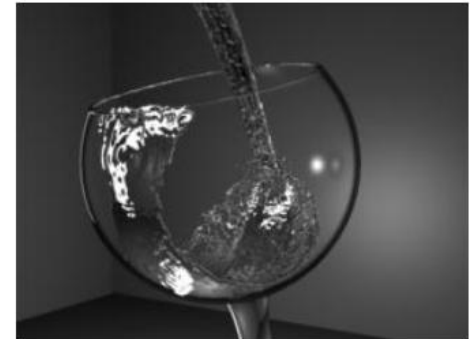
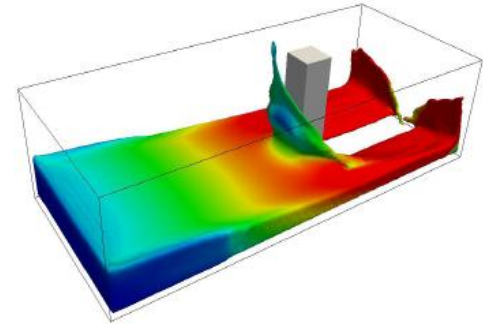


# Structure of the DualSPHysics code



The University of Manchester

**Dr José Manuel Domínguez Alonso**  
[jmdominguez@uvigo.es](mailto:jmdominguez@uvigo.es)



UniversidadeVigo

**Dr George Fourtakas**  
[Georgios.Fourtakas@postgrad.manchester.ac.uk](mailto:Georgios.Fourtakas@postgrad.manchester.ac.uk)

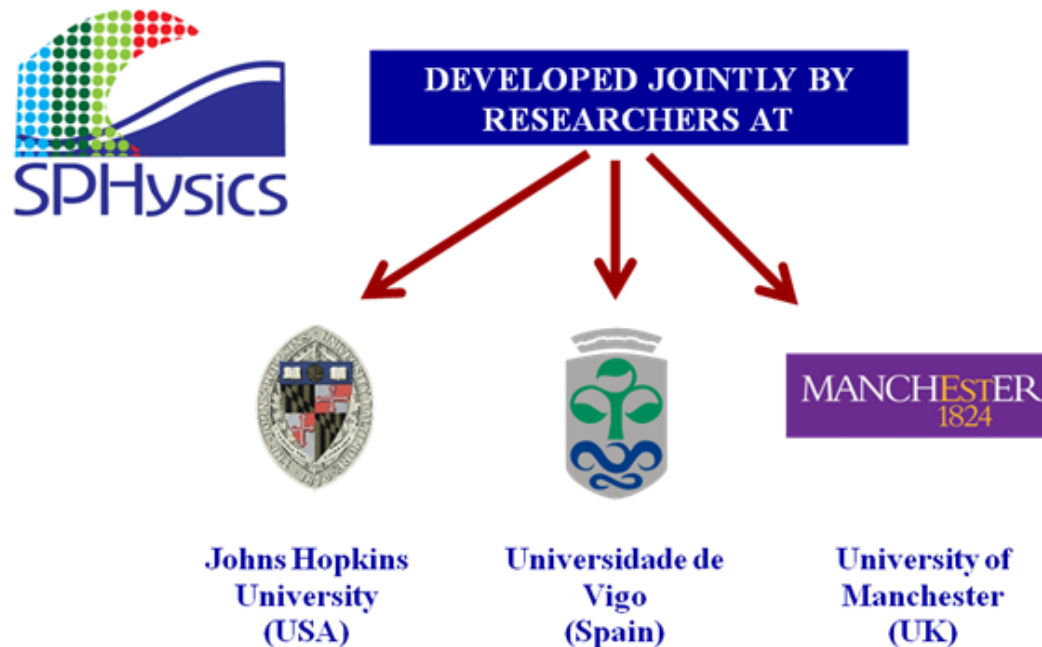
# Outline

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



# 1.1. Origin of DualSPHysics

**The DualSPHysics code was created starting from SPHysics.**



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

It is a code written in Fortran90 with numerous options (different kernels, several boundary conditions, etc.), which has 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 REQUIRES HIGH COMPUTATIONAL COST THAT INCREASES WHEN INCREASING THE NUMBER OF PARTICLES
- THE SIMULATION OF REAL PROBLEMS REQUIRES HIGH RESOLUTION WHICH IMPLIES SIMULATING MILLIONS OF PARTICLES

**IT WAS NECESSARY TO INCREASE THE PERFORMANCE OF THE CODE A FACTOR  
x100**

Classic options:

- **OpenMP:** Distribute the workload among all CPU cores ( $\approx 4x$ )
- **MPI:** Combines the power of multiple machines connected via network (high cost).

New option:

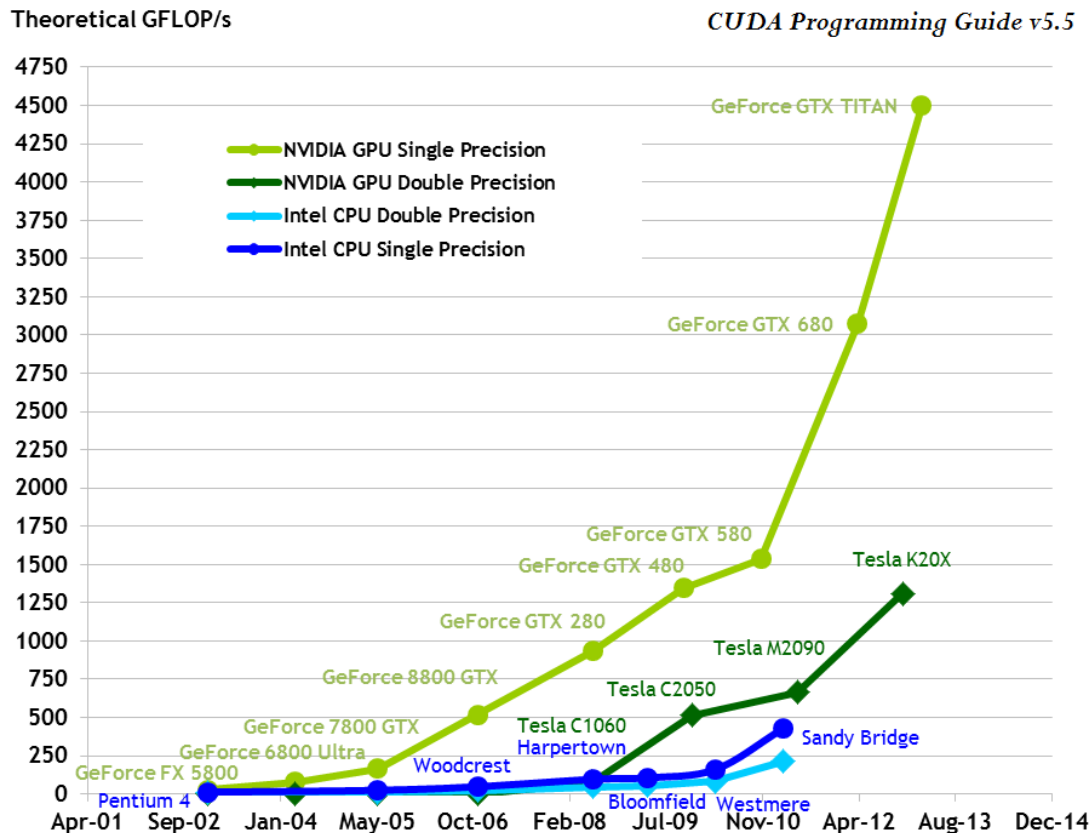
- **GPU:** Graphics cards with a high parallel computing power (cheap and accessible).

## 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 x100 or more.



**Advantages:** GPUs provide the necessary computational 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,  
many numerical methods in CFD and other computational  
methods



5X

Digital Content Creation  
Adobe



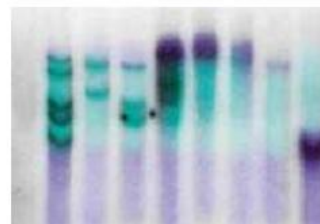
18X

Video Transcoding  
Elemental Technologies



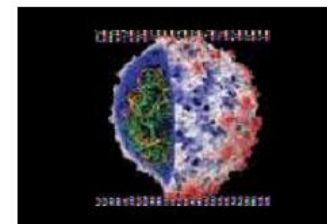
20X

3D Ultrasound  
TechniScan



30X

Gene Sequencing  
U of Maryland



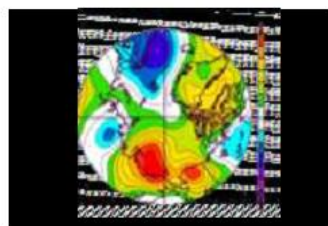
36X

Molecular Dynamics  
U of Illinois, Urbana-Champaign



50X

MATLAB Computing  
AccelerEyes



80X

Weather Modeling  
Tokyo Institute of Technology



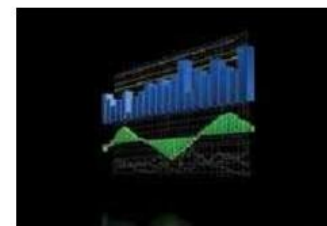
100X

Astrophysics  
RIKEN



146X

Medical Imaging  
U of Utah



149X

Financial Simulation  
Oxford University

<http://www.nvidia.com>

## 1.2. Why GPUs?

<http://www.top500.org/list/2015/06/>

### TOP500 LIST – JUNE 2015

$R_{\max}$  and  $R_{\text{peak}}$  values are in TFlops. For more details about other fields, check the [TOP500 description](#).

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	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	3120000	33862.7	54902.4	17808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 Villifx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8586.6	10066.3	3945



## 1.3. DualSPHysics project



First version in late 2009.

It includes **two implementations**:

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

Both implementations optimized for best performance for each architecture.

### Why two implementations?

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

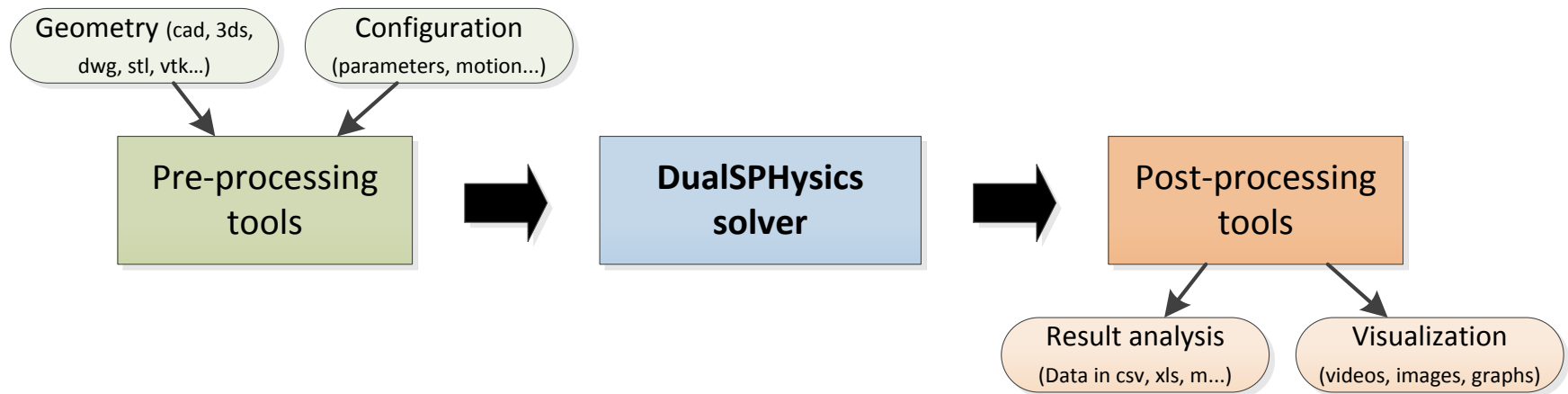
It is easier to understand the code in CUDA when you can see the same code in C++.

**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 from SPH particles.

# 1.3. DualSPHysics project



DualSPHysics

Downloads DualSPHysics Project GPU Computing SPHysics Project Validation  
Applications Animations References Forums FAQ News Contact



Universidade de Vigo

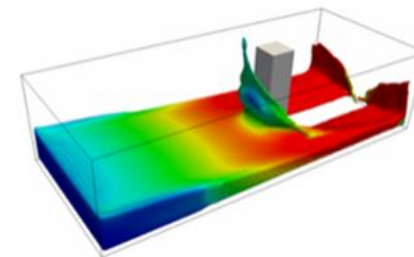
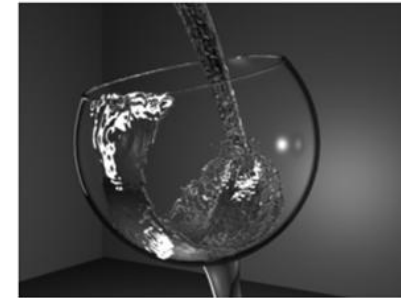


DualSPHysics is based on the Smoothed Particle Hydrodynamics model named SPHysics ([www.sphysics.org](http://www.sphysics.org)).

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++ and CUDA codes to deal with real-life engineering problems.

