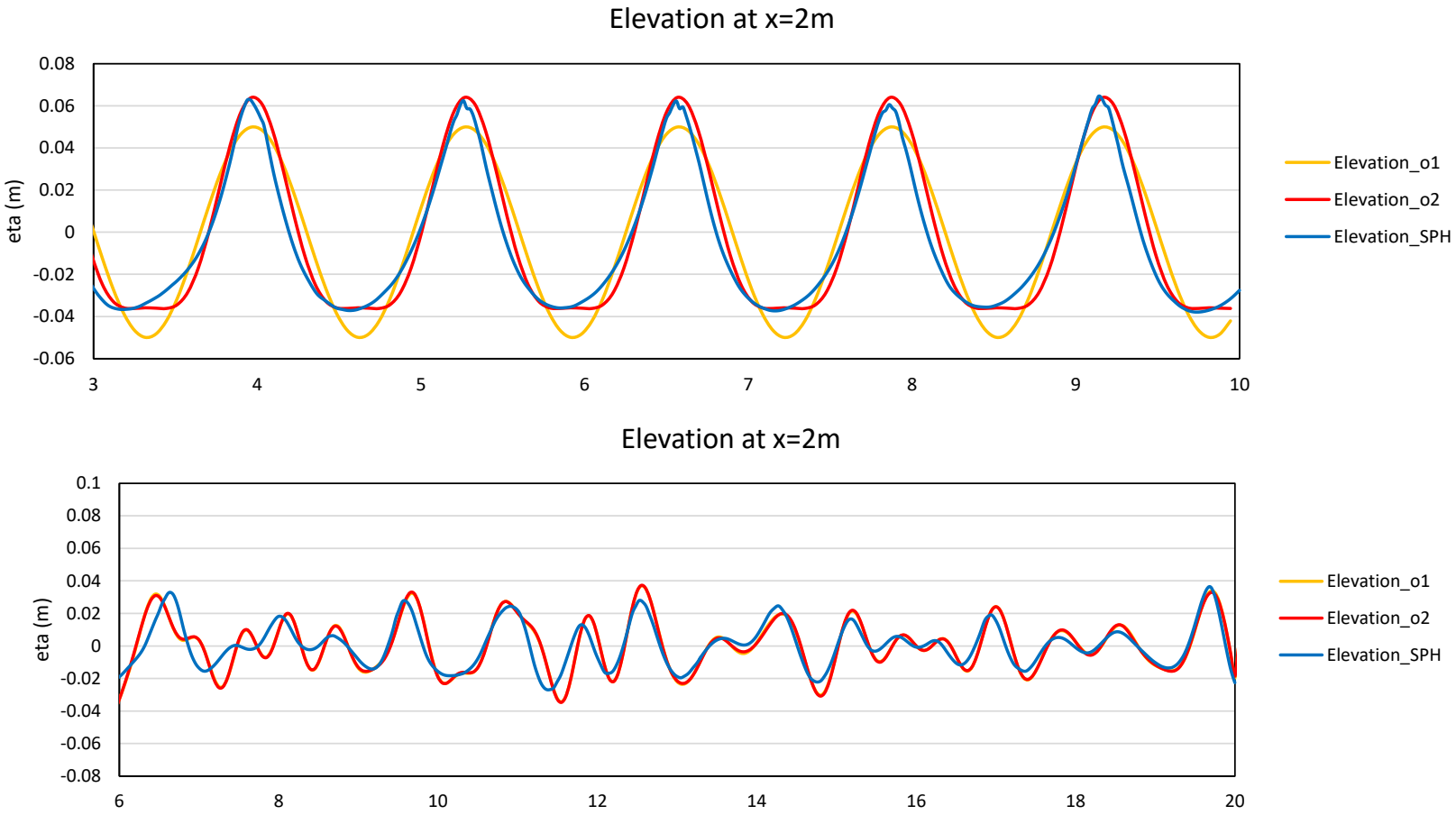




5. Test cases online

CASEWAGENERATION (Automatic wave generation):

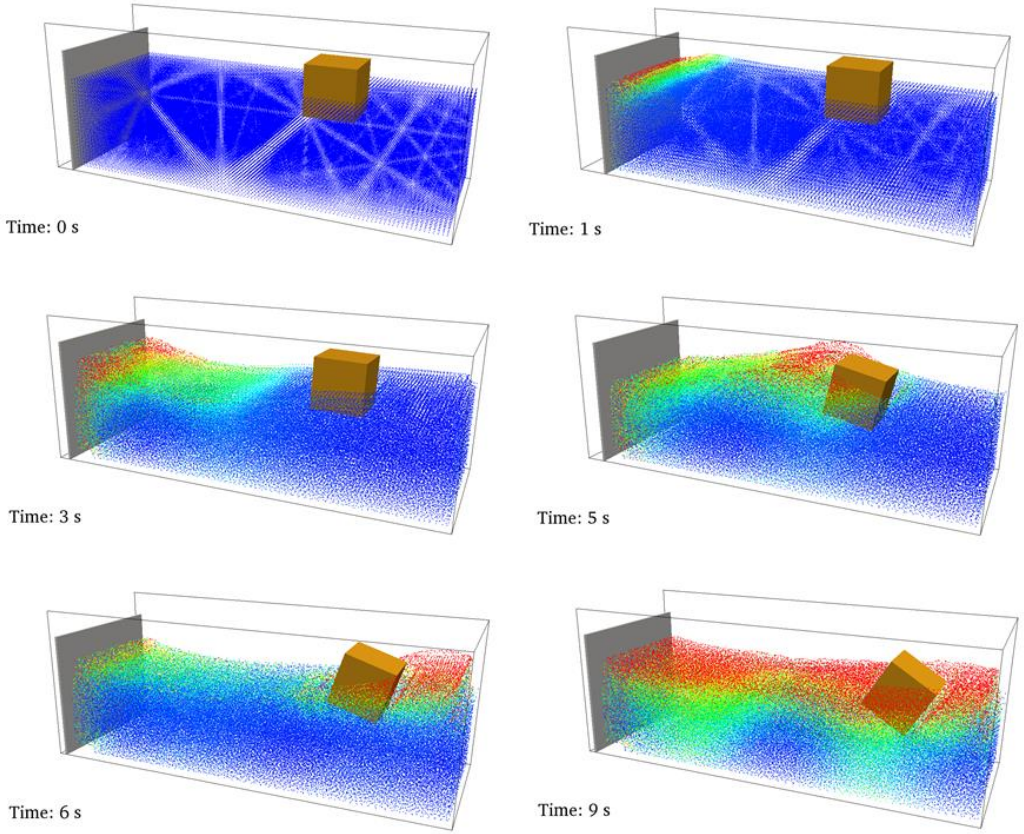
- 2-D automatic generation of **regular** waves (H,T,d) and comparison with wave theory.
- 2-D automatic generation of **irregular** waves (H_s,T_p,d) and comparison with wave theory.



