INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids (2010)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/fld.2481

Neighbour lists in smoothed particle hydrodynamics

J. M. Domínguez[1], A. J. C. Crespo[1,*,†], M. Gómez-Gesteira[1] and J. C. Marongiu[2]

[1]*EPHYSLAB Environmental Physics Laboratory, Universidad de Vigo, Ourense, Spain*
[2]*Department of Hydraulic Research, Andritz-Hydro, Vevey, Switzerland*

SUMMARY

Since smoothed particle hydrodynamics (SPH) is based on interactions with the closer neighbouring particles, implementing the neighbour list is a key point in terms of the high performance of the code. The efficiency of the method depends directly on how to build and use the neighbour list. In the present work, the available searching algorithms for SPH codes are analyzed. Different gridding algorithms are evaluated, the gains in efficiency obtained from reordering of particles is investigated and the cell-linked list and Verlet list methods are studied to create the neighbour list. Furthermore, an innovative searching procedure based on a dynamic updating of the Verlet list is proposed. The efficiency of the algorithms is analyzed in terms of computational time and memory requirements. Copyright © 2010 John Wiley & Sons, Ltd.

Received 2 December 2009; Revised 15 September 2010; Accepted 9 October 2010

KEY WORDS: SPH; meshless methods; neighbour list; Verlet list

1. INTRODUCTION

Smoothed particle hydrodynamics (SPH) is a meshless method developed during 1970s in astrophysics [1, 2], which has been applied to a variety of fields, such as solid simulation [3, 4]. Despite being the most widely established mesh-free method, SPH was not used in fluid dynamics until the beginning of the 1990s [5]. In particular, the method has been successfully applied to the study of free-surface flows [6–18]. More than a decade later, SPH is quickly approaching its mature stage and the accuracy, stability and adaptivity of the model have reached an acceptable level for practical engineering applications. Some complete revisions on classical SPH can be found at [19–22].

In fluid dynamics, SPH integrates the dynamical equations of motion for each fluid 'particle'. The 'particles' represent points of the domain where physical properties are solved. These physical quantities are computed as an interpolation of the values of the closest neighbouring particles and are updated for each time step along the numerical simulation. The dynamical equations in fluid dynamics represented by partial differential equations are moved to integral equations, which in SPH notation are transformed in sums over neighbouring points. A kernel function is used to determine the influence domain where each particle interacts with its neighbours. Thus, each particle is only influenced by the particles in its close neighbourhood, in such a way that only particles at a distance shorter than nh (n depends on the kernel choice and h is the smoothing length) should be evaluated. For the sake of clarity a cut-off limit of $2h$ will be considered from now on.

*Correspondence to: A. J. C. Crespo, EPHYSLAB Environmental Physics Laboratory, Universidad de Vigo, Ourense, Spain.
†E-mail: alexbexe@uvigo.es

Table I. Generic modules of an SPH code using cell-linked lists (CLL) and Verlet lists (VL).

SPH step	Cell-linked list (CLL)		Verlet list (VL)	
Neighbour List	Cells division Particles in cells	1.7%	Cells division Particles in cells Search of neighbours in adjacent cells	20.5%
Force Computation	Neighbours search Interaction forces	96.7%	Neighbour list construction Load neighbour list Interaction forces	78%
System Update	Solve variables of next step	1.6%	Solve variables of next step	1.5%

Thus, the determination of which particles are inside the interaction range requires the computation of all pair-wise distances, a procedure with high requirements in terms of computational time for large domains. The efficiency of this procedure, which involves a number of interactions on the order of N^2, is so poor that this brute force evaluation of interactions can only be used in academic exercises as pointed out in [23].