Contact E-Mail: [dualsphysics@gmail.com](mailto:dualsphysics@gmail.com)



[www.dual.sphysics.org](http://www.dual.sphysics.org)

# 1.3. DualSPHysics project

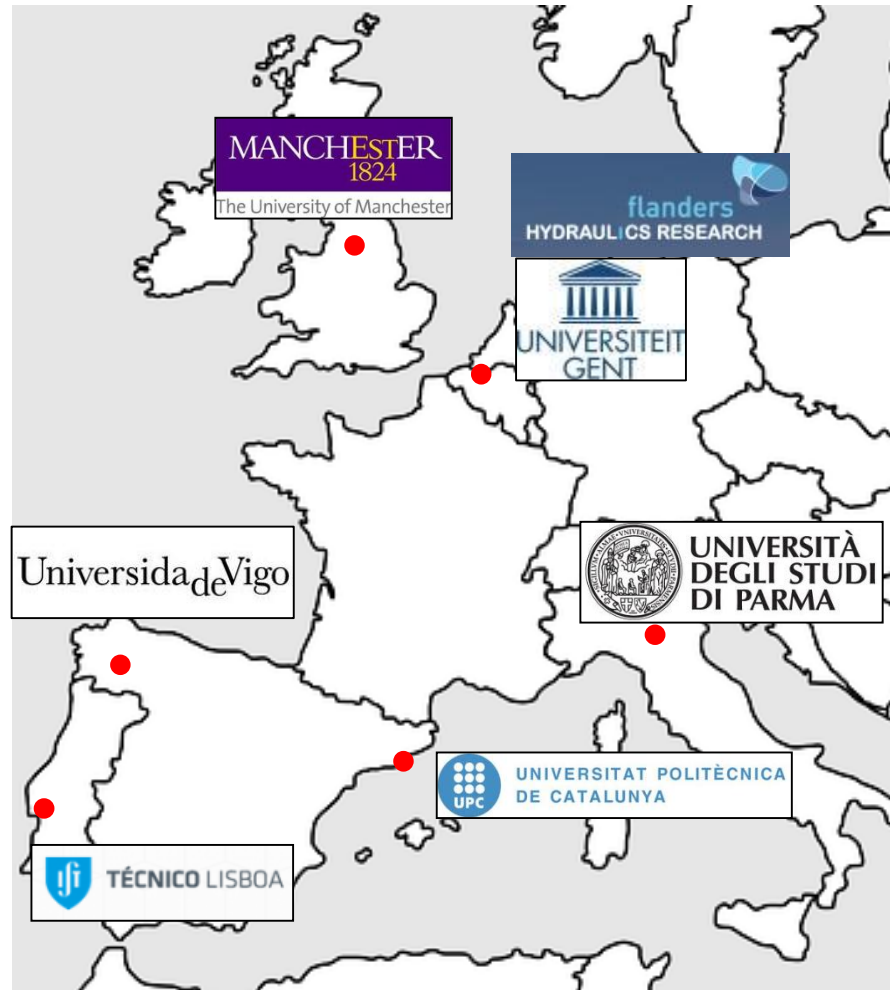


## People working on DualSPHysics project:

**Dr Benedict D. Rogers**  
**Dr Athanasios Mokos**  
**Dr Georgios Fourtakas**  
**Dr Stephen Longshaw**  
**Abouzied Nasar**  
**Prof. Peter Stansby**

**Prof. Moncho Gómez Gesteira**  
**Dr Alejandro J.C. Crespo**  
**Dr Jose M. Domínguez**  
**Dr Anxo Barreiro**  
**Orlando G. Feal**  
**Carlos Alvarado**

**Prof. Rui Ferreira**  
**Dr Ricardo Canelas**



**Dr Corrado Altomare**  
**Dr Tomohiro Suzuki**

**Dr Renato Vacondio**  
**Prof. Paolo Mignosa**

**Dr Xavier Gironella**  
**Andrea Marzeddu**

# Outline

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

## 2. SPH formulation: DualSPHysics v3

- Time integration scheme:
  - Verlet [Verlet, 1967]
  - Symplectic [Leimkhuler, 1996]
- Variable time step [Monaghan and Kos, 1999]
- Smoothing kernel functions:
  - Cubic Spline kernel [Monaghan and Lattanzio, 1985]
  - Wendland kernel [Wendland, 1995]
- Weakly compressible approach using Tait's equation of state
- 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]
- Dynamic boundary conditions [Crespo et al., 2007]
- Floating objects [Monaghan et al., 2003]
- Periodic open boundaries

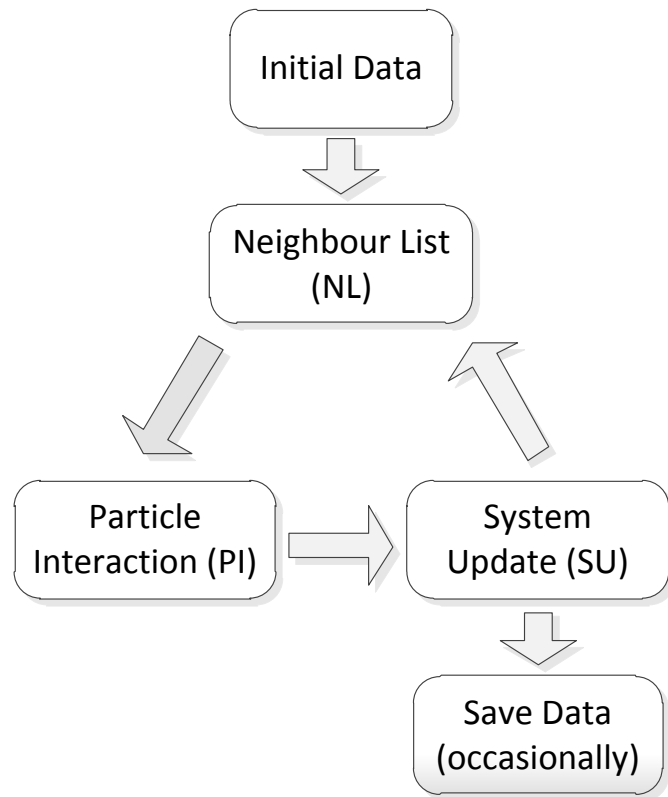
# Outline

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



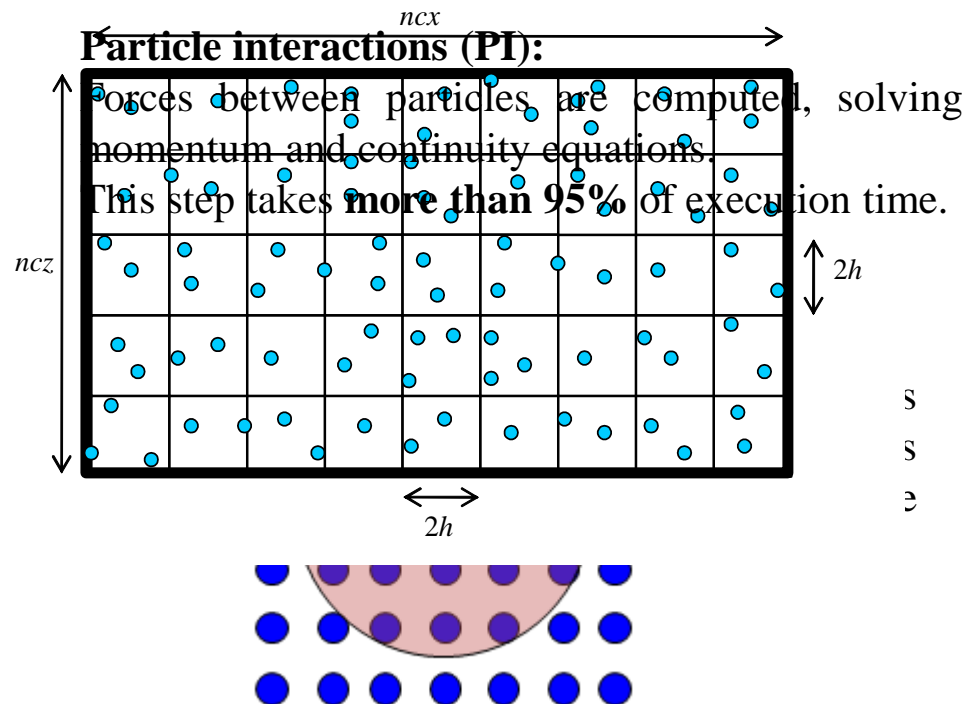
## 3.1. Stages of simulation

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 performance (memory access).



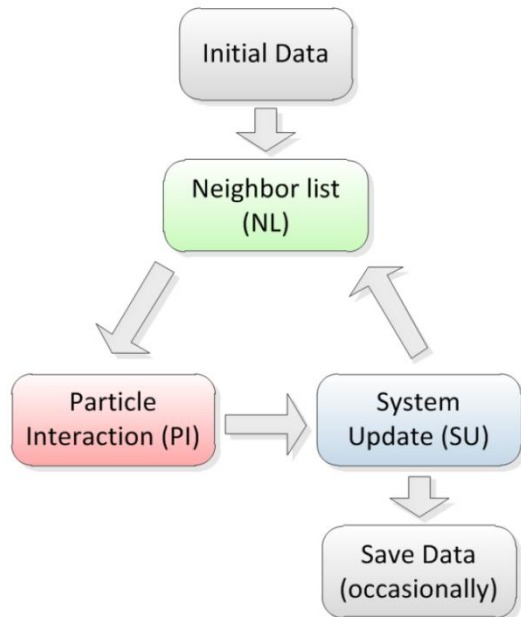


## 3.1. Stages of simulation

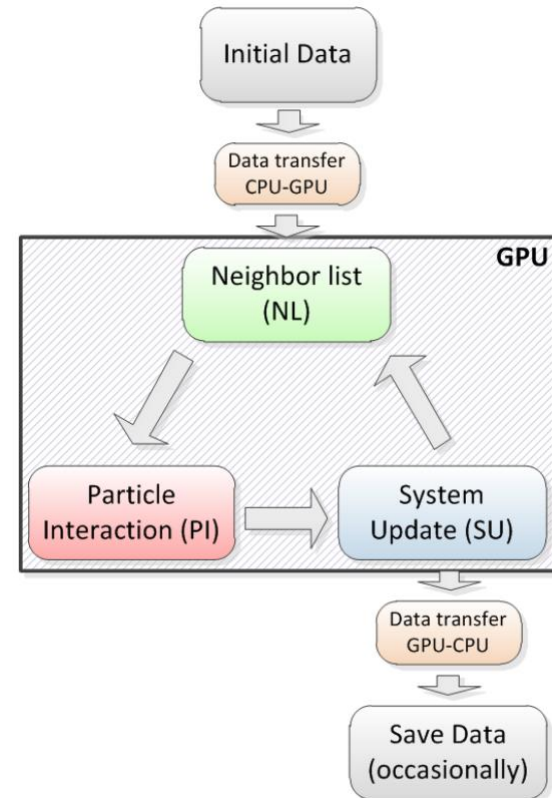
### Full GPU implementation

- 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

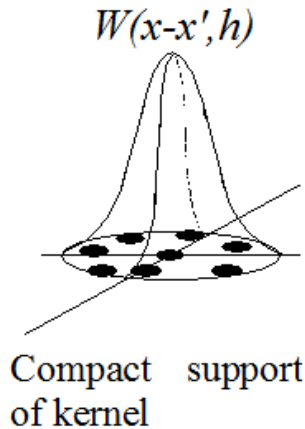


GPU implementation

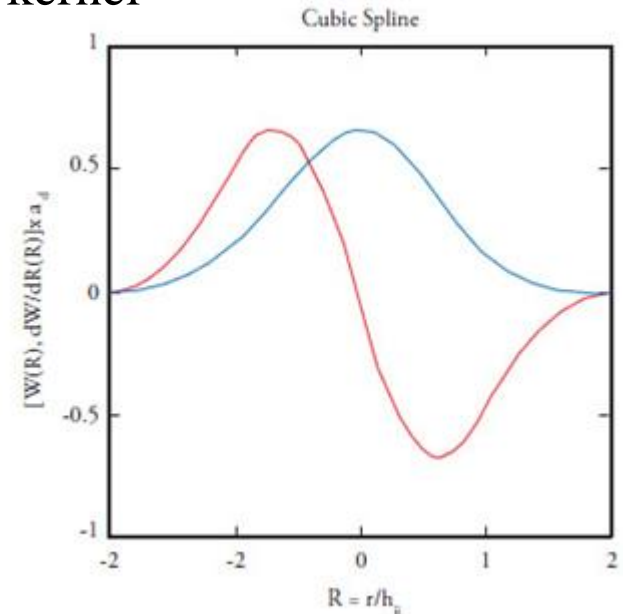


## 3.2. Source files

**SPH smoothing kernel:** This is the SPH smoothing kernel



$$W(R) = a_d \begin{cases} 1 - \frac{3}{2}R^2 + \frac{3}{4}R^3 & 0 \leq R \leq 1 \\ \frac{1}{4}(2-R)^3 & 1 \leq R \leq 2 \\ 0 & R \geq 2 \end{cases}$$