5. Test cases online

CASEFLOATING (Floating body):

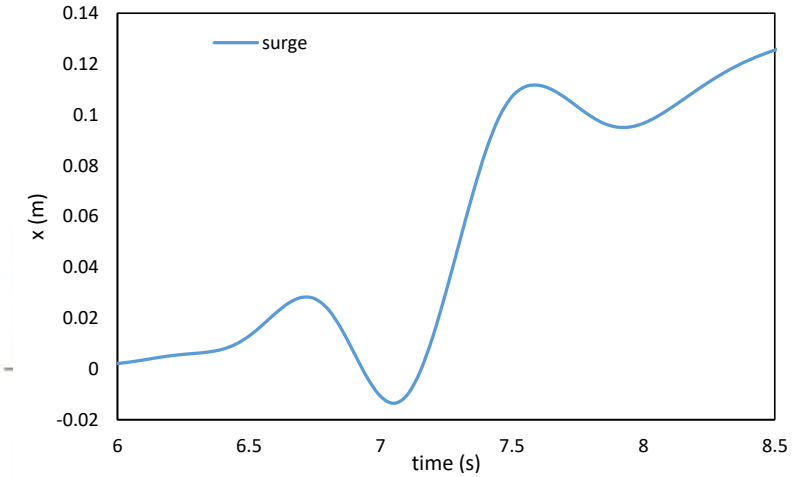
- 3-D floating box in a wave tank with Periodicity in Y direction and piston with sinusoidal movement.

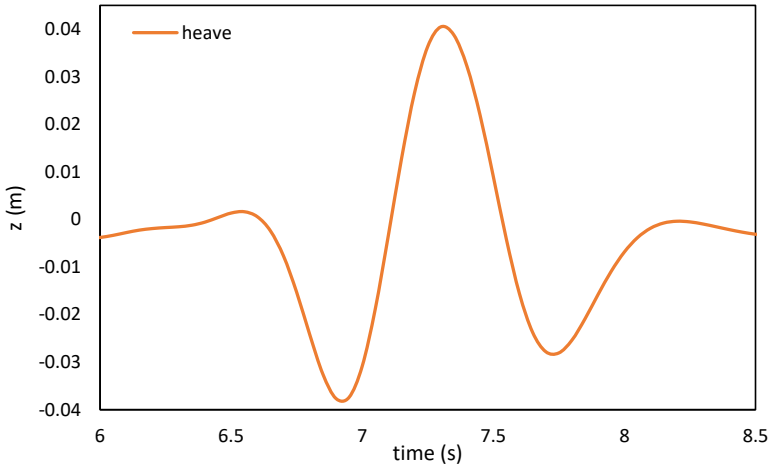


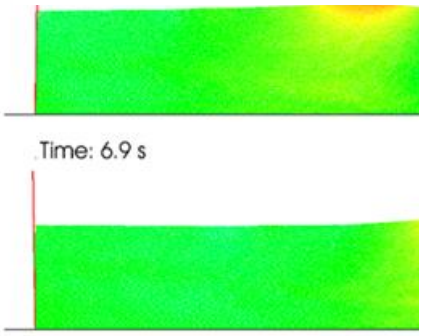
5. Test cases online

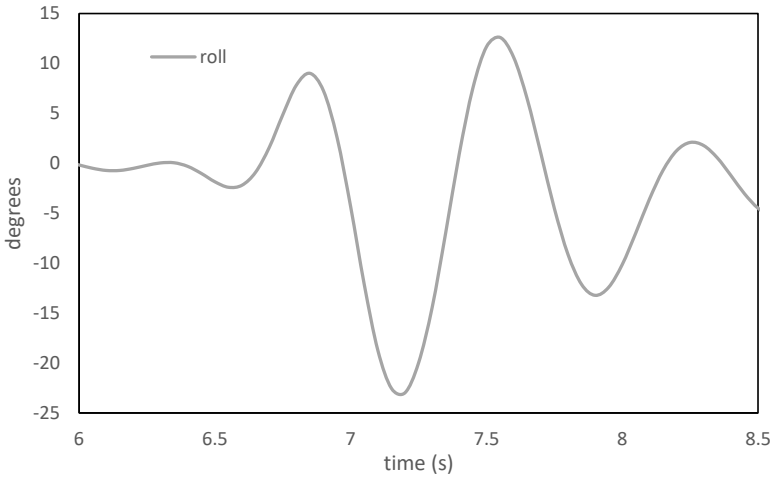
CASEFLOATING (Floating body with external motion):

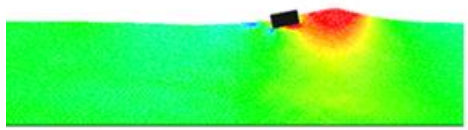
- 2-D floating box under action of non-linear waves in a tank with flap that reads rotational motion from an external file.
- Laminar+SPS viscosity, validation data from (Hadzic et al., 2005).