**NVIDIA CUDA kernel** (C++ directives): This is not SPH related but refers to the CUDA function

```
__global__ void KerPreInteractionSimple(unsigned n, const double2 *posxy, const double *posz,
    const float4 *velrhop, float4 *pospress, float cteb, float gamma)
{
    unsigned p = blockIdx.y * gridDim.x * blockDim.x + blockIdx.x * blockDim.x + threadIdx.x; // N° de la partícula
    if (p < n) {
        const float rrhop = velrhop[p].w;
        float press = cteb * (powf(rrhop * OVERRHOPCERO, gamma) - 1.0f);
        double2 rpos = posxy[p];
        pospress[p] = make_float4(float(rpos.x), float(rpos.y), float(posz[p]), press);
    }
}
```


## 3.2. Source files

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) JPtexasInfo (.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 <i>JFormatFiles2.lib / libjformatfiles2.a</i>  JSphMotion.h <i>JSphMotion.lib / libjsphmotion.a</i>  JXml.h <i>JXml.lib / libjxml.a</i>	main.cpp JCfgRun (.h .cpp) JSph (.h .cpp) JPartsLoad (.h .cpp) JPartsOut (.h .cpp) JSphDtFixed (.h .cpp) JSphVarAcc (.h .cpp) Types.h	 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)

### Files:

40 C++ headers  
 33 C++ codes  
 4 CUDA headers  
 4 CUDA codes

## 3.2. Source files: Common files

Common		
<p>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</p>	 <p>JSphDtFixed (.h .cpp)  JSphVarAcc (.h .cpp)  Types.h</p>	<p>Common files used in several pre-processing and post-processing applications.  They provide basic functionalities.  Users do not need to modify them.</p>
<p>JFormatFiles2.h  <i>JFormatFiles2.lib / libjformatfiles2.a</i></p> <p>JSphMotion.h  <i>JSphMotion.lib / libjsphmotion.a</i></p> <p>JXml.h  <i>JXml.lib / libjxml.a</i></p>	<p>JSphCpu (.h .cpp)  JSphCpuSingle (.h .cpp)  JSphTimersCpu.h  JCellDivCpu (.h .cpp)  JCellDivCpuSingle (.h .cpp)  JPeriodicCpu (.h .cpp)</p>	<p>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)</p>

## 3.2. Source files: Precompiled files

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	main.cpp JCfgRun (.h .cpp) JSph (.h .cpp) JPartsLoad (.h .cpp) JPartsOut (.h .cpp) JSphDtFixed (.h .cpp) JSphVarAcc (.h .cpp) Types.h	
JFormatFiles2.h <i>JFormatFiles2.lib / libjformatfiles2.a</i>  JSphMotion.h <i>JSphMotion.lib / libjsphmotion.a</i>  JXml.h <i>JXml.lib / libjxml.a</i>	JSphCpu (.h .cpp)  JSphCpuSingle (.h .cpp) JSphTimersCpu.h JCellDirCpu (.h .cpp) JCell JPeri	JSphGpu (.h .cpp) JSphGpu_ker (.h .cu)  JSphGpuSingle (.h .cpp)  JSphTimersGpu.h  JCellDirGpu (.h .cpp)  JGpuArrays (.h .cpp)

Precompiled code in libraries.

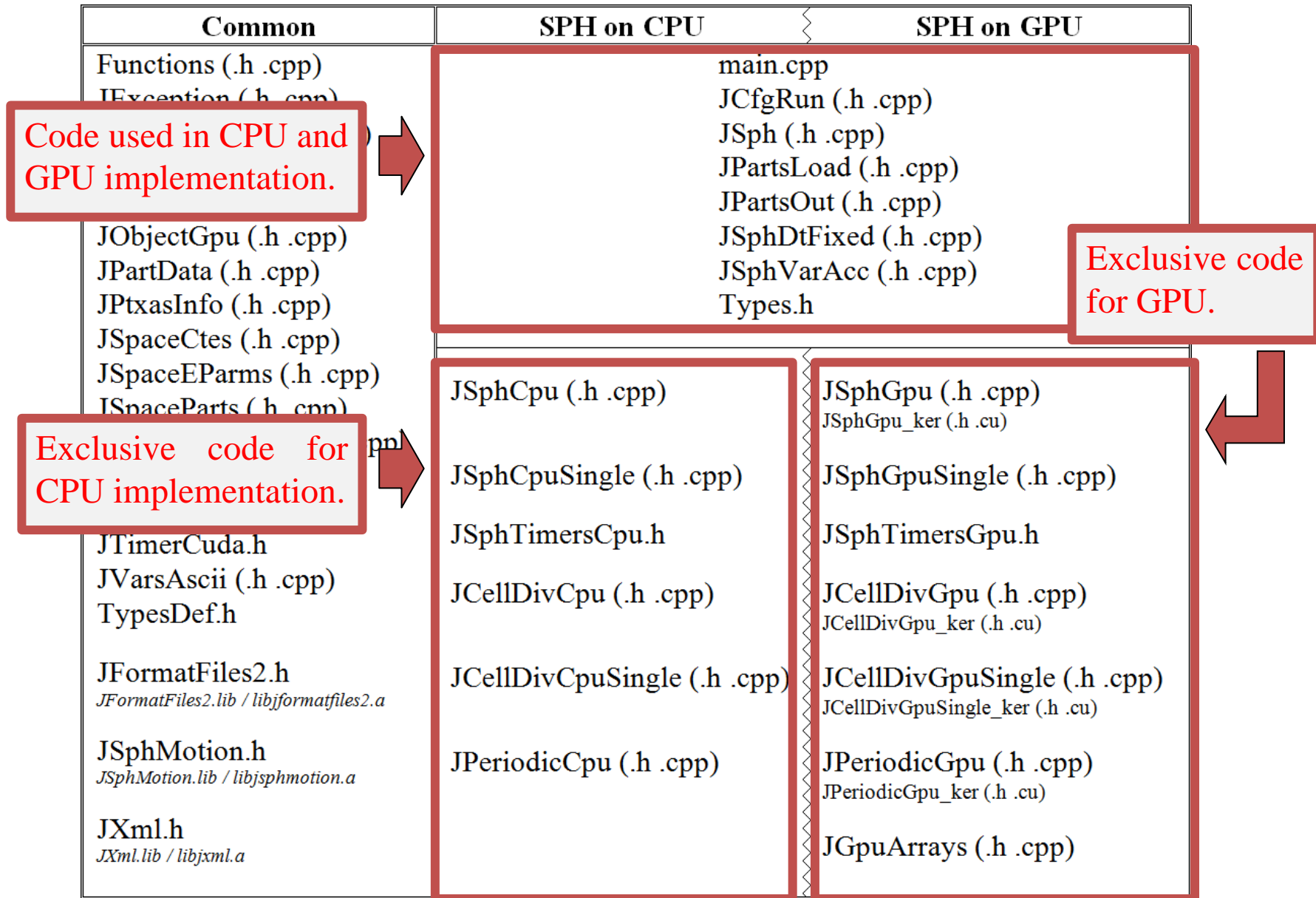
Common files used in several pre-processing and post-processing applications.

They provide basic functionalities.

## 3.2. Source files: SPH files

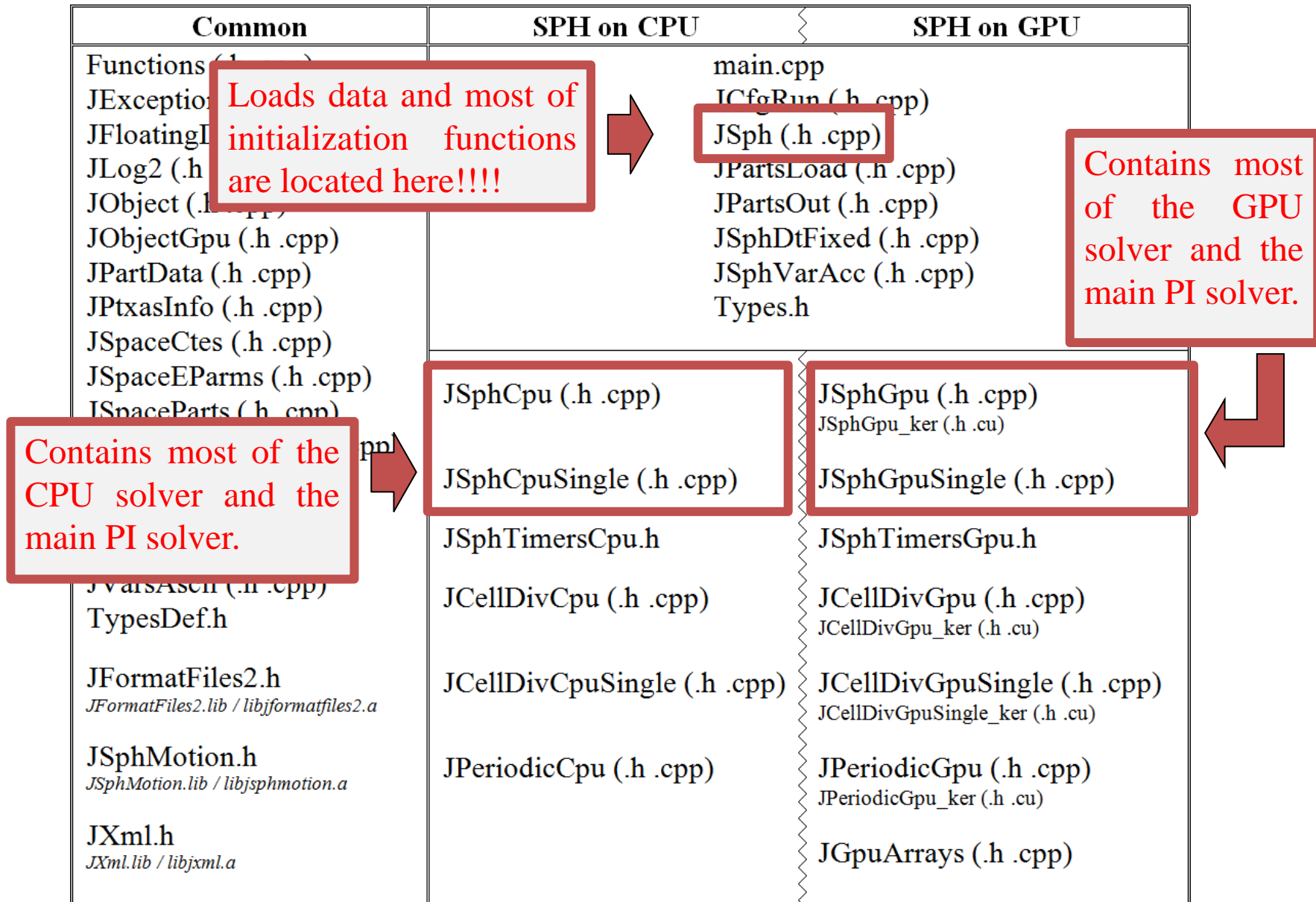
Common	SPH on CPU	SPH on GPU
<p>Exclusive source code files of DualSPHysics to implement the SPH model.</p> <p>JPartData (.h .cpp)  JPartxasInfo (.h .cpp)  JSpaceCtes (.h .cpp)  JSpaceEParms (.h .cpp)  JSpaceParts (.h .cpp)  JSpaceProperties (.h .cpp)  JRangeFilter (.h .cpp)  JTimer.h  JTimerCuda.h  JVarsAscii (.h .cpp)  TypesDef.h</p> <p>JFormatFiles2.h  <i>JFormatFiles2.lib / libjformatfiles2.a</i></p> <p>JSpHmotion.h  <i>JSpHmotion.lib / libjsphmotion.a</i></p> <p>JXml.h  <i>JXml.lib / libjxml.a</i></p>	<p>main.cpp  JCfgRun (.h .cpp)  JSph (.h .cpp)  JPartsLoad (.h .cpp)  JPartsOut (.h .cpp)  JSphDtFixed (.h .cpp)  JSphVarAcc (.h .cpp)  Types.h</p> <p>JSpHcpu (.h .cpp)  JSpHcpuSingle (.h .cpp)  JSpHtimersCpu.h  JCellDivCpu (.h .cpp)  JCellDivCpuSingle (.h .cpp)  JPeriodicCpu (.h .cpp)</p>	<p>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)</p>

## 3.2. Source files: SPH files





## 3.2. Source files: A starting point to modify DualSPHysics





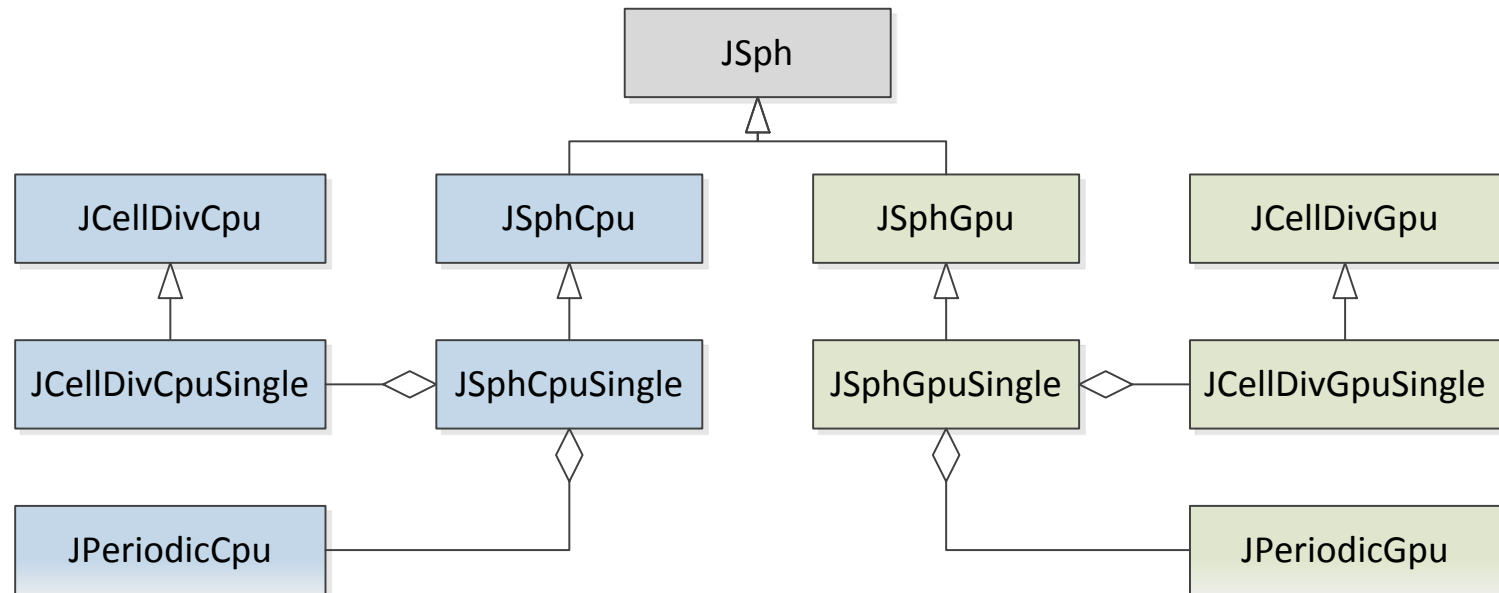
### 3.2. Source files

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp)		main.cpp
JException (.h .cpp)		JCfgRun (.h .cpp)
JFloatingData (.h .cpp)		JSph (.h .cpp)
JLog2 (.h .cpp)		JPartsLoad (.h .cpp)
JObject (.h .cpp)		JPartsOut (.h .cpp)
JObjectGpu (.h .cpp)		JSphDtFixed (.h .cpp)
JPartData (.h .cpp)		.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		
<i>JFormatFiles2.lib / libjformatfiles2.a</i>		
JSphMotion.h		
<i>JSphMotion.lib / libjsphmotion.a</i>		
JXml.h		
<i>JXml.lib / libjxml.a</i>		
	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)

### 3.3. Object-Oriented Programming

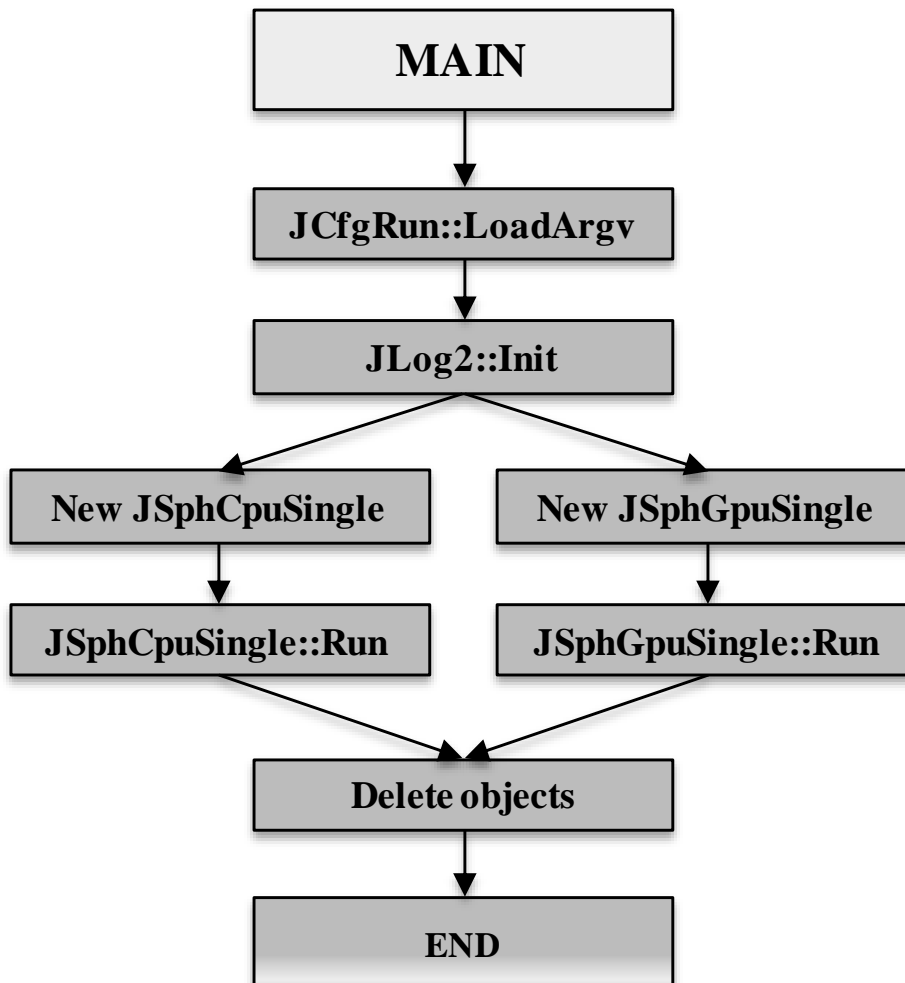
DualSPHysics uses Object-Oriented Programming (OOP) to organise the code. However, the particles data is stored in arrays to improve the performance. Class name matches the file name.

**Diagram main classes**



### 3.4. Execution diagram: Main

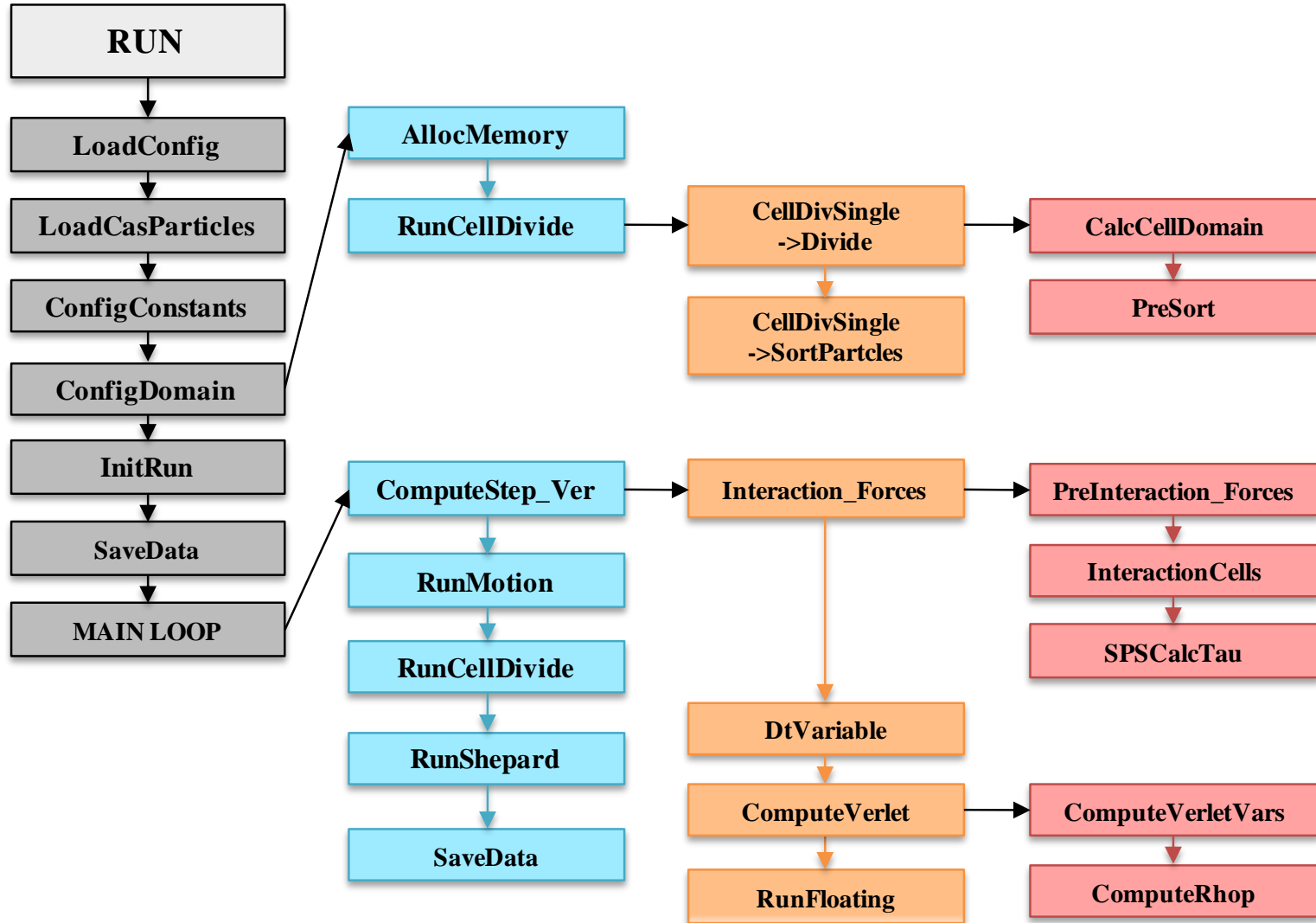
The execution of DualSPHysics begins in function `main()` in file `main.cpp`



- Loads parameters from command line
- Initializes log file
- Creates object for execution on Cpu or Gpu
- **Executes method `Run()` of SPH object to start simulation**
- Deletes objects and frees memory