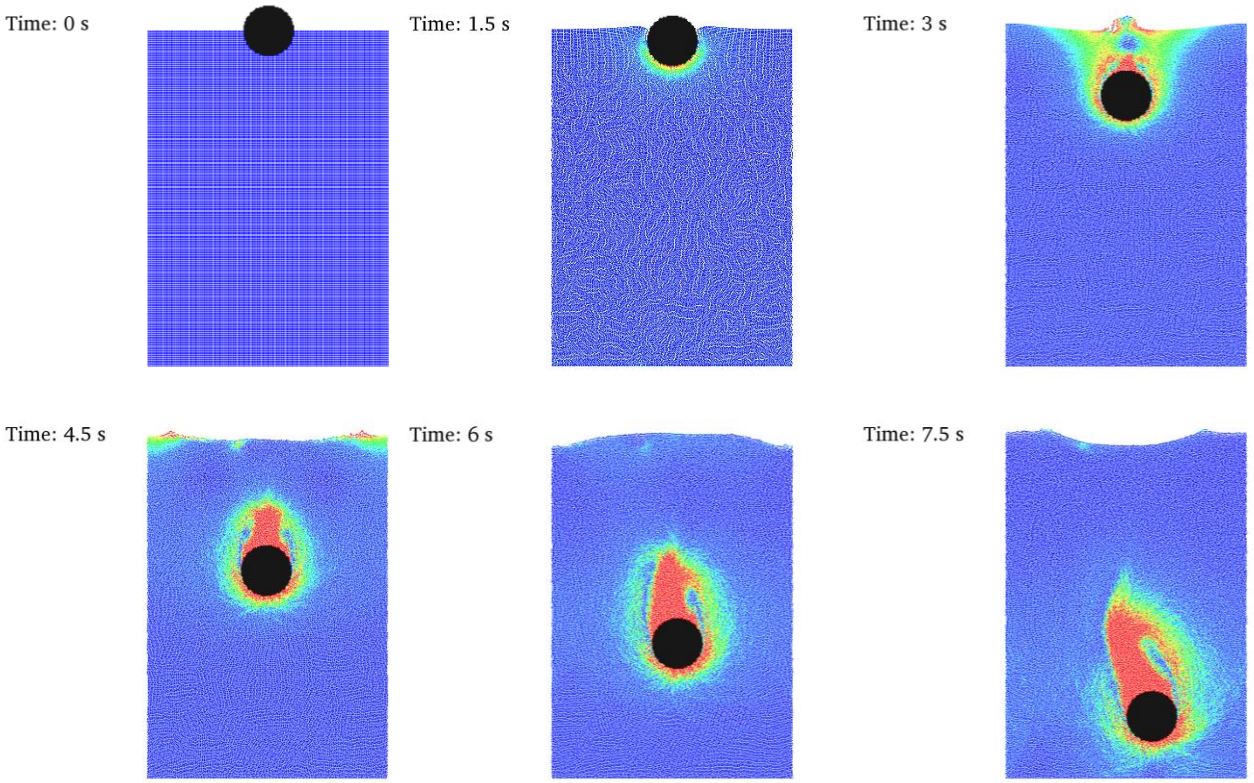




5. Test cases online

CASEFLOATING (Falling sphere under gravity):

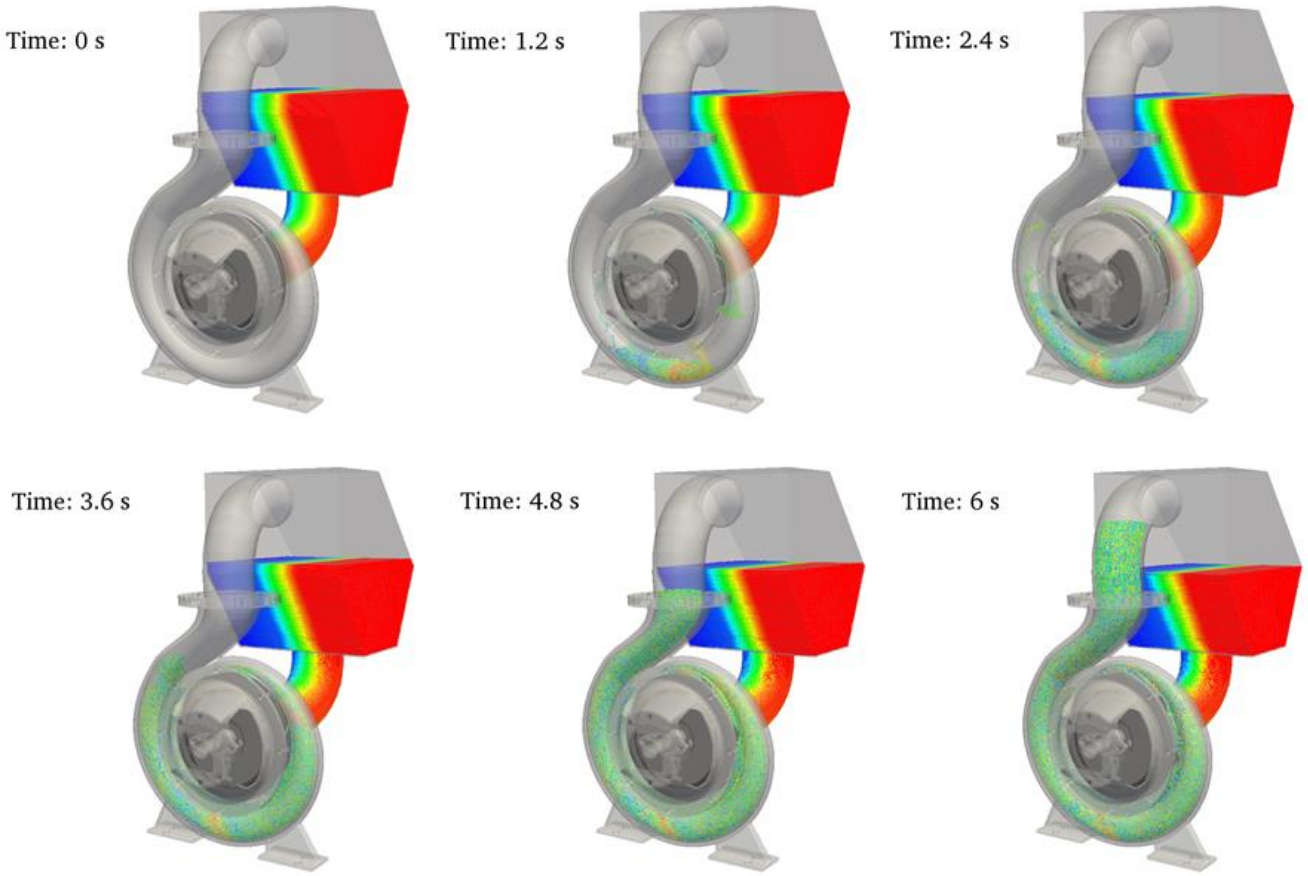
- 2-D falling sphere that uses laminar+SPS viscosity.
- Validation data from (Fekken, 2004) and (Moyo and Greenhow, 2000).