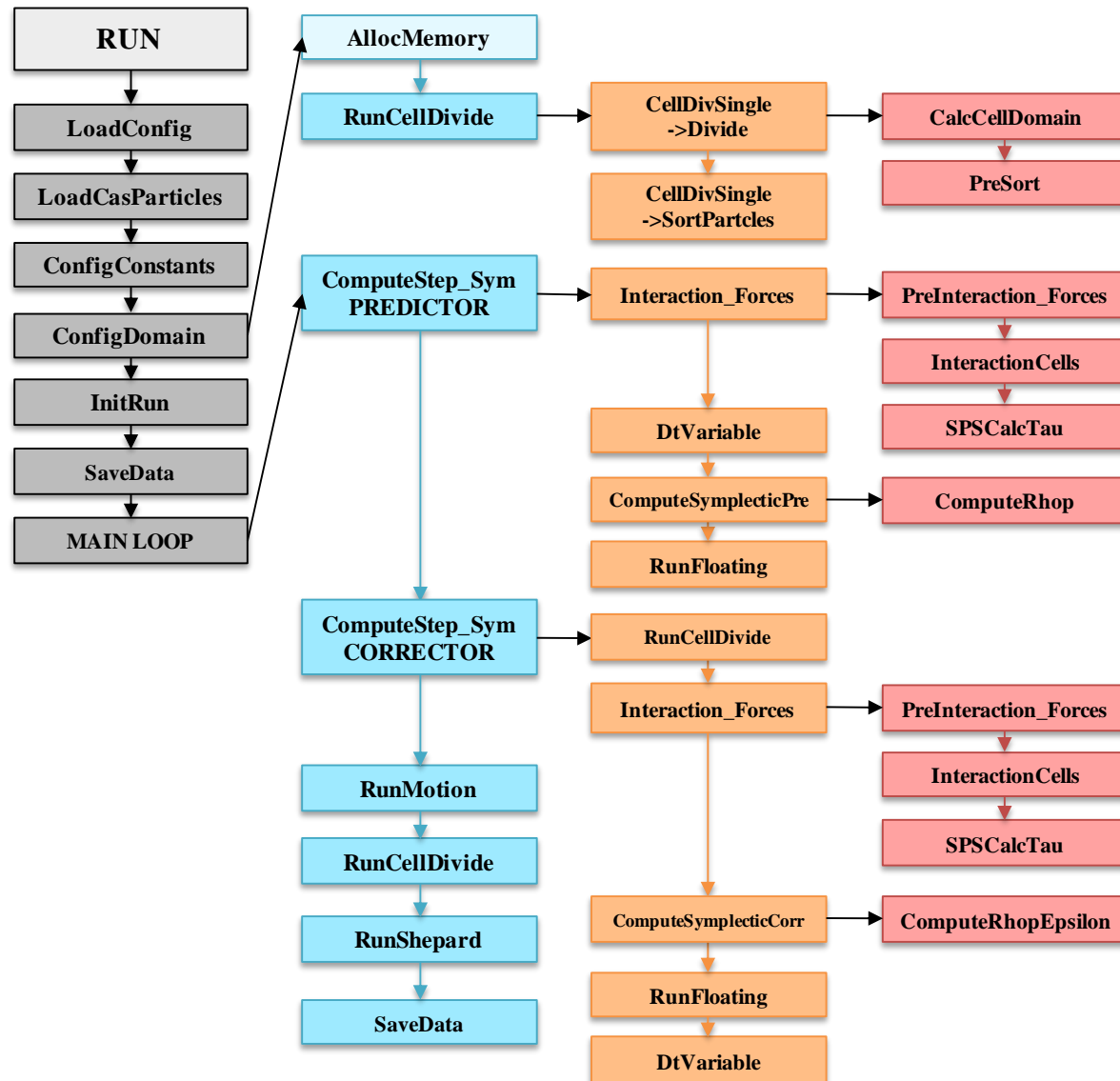
### 3.4. Execution diagram: CPU

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



### 3.4. Execution diagram: CPU

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

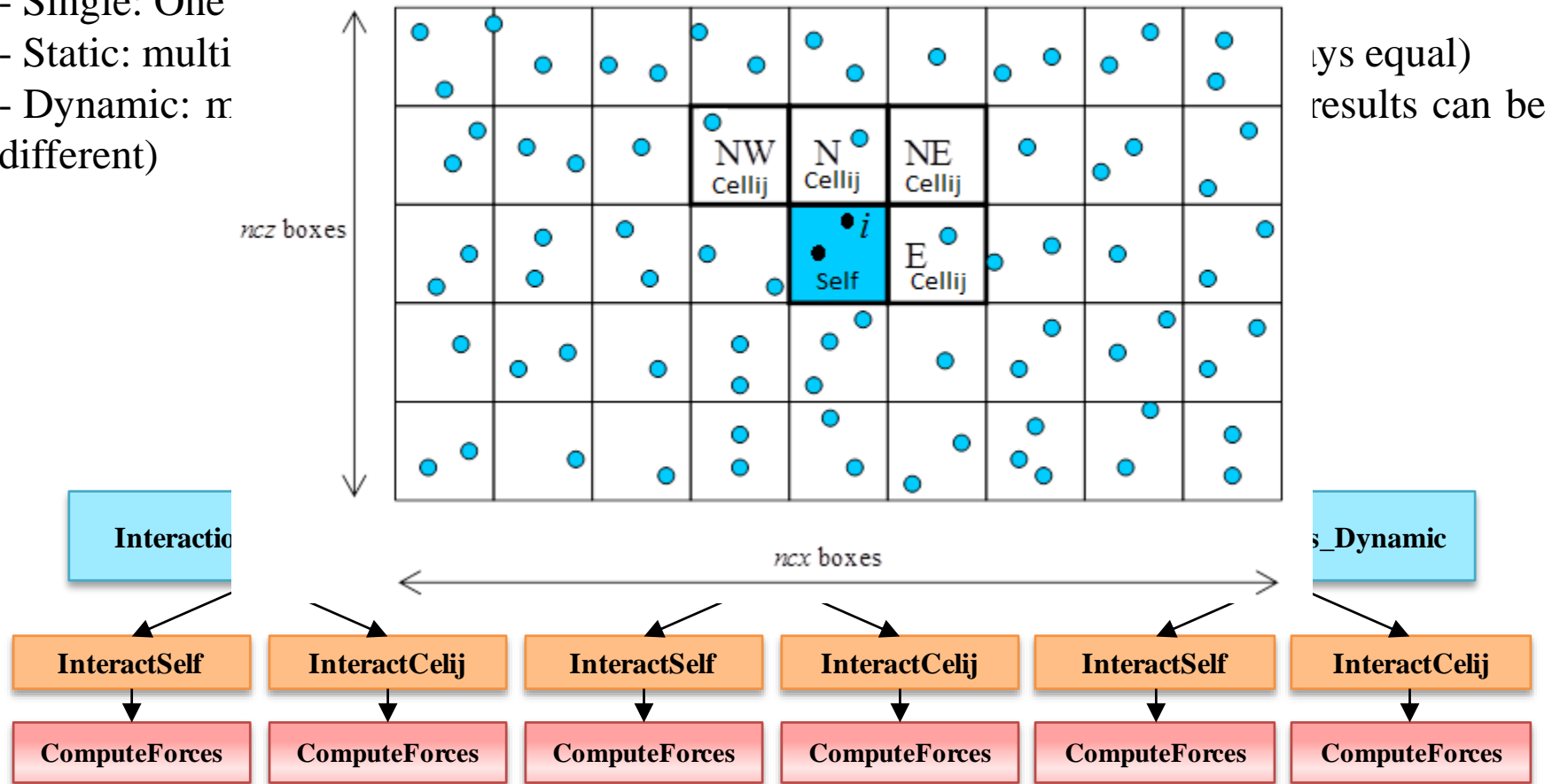


### 3.4. Execution diagram: CPU interaction

InteractionCells performs the interaction between particles to calculate forces or Shepard correction.

Three execution modes:

- Single: One
  - Static: multi
  - Dynamic: n
- (different)

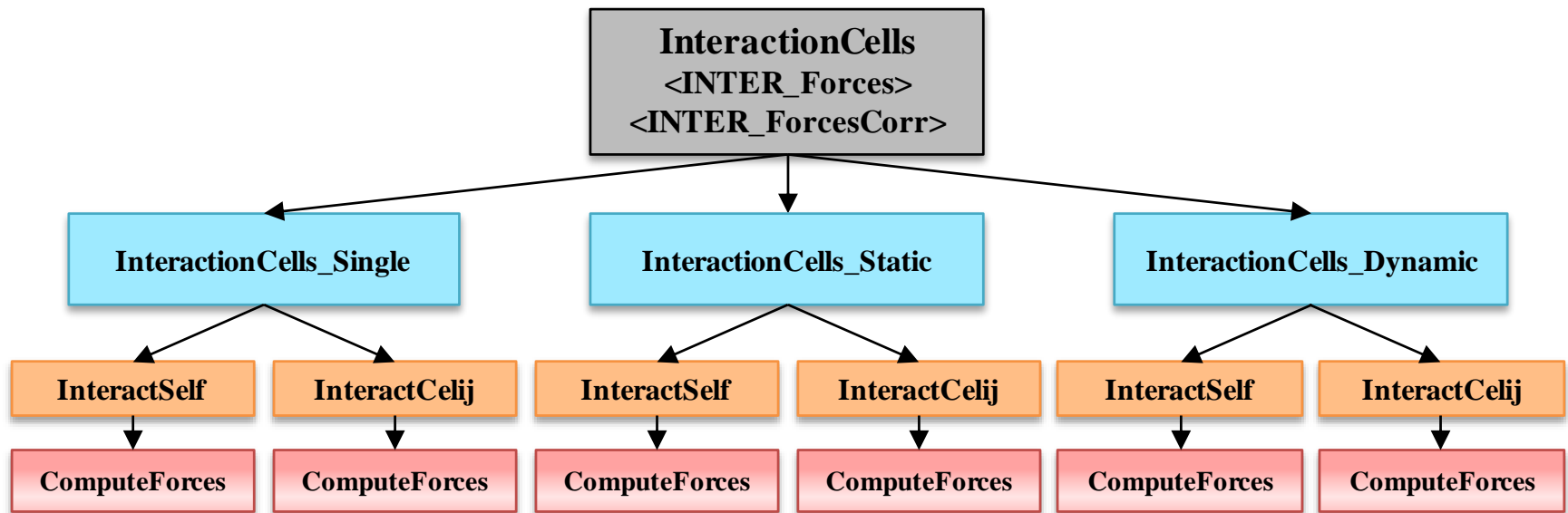


### 3.4. Execution diagram: CPU interaction

The interaction between particles is performed by cells:

- Interaction between particles in the same cell (InteractSelf)
- Interaction between particles from one cell with particles from another cell (InteractCelij)

The method **ComputeForces** performs the interaction between two particles **to calculate forces**.



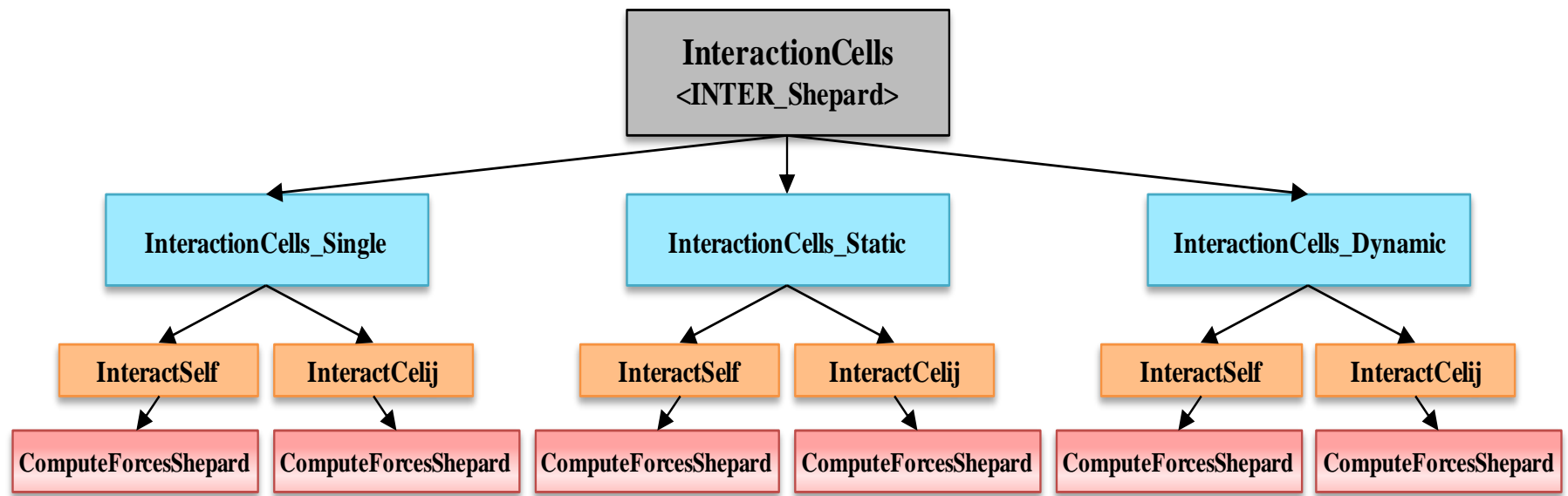
### 3.4. Execution diagram: CPU interaction

The interaction between particles is performed by cells:

- Interaction between particles in the same cell (InteractSelf)
- Interaction between particles from one cell with particles from another cell (InteractCelij)

The method `ComputeForces` performs the interaction between two particles to calculate forces.

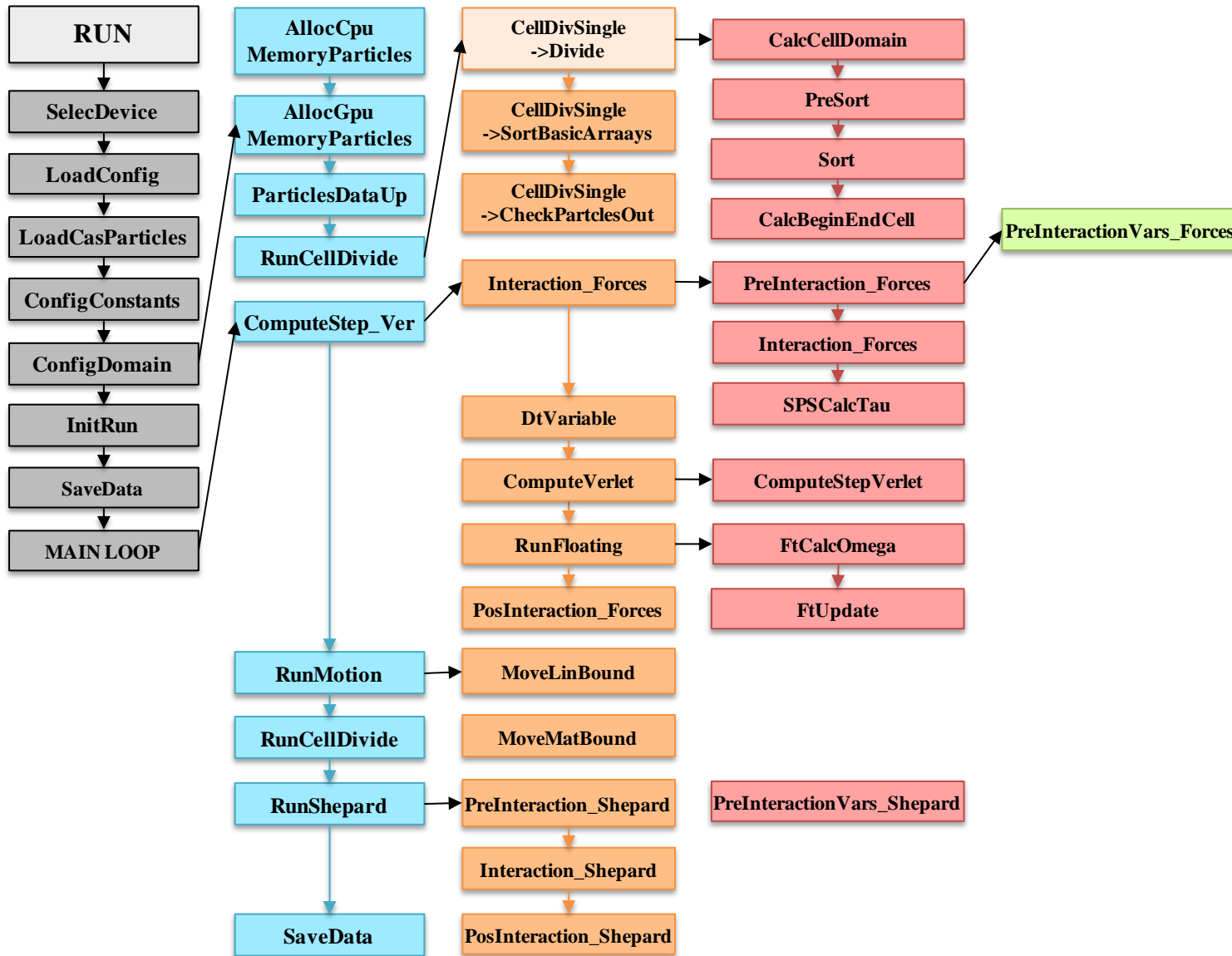
The method **`ComputeForcesShepard`** performs the interaction between two particles to calculate Shepard correction.





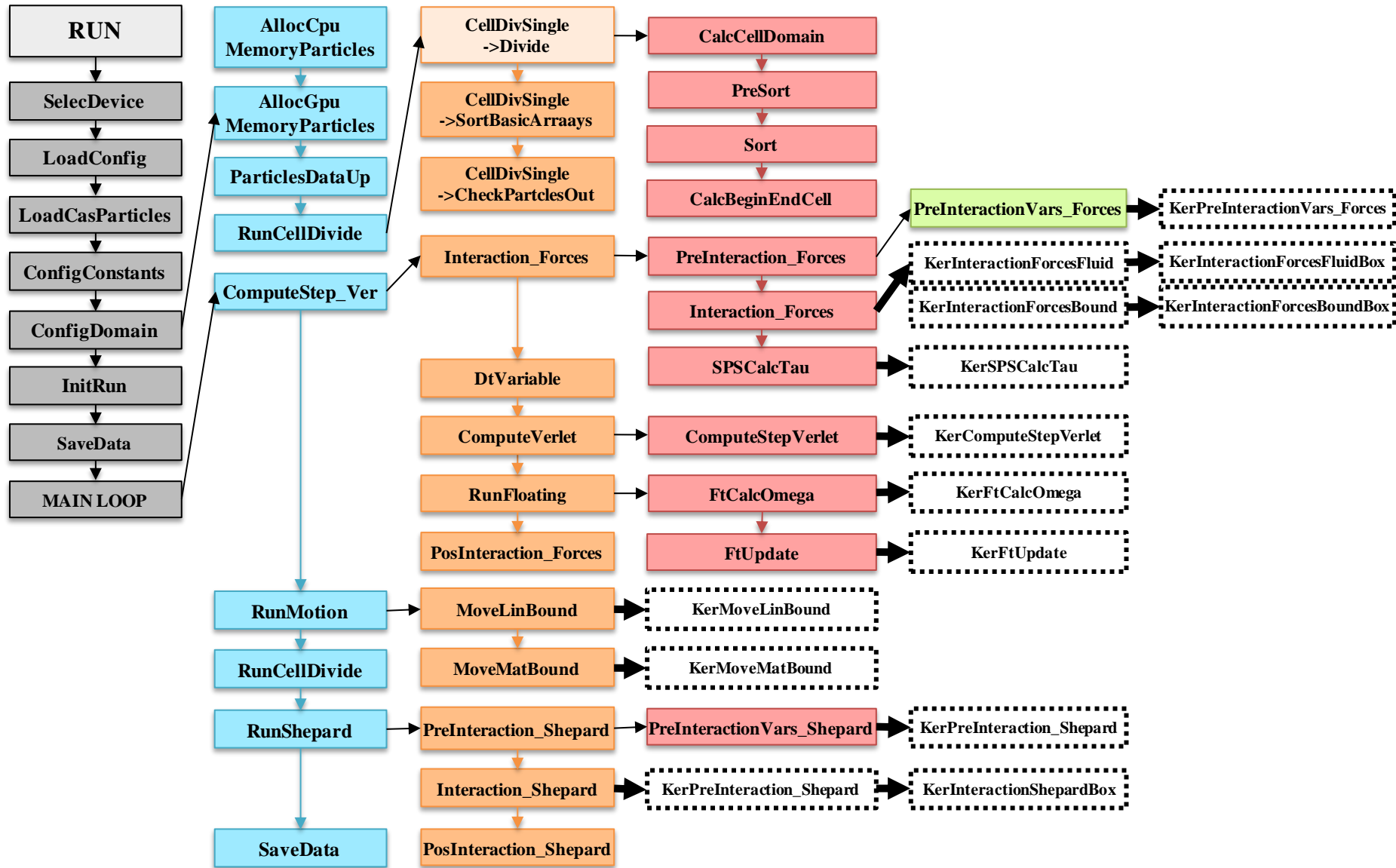
## 3.4. Execution diagram: GPU

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.

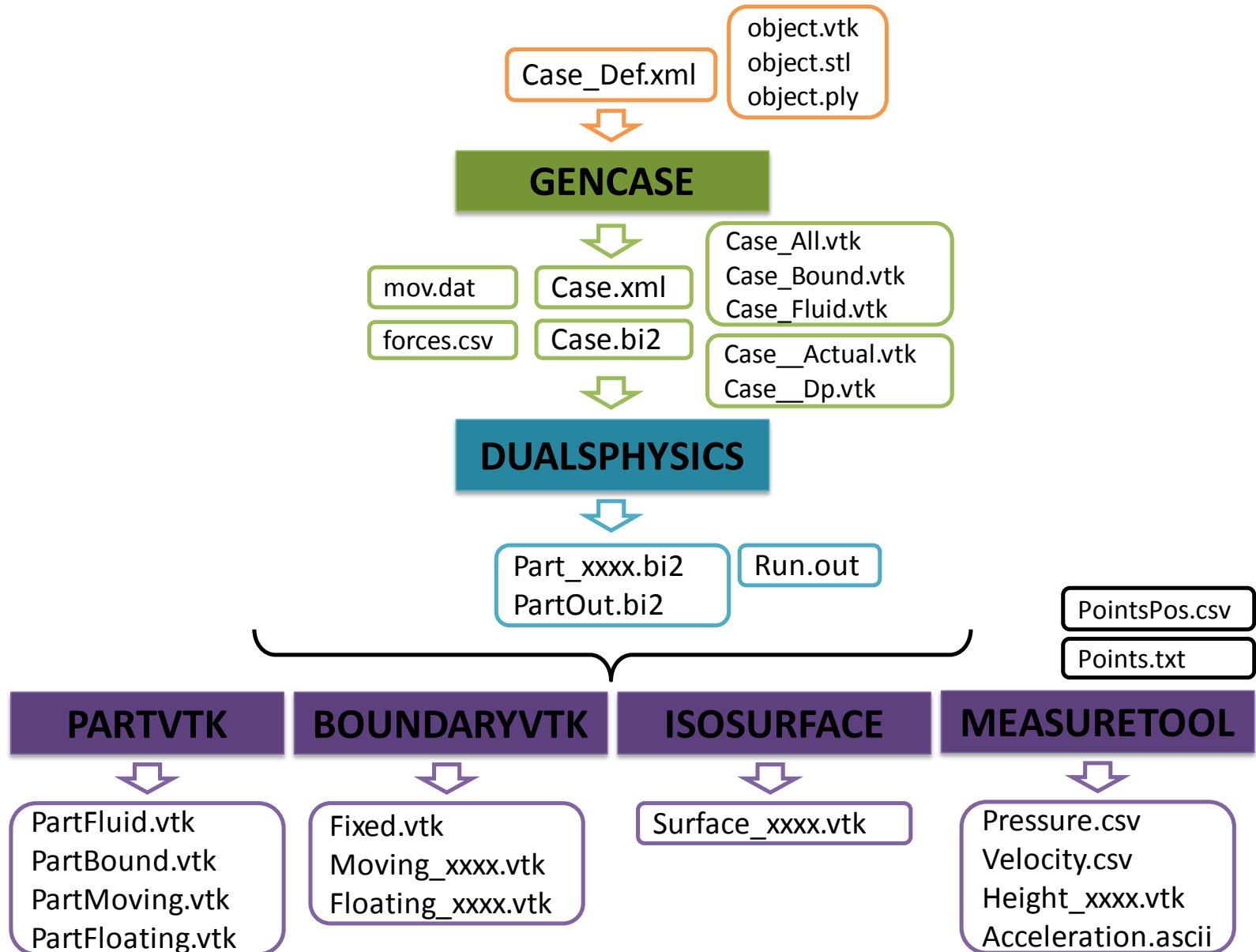


# Outline

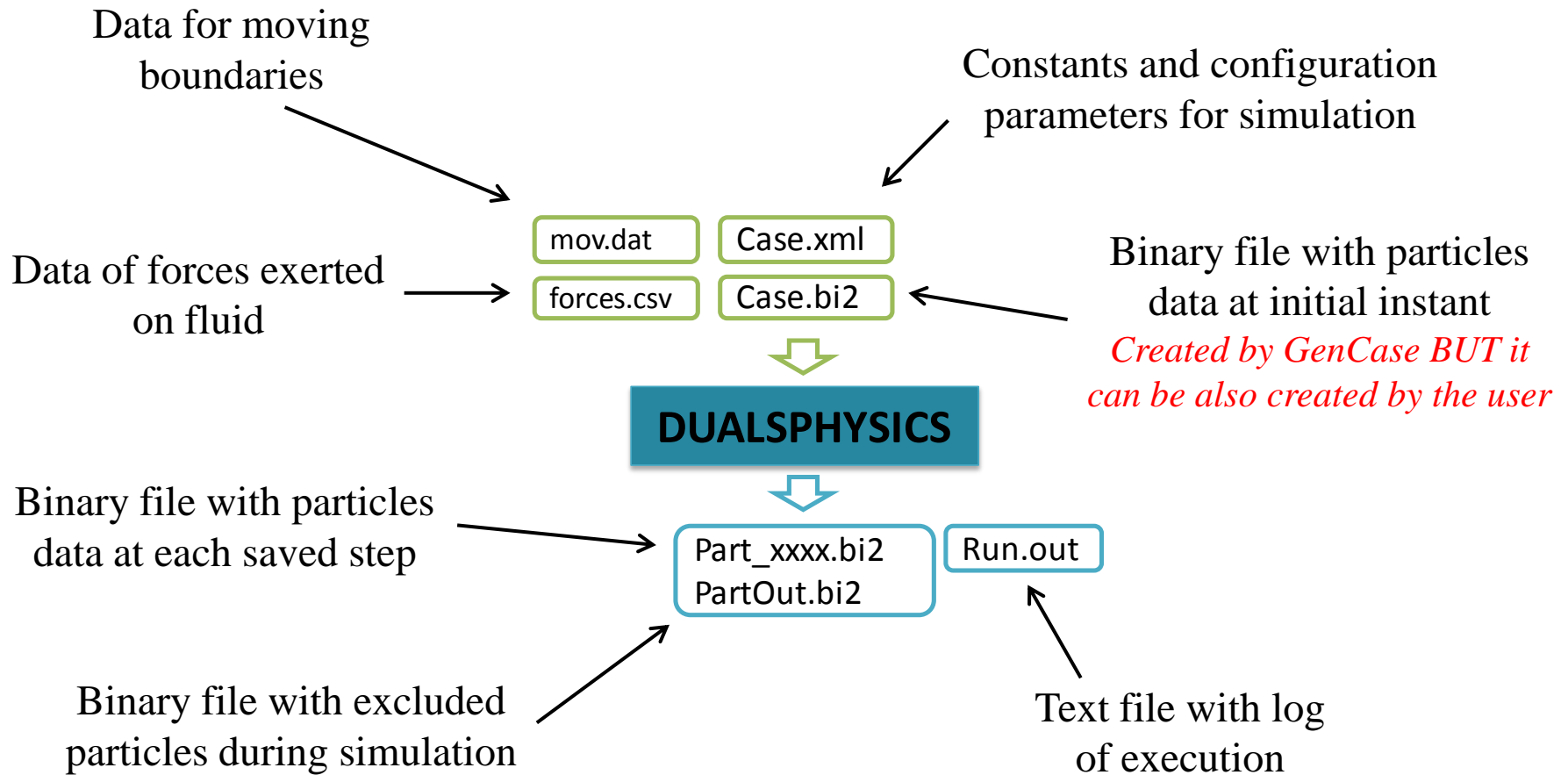
1. DualSPHysics
  - 1.1. Origin of DualSPHysics
  - 1.2. Why GPUs?
  - 1.3. DualSPHysics project
2. SPH formulation
3. Structure of code
  - 3.1. Steps of simulation
  - 3.2. Source files
  - 3.3. Object-Oriented Programming
  - 3.4. Execution diagram
- 4. Input & output files**
5. Test cases online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

## 4. Input & output files

### Pre-Processing



## 4. Input & output files



## 4. Input & output files: Format files

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 edited easily using any text editor.