5. Test cases online

CASEPUMP (External CAD file):

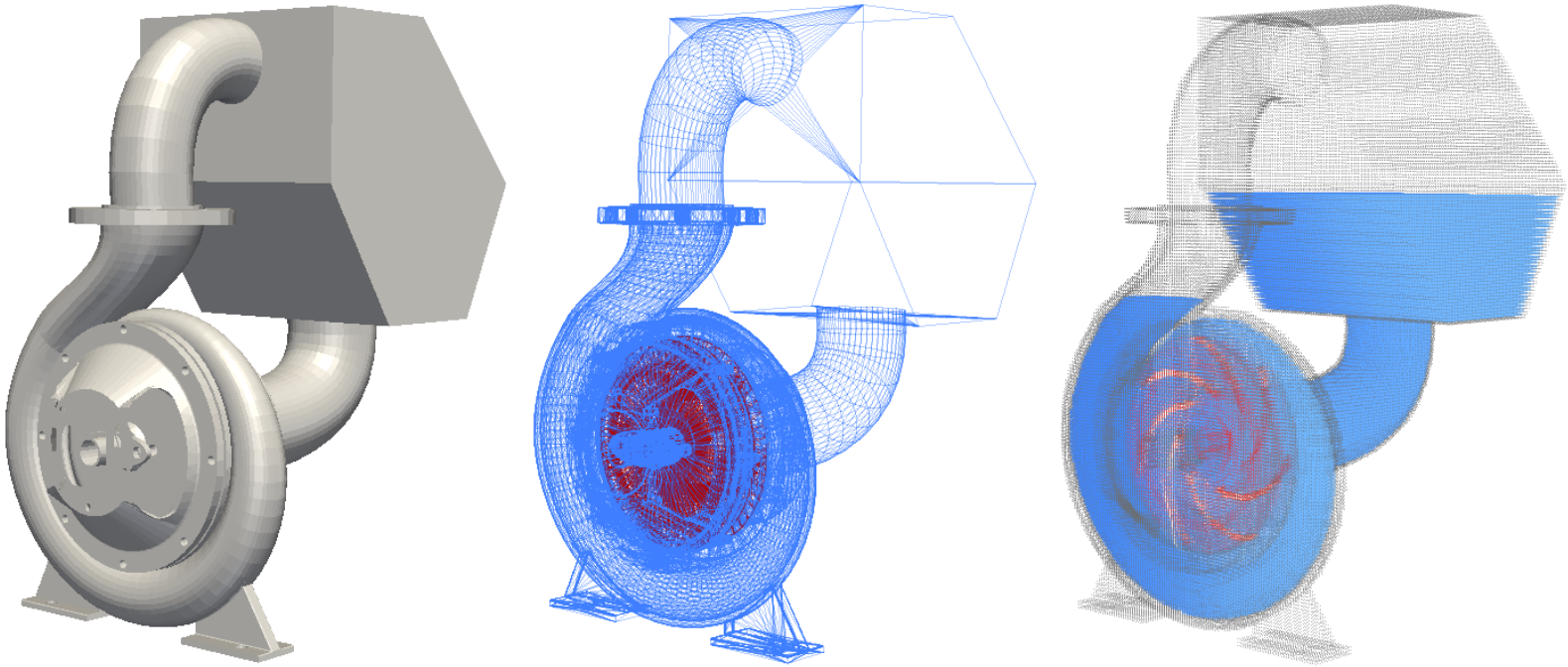
- 3-D external geometries are imported (STL) and rotational movement is imposed.



5. Test cases online

CASEPUMP (External CAD file):

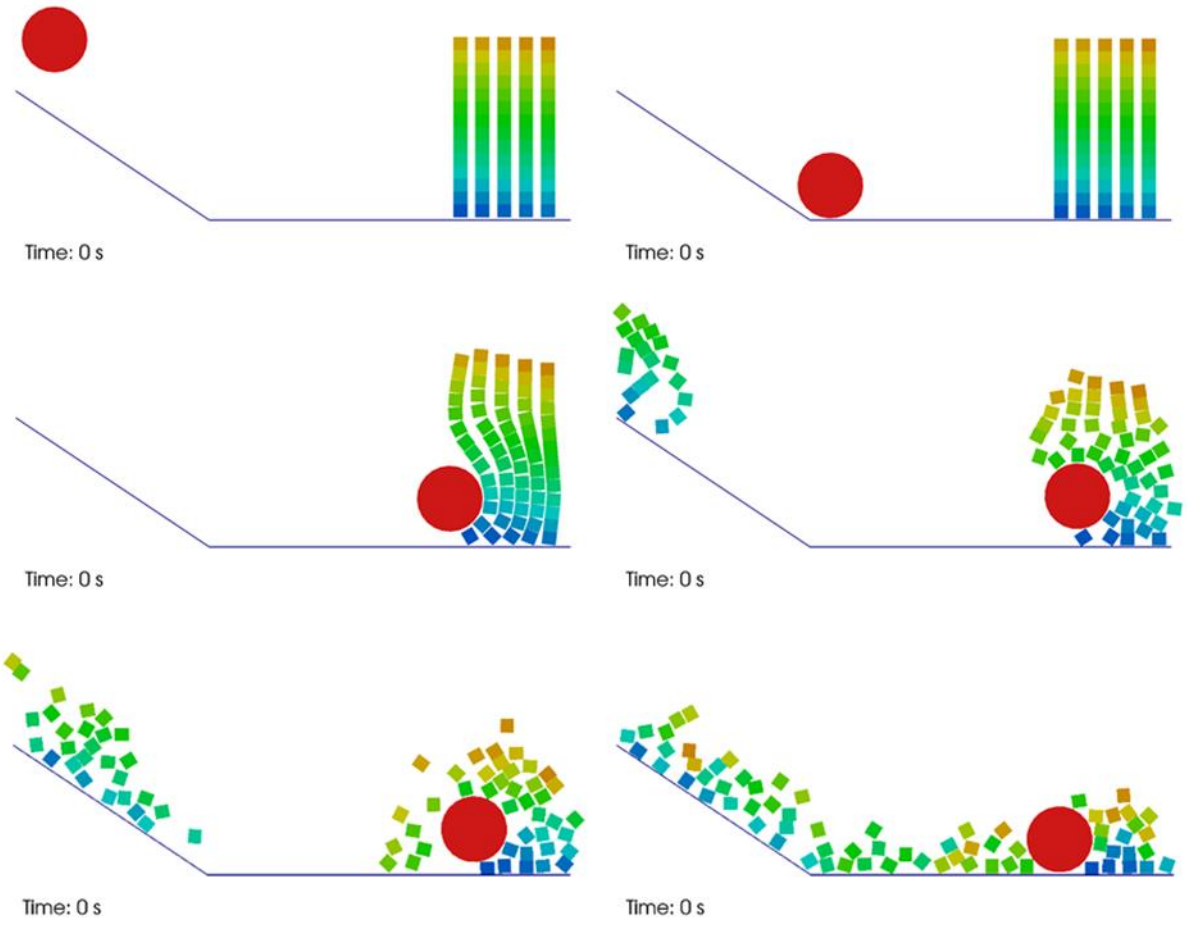
- 3-D external geometries are imported (STL) and rotational movement is imposed.