Case.bi2

Part\_xxxx.bi2  
PartOut.bi2

### BINARY File

- Binary format consumes at least six times less memory than text (ASCII) format.
- Reading or writing is several times faster using a binary format.
- A special code is required to read the data (JPartData.cpp/.h).

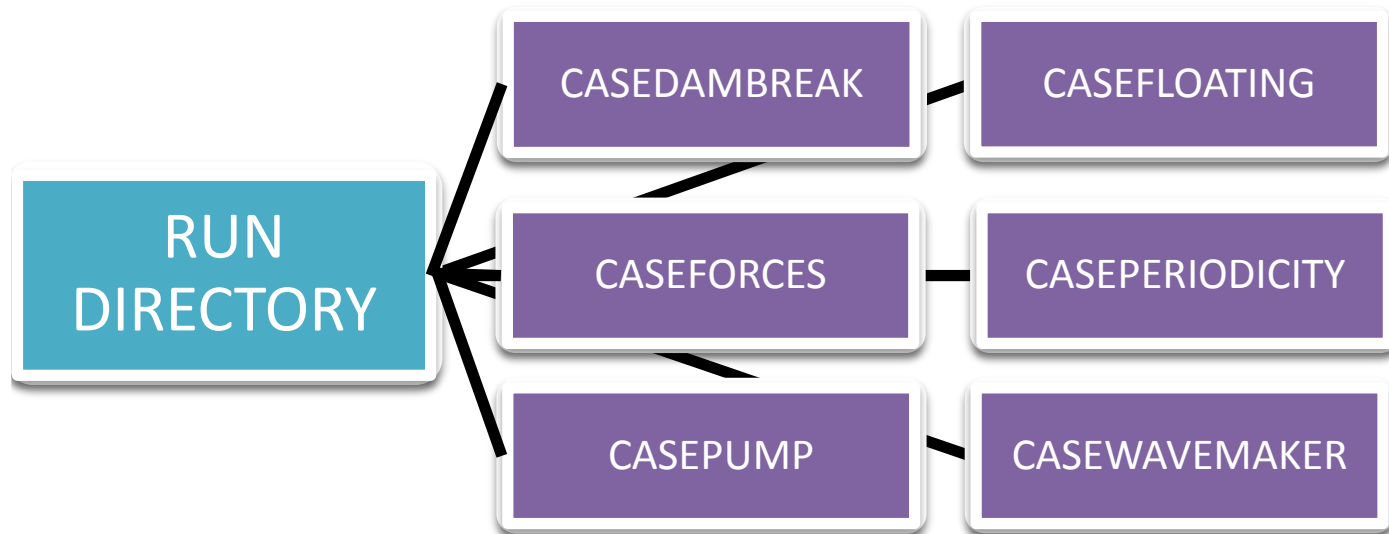
# Outline

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. Novelties in next release v4.0

## 5. Test cases online

Some demonstration cases are included in the DualSPHysics package to show:

- Fixed boundaries
- Moving boundaries & piston wavemaker
- Complex geometries imported from external files
- Periodic boundaries
- Floating objects

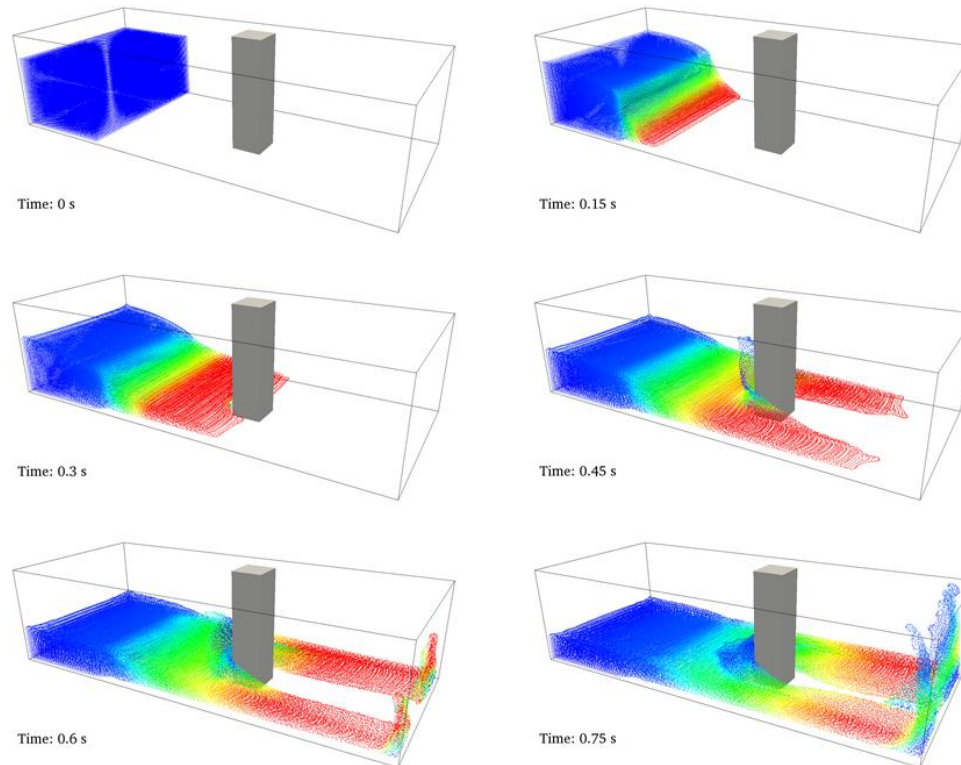




## 5. Test cases online

### CASEDAMBREAK:

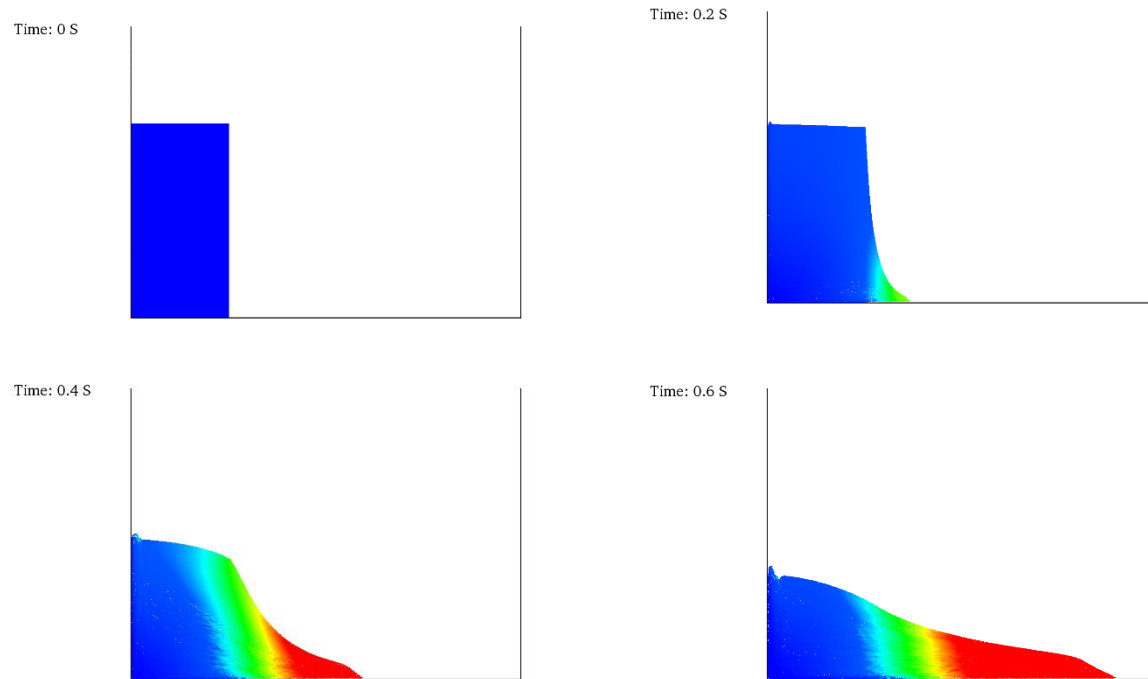
The first test case consists of a 3-D dam break impacting on a structure inside a tank. There are no moving boundaries in this simulation.



## 5. Test cases online

### CASEDAMBREAK:

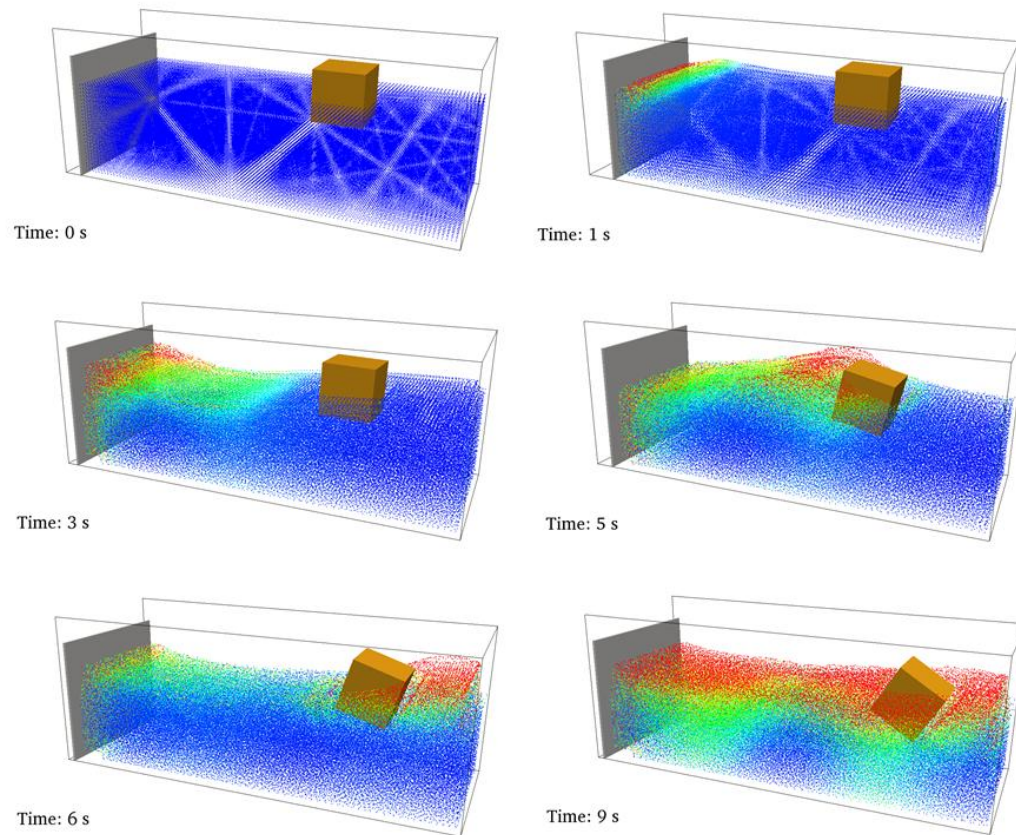
A validation in 2-D is also included for a different dam break where the experimental data of Koshizuka and Oka, 1996 can be numerically reproduced



## 5. Test cases online

### CASEFLOATING:

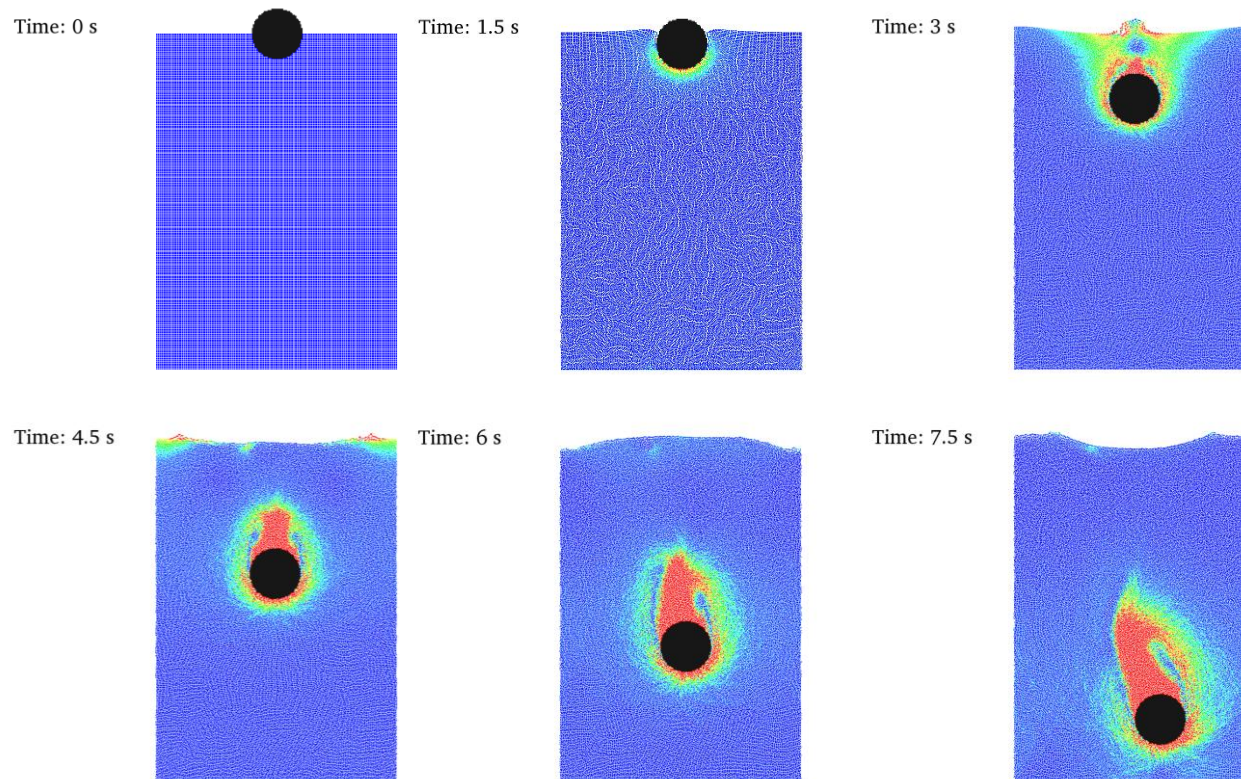
A floating box moves due to waves generated with a piston



## 5. Test cases online

### CASEFLOATING:

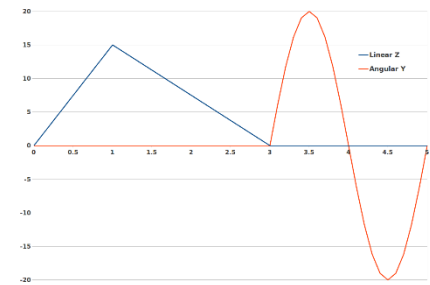
The 2-D case of fluid-structure interaction is also provided. The text files included in the folder contains the experimental displacement and velocity of experiments in Fekken, 2004 and Moyo and Greenhow, 2000.



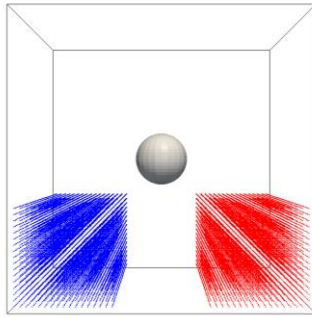
## 5. Test cases online

### CASEFORCES:

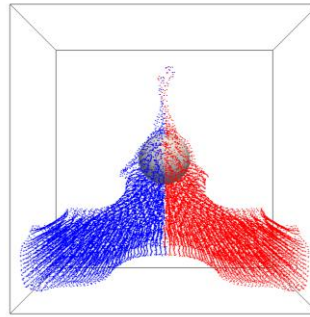
This is a new test case where external forces are applied to the system. The external forces can be loaded from the files `varAccInputFile_0.csv` and `varAccInputFile_1.csv`.



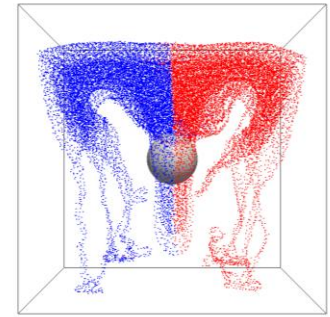
Time: 0 s



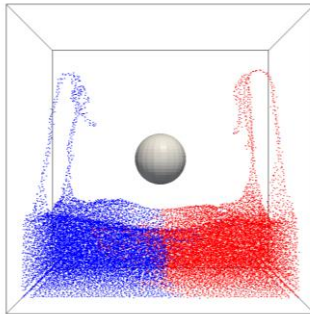
Time: 1 s



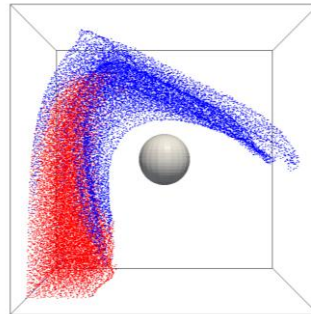
Time: 2 s



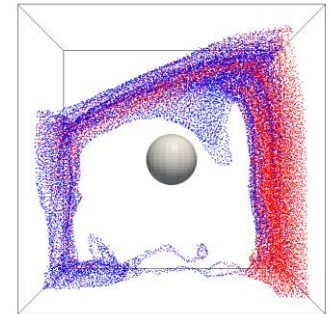
Time: 3 s



Time: 4 s



Time: 5 s

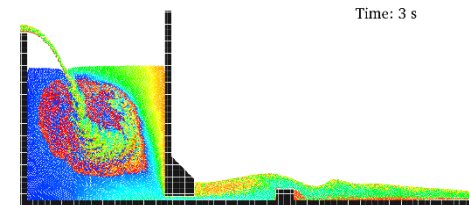
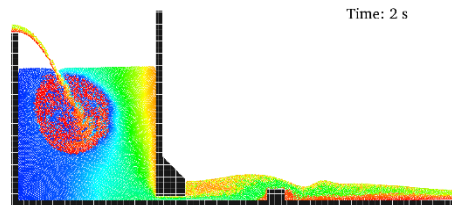
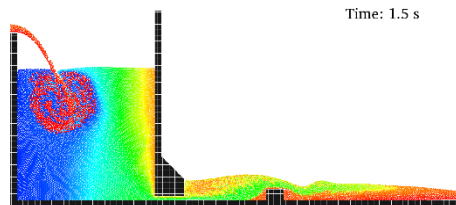
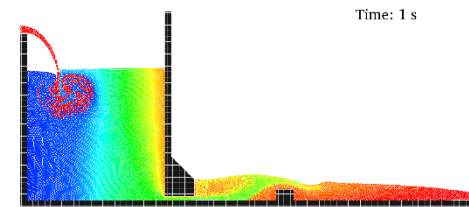
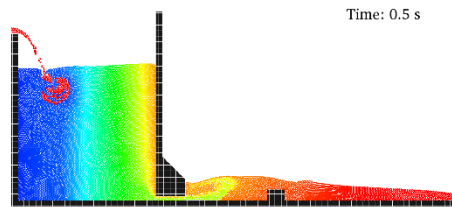
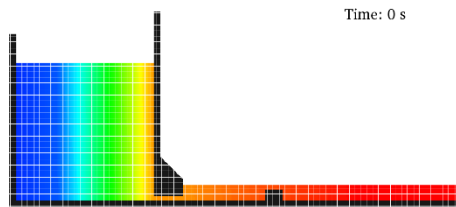




## 5. Test cases online

### CASEPERIODICITY:

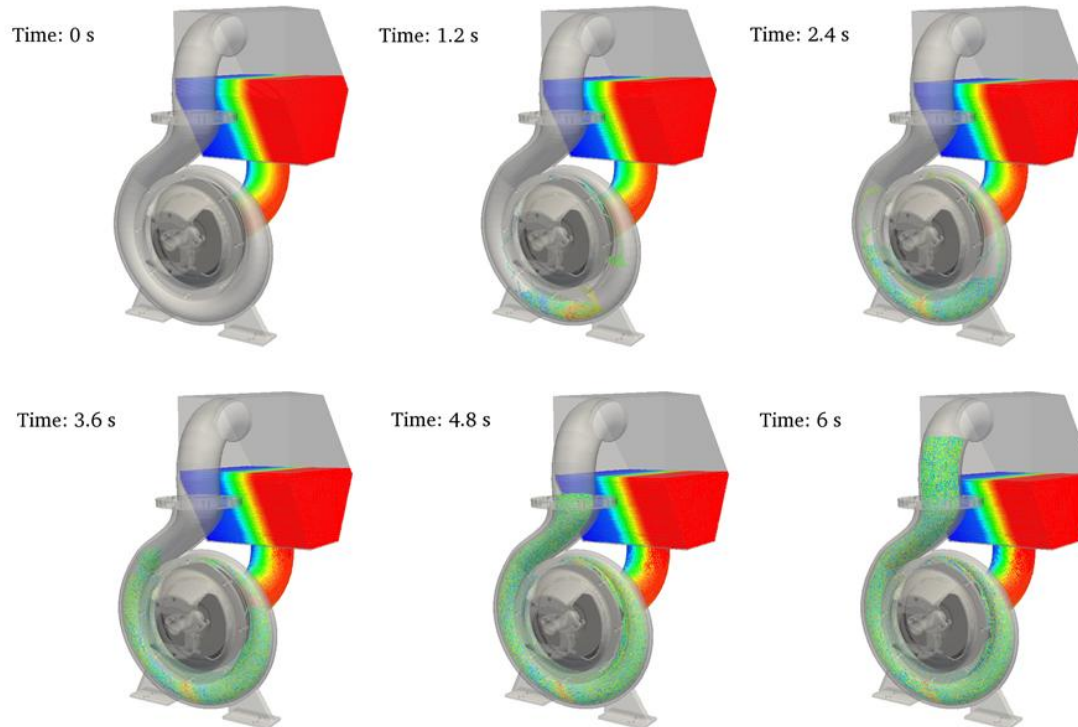
This test case is an example of periodicity applied to 2D case where particles that leave the domain through the right side are introduced through the left side with the same properties but where the vertical position can change (with an increase of +0.3 in Z-position).



## 5. Test cases online

### CASEPUMP:

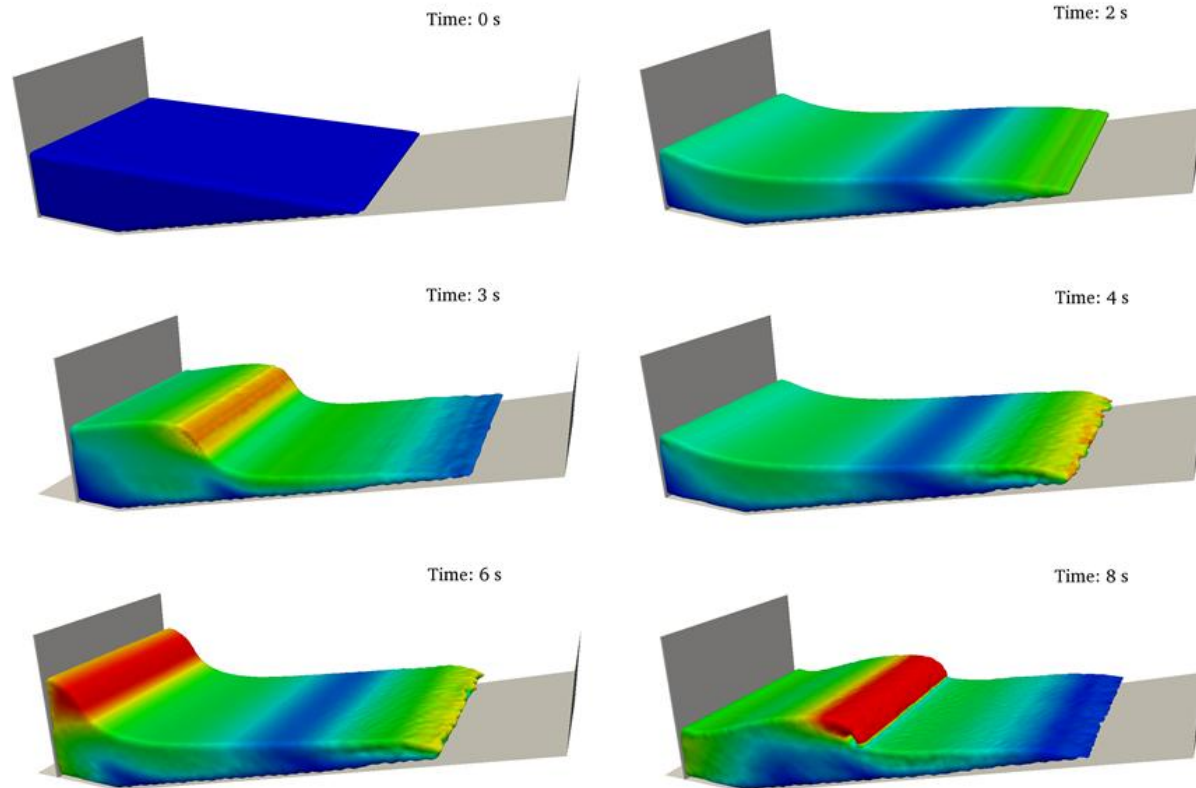
This 3D test case loads an external model of a pump with a fixed (pump\_fixed.vtk) and a moving part (pump\_moving.vtk). The moving part describes a rotational movement and the reservoir is pre-filled with fluid particles.



## 5. Test cases online

### CASEWAVEMAKER:

CaseWavemaker simulates several waves breaking on a numerical beach. A wavemaker is performed to generate and propagate the waves. In this test case, a sinusoidal movement is imposed to the boundary particles of the wavemaker





# Outline

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. **Novelties in next release v4.0**

## 6. Novelties in next release v4.0

- 1) New CPU structure that mimics the GPU threads.
- 2) New GPU structure with the code better organized  
Easy to follow and modify by the user.
- 3) Double precision implementation; several options.
- 4) Floating bodies formulation is corrected.
- 5) Delta-SPH of Molteni&Colagrossi plus others (to be confirmed)
- 6) Shifting algorithm
- 7) New wave generation (regular, irregular by given H/Hs, T/Tp and depth)
- 8) Source code of DEM (Discrete Element Method )
- 9) Multi-phase liquid-sediment solver release v3.2 (September 2015)

Check the website for news / releases / info