5. Test cases online

CASEDEM (DEM model example):

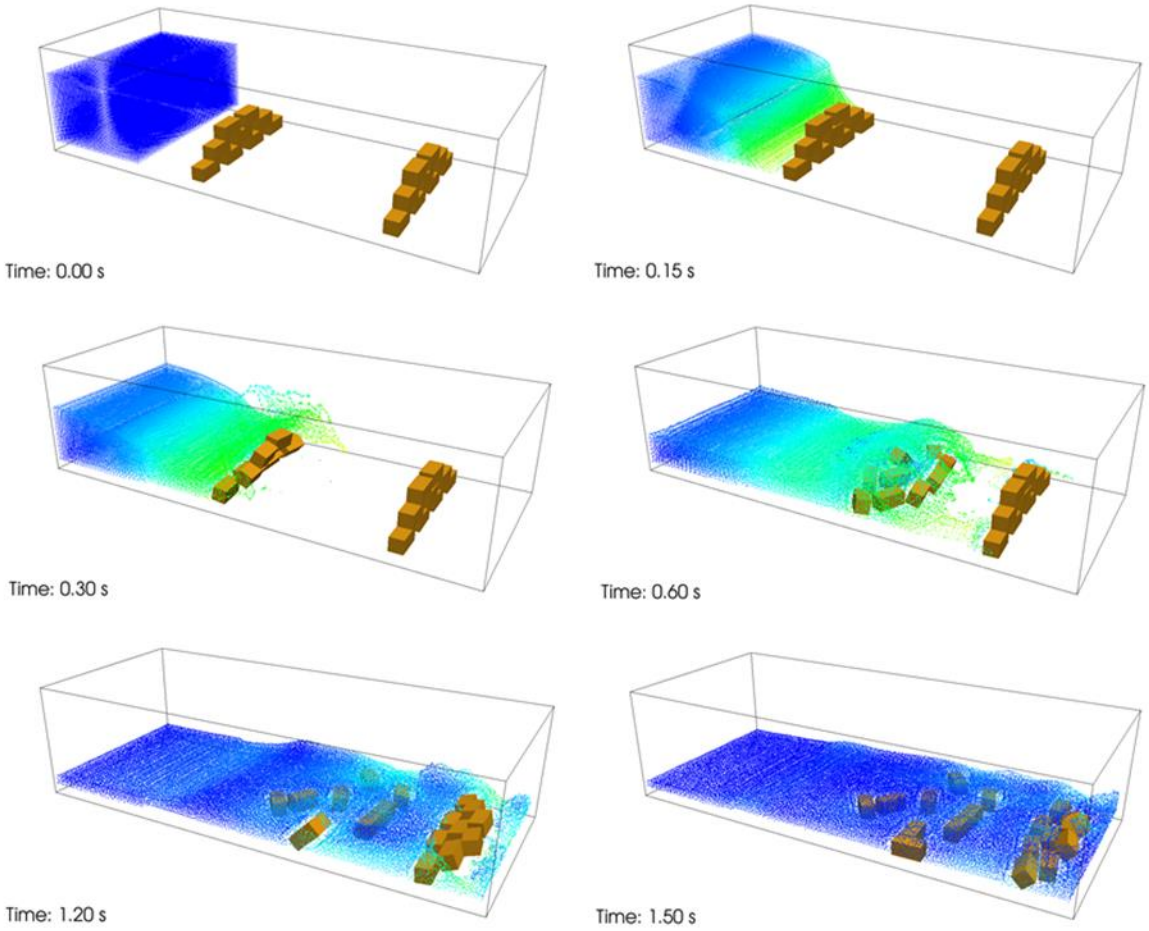
- 2-D case only with DEM of a ball that impacts with blocks.
- Example without fluid particles.



5. Test cases online

CASEDEM (3-D Dam break with DEM):

- 3-D dam break and blocks where interaction between blocks and with walls used DEM and properties of materials (Canelas et al, 2016).



5. Test cases online

CASETWOPHASES (Two phase flow (liquid-sediment or non-Newtonain flow)):

- 2-D dam break over erodible sediment (Fourtakas et al, 2016).

Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0

6. HELP

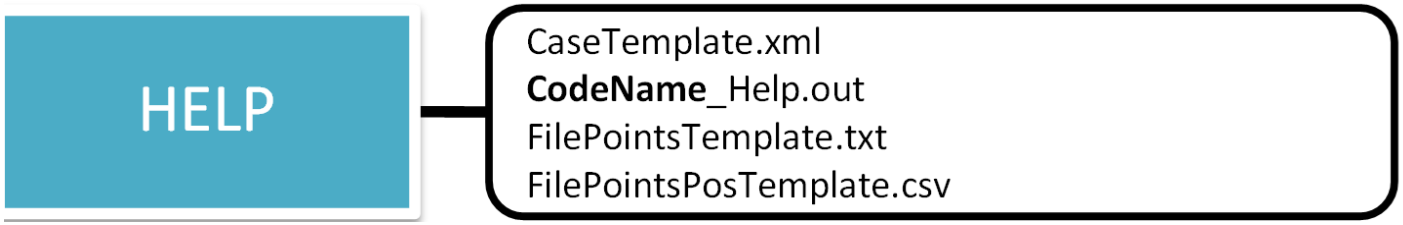
THERE ARE A LOT OF POSSIBILITIES TO HELP USERS:

- **HELP** folder in the package to download.
- **DOCUMENTATION**: User guide, XML guide, other PDFs.
- **FAQ**: Frequently asked questions
  http://dual.sphysics.org/index.php/forums/
- **Forums**: http://dual.sphysics.org/index.php/forums1/
- **YouTube** channel: https://www.youtube.com/user/DualSPHysics
- **Twitter** account: https://twitter.com/DualSPHysics
- **Mail** to dualsphysics@gmail.com

Please get in touch

6. HELP

- **HELP folde**r: in the package to download



- *CaseTemplate.xml*, a XML example with all the different labels and formats that can be used
- Description of the execution parameters of the code is presented in *HELP_NameCode.out*.
- Other TXT and CSV files with examples of positions of points to be used with MeasureTool code.

6. HELP

- **DOCUMENTATION:**

  - ***DualSPHysics_v4.0_GUIDE.pdf***:

  Manuscript with all information about SPH theory, source files, how to run the codes, description of the different paramters, description of the testcases

  - ***XML_GUIDE.pdf***:

  Helps to create a new case using the input XML file and all XML parameters are explained and related to SPH equations.

  - ***ExternalModelsConversion_GUIDE.pdf****:*

  Describes how to convert the file format of any external geometry of a 3-D model to VTK, PLY or STL using open-source codes.

  - ***PostprocessingCalculations.pdf****:*

  Explains how numerical magnitudes are computed.

6. HELP

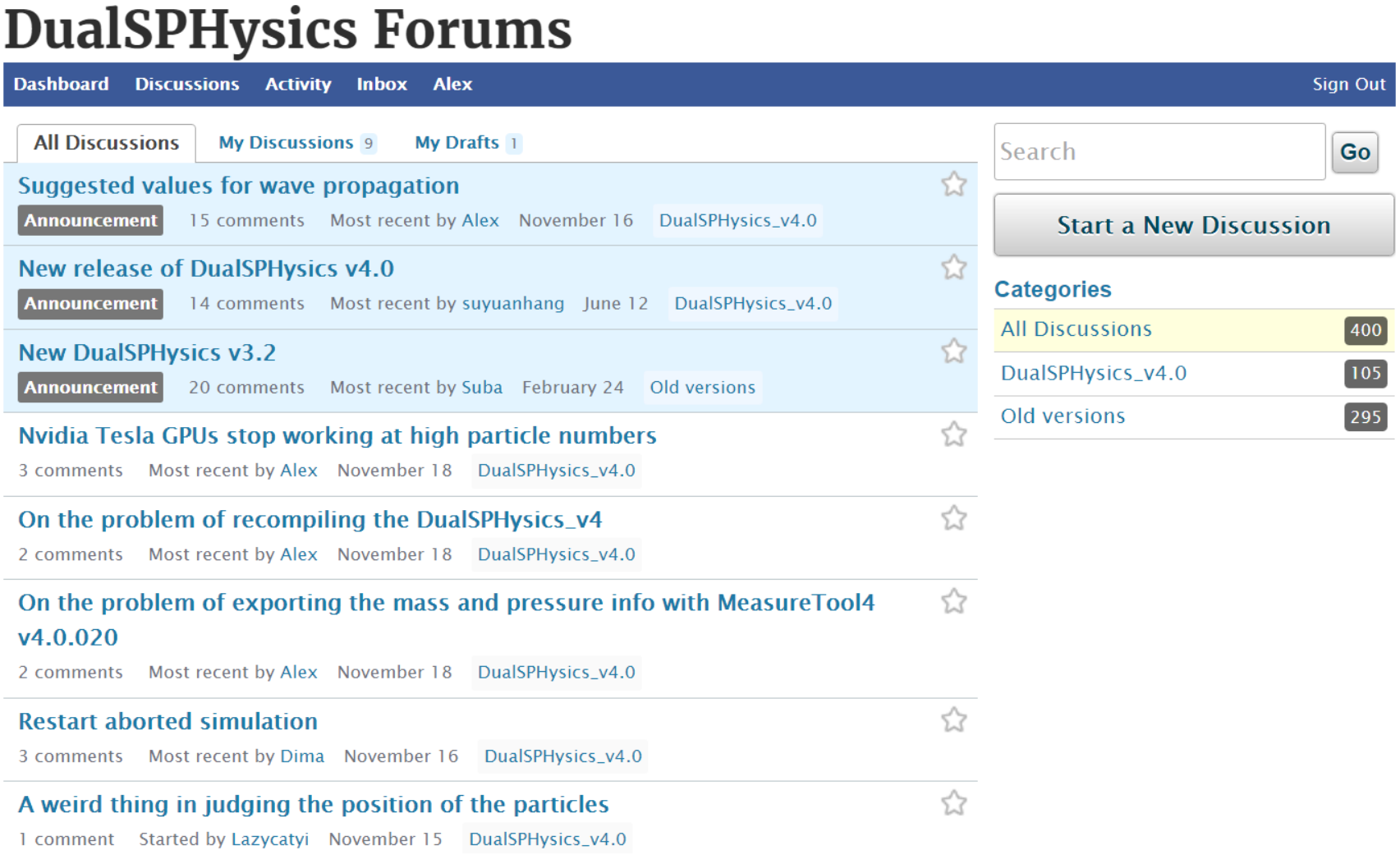
- **FAQ:** Frequently asked questions http://dual.sphysics.org/index.php/forums

FAQ: Frequently asked questions

1. What do I need to use DualSPHysics? What are the hardware and software requirements?
2. Why DualSPHysics binary is not running?
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 can I modify the code of the precompiled libraries?
9. How does the code define the limits of the domain?
10. How can I define the movement of boundaries?
11. How can I include a new type of movement?
12. How do I prevent the boundary particles from going outside of the domain limits when applying motion?
13. Why do I observe a gap between boundaries, floating bodies and fluid in the solution?
14. How do I prevent the fluid particles from penetrating inside floating bodies?
15. How do I numerically compute the motion and rotation of floating bodies?
16. When are the fluid particles excluded from the simulation?
17. How do I create a 2-D simulation?
18. How can I solve the error "Constant 'b' cannot be zero"?
19. How can I create a case without gravity?
20. How can I define the speed of sound?
21. What is the recommended alpha value in artificial viscosity?
22. How can I define new properties of the particles?
23. How can I store new properties of the particles (e. g. *Temperature*)?
24. How must I cite the use of the code in my paper?

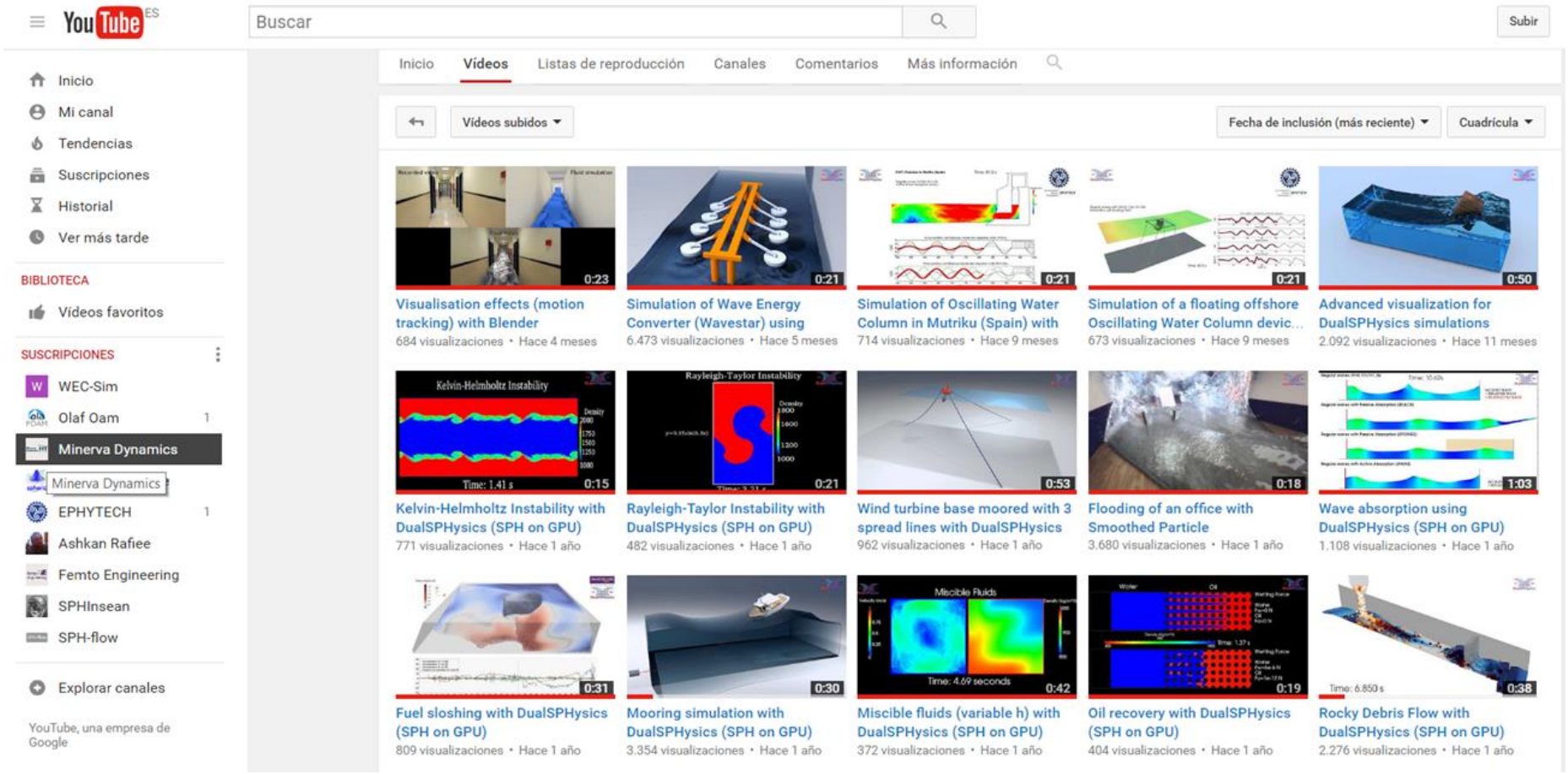
6. HELP

- **Forums:** http://dual.sphysics.org/index.php/forums1/



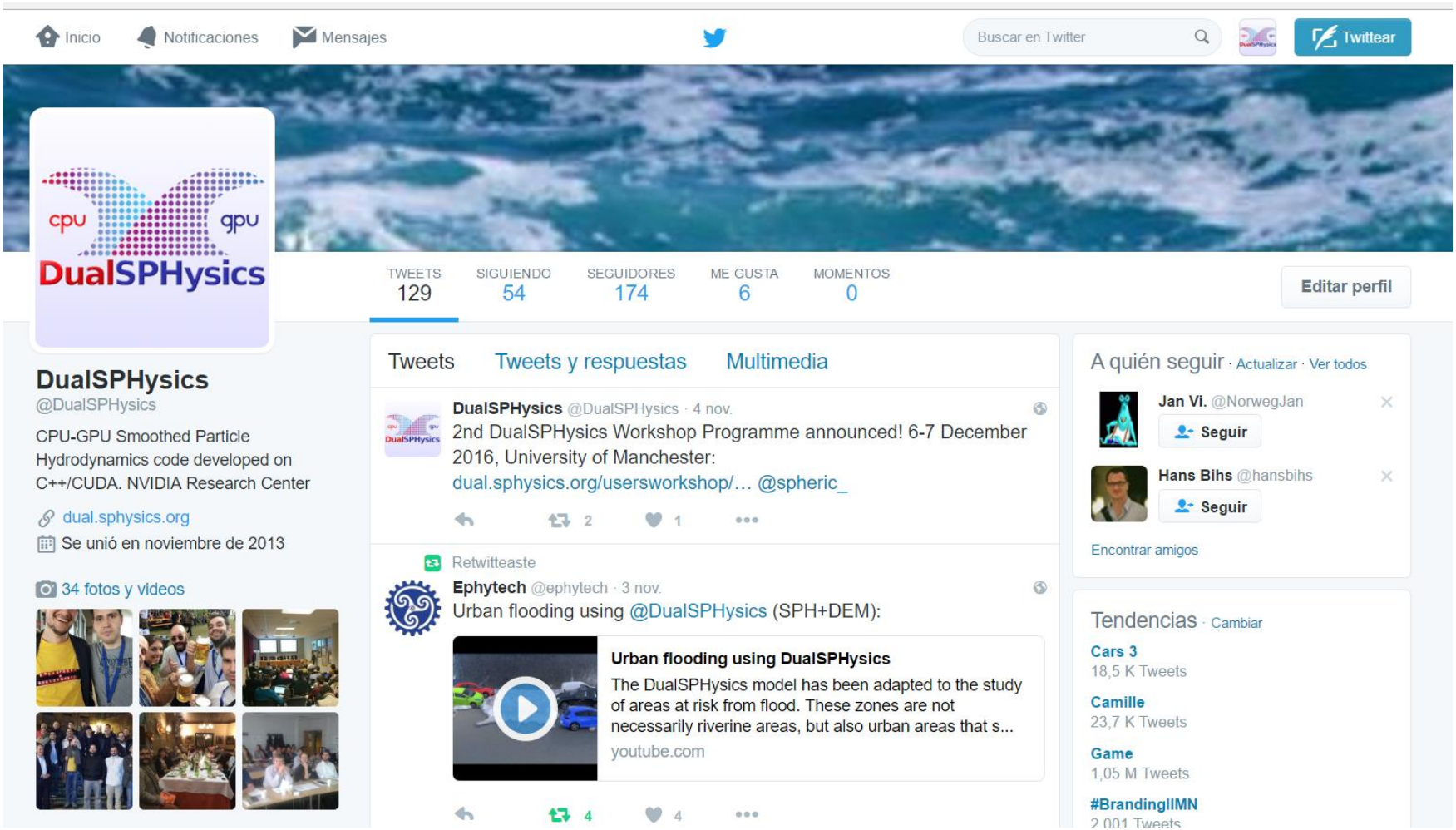
6. HELP

- **YouTube channel:** https://www.youtube.com/user/DualSPHysics



6. HELP

- **Twitter account:** https://twitter.com/DualSPHysics



Outline of Presentation

1. DualSPHysics
   - 1.1. Origin of DualSPHysics
   - 1.2. Why GPUs?
   - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
   - 3.1. Steps of simulation
   - 3.2. Source files
   - 3.3. Object-Oriented Programming
   - 3.4. Execution diagram
4. Input & output files
5. Test cases online
6. HELP
7. New Features in v4.0

6. New Features in version v4.0

- **Structure of the CPU and GPU code:**
  - Common structure for the CPU and GPU code (Only CUDA kernels are implemented in the CUDA files (*.cu*)).
  - Symmetry of pair-wise interactions is not applied in version 4.0.
  - Optimised for execution in multi-core CPU machines (an efficiency of 86.2% is achieved using 32 cores to simulate 150k particles).

- **Optimisation of the size of blocks for execution of CUDA kernels:**
  - New automatic estimation of the optimum block size of CUDA kernels.
  - This optimum block size depends on:
    (i) features of the kernel (registers and shared memory),
    (ii) compilation parameters and CUDA version,
    (iii) hardware to be used and GPU specifications
    (iv) input data to be processed by the kernel (divergence, memory coalescent access).

- **Double precision:**
  - Double precision for position and updating state of particles.
  - Reduction of noise for long domains with a large number of particles.

6. New Features in version v4.0

- **Improved formulation for adding external forces:**
  - Translational and rotational external forces (Longshaw and Rogers 2015).

- **Movement from external file for rotation.**
  - External file movement with translational and rotational motion.

- **Coupled SPH & DEM.**
  - Coupled numerical solution, based on SPH and DEM discretisation, resolves solid-solid and solid-fluid interactions in broad range of scales (Canelas et al., 2016).

- **Multi-Phase soil-water.**
  - The DualSPHysics code has been validated for multi-phase simulations involving water and sediment for fully saturated flows (Fourtakas and Rogers, 2016).

6. New Features in version v4.0

- **Novel Shifting algorithm:**
  - Shifting algorithm (Lind et al., 2012).
  - Better particle distribution and moments, less noise and void reduction.

- **Automatic wave generation:**
  - Wave generation for long-crested waves, regular and random waves can be generated.

- **More powerful GenCase.**
  - Larger cases can be rapidly created using OpenMP.

- **New options in MeasureTool.**
  - Now the MeasureTool code can compute magnitudes at locations/points that change position with time.

- **Force computation.**
  - This code computes the force exerted directly from the momentum equation during interaction between fluid and boundary particle.

6. New Features in version v4.0

- **More information for floating bodies and motion analysis.**
  - The new code "*FloatingInfo*" allows to obtain different variables of interest of the floating objects during the simulation.
  - Available variables: Positions of the centre, linear velocity, angular velocity, motions (surge, sway and heave) and angles of rotation (pitch, roll and yaw).

- **Store information with variable output time.**
  - The new version allows user to define intervals of time with different output time to save results.

- **New properties for different type of fluids or boundaries.**
  - The user can define new properties for the particles according to "mk" in the XML file and then these can be loaded in DualSPHysics.

6. New Features in version v4.0

- **New variables in post-processing tools defined by the user.**
  - A user can create a new variable in DualSPHysics source files and one can define its own output data that can be then used by PartVTK, MeasureTool, IsoSurface, etc.

- **Alternative building method via CMAKE** (https://cmake.org/).
  - CMAKE is a cross-platform and an independent building system for compilation. This software generates native building files (like makefiles or Visual Studio projects) for any platform. *Note that this method is on trial for version 4!*

MANCHESTER 1824

The University of Manchester

Thank you

Acknowledgements

- DualSPHysics team: all developers and contributors

Website

Free open-source **DualSPHysics** code:

http://www.dual.sphysics.org