Different approaches coexist in SPH to create a list of neighbours. Here, we will focus on two of them, the cell-linked list (CLL) and the Verlet list (VL). There are more methods, such as oct-tree methods that are used mostly in astrophysical problems [24], where different variable time scales and long-range interactions like gravity take place. The choice of each particular method affects both the search of neighbours and the data-flow diagram of the code (see Table I). In general, an SPH code can be generally split into three main parts: *Neighbour List*, *Force Computation* and *System Update*. Although both approaches will be described in detail in Section 2, the main features of both methods will be briefly introduced here. In the CLL, the computational domain is divided into cells of side $2h$ (cut-off limit), then particles are stored according to the cell they belong to. Thus, an array of particles reordered through the cells is obtained. This array is loaded on the *Force Computation*, where each particle of interest only looks for its potential neighbours in the adjacent cells (the candidates to be neighbours). When the distance between two particles is less than $2h$, then a real neighbour of the particle of interest is found and forces will be computed. In this case, a list will be associated with each cell. In the simplest version of VL, the domain is also divided into cells of size $2h$, and the particles are also allocated in an array where they are grouped according to the cell they belong to. However, in this case, a new array is obtained from the previous one, the properly named neighbour list includes all the particles of the adjacent cells at a distance shorter than $2h$ for each particle of the domain. Thus, this so-called Verlet list [25] contains the real neighbours of each particle. This array of neighbours is loaded during the force computation where only the computation of the interaction forces between neighbouring particles is carried out. Percentages in Table I are referred to the 100% total runtime of the simulation which implies the execution of several time steps. These values are related to a given solution where particle forces are computed once. The creation of the VL is more complex since it involves all calculations needed to generate CLL and the additional construction of the Verlet list. However, this list can be kept during several time steps considering cells of size slightly higher than $2h$, as it will be shown

The aim of the manuscript is to analyze the efficiency of different methods to create a neighbour list. Thus, the same initial condition and physical assumptions will be considered in the different runs. The only difference among runs will be the procedure to calculate and organize the neighbour list. Two different approaches (CLL and VL) will be used to create the neighbour list. Both approaches are initially similar in such a way that they need to grid the computational domain to identify particles in cells. A particle will only interact with particles placed in the same or adjacent cells. Thus, different gridding algorithms will be developed and analyzed in terms of computational runtime and memory requirements. Reordering particles constitutes a possible improvement in runtime not only in the creation of the list but also in its further use in force computation.

DOI: 10.1002/fld

The advantages and disadvantages of ordering and different ordering procedure will be analyzed. Finally, the performance of CLL and VL methods will be compared. Improvements to the classical VL will be proposed and assessed.

2. METHODS

2.1. The model

SPHysics is an SPH code created by researchers at the Johns Hopkins University (U.S.), the University of Vigo (Spain) and the University of Manchester (U.K.). The code is available for public use at http://www.sphysics.org. A complete description of the software is found in the SPHysics user's guide [26]. Different formulations (kernel function, viscosity treatment, time-step algorithm, boundary conditions, etc.) can be used selecting the available compiling options. The following options will be considered throughout the test.

The cubic spline kernel developed in [27] is used to determine the intensity of the interaction between neighbouring particles

$$W_{ab}=\frac{1}{\pi h^3}\begin{cases}1-\frac{3}{2}q^2+\frac{3}{4}q^3 & \text{if } 0\leqslant q\leqslant 1\\ \frac{1}{4}(2-q)^3 & \text{if } 1\leqslant q\leqslant 2\\ 0 & \text{otherwise}\end{cases} \tag{1}$$

where $q=r/h$, being r the distance between particles a and b, and h the smoothing length. Furthermore, the tensile instability correction proposed by Monaghan [28] is applied to avoid particle clumping.

The momentum equation given by Monaghan [29] is used to describe the forces exerted on particles. The artificial viscosity defined in [29] is used to represent viscous effects. Thus, in SPH notation, the momentum equation is written as

$$\frac{d\vec{v}_a}{dt}=-\sum_b m_b\left(\frac{P_b}{\rho_b^2}+\frac{P_a}{\rho_a^2}+\Pi_{ab}\right)\vec{\nabla}_a W_{ab}+\vec{g} \tag{2}$$

where $\vec{g}=(0,\ 0,\ -9.81)\,\mathrm{ms}^{-2}$ is the gravitational acceleration and Π_{ab} is the artificial viscous term

$$\Pi_{ab}=\begin{cases}\dfrac{-\alpha\overline{c_{ab}}\mu_{ab}+\beta\mu_{ab}^2}{\overline{\rho_{ab}}}, & \vec{v}_{ab}\cdot\vec{r}_{ab}<0\\ 0, & \vec{v}_{ab}\cdot\vec{r}_{ab}>0\end{cases} \tag{3}$$

with $\mu_{ab}=h\vec{v}_{ab}\vec{r}_{ab}/(\vec{r}_{ab}^2+\eta^2)$, where $\vec{r}_{ab}=\vec{r}_a-\vec{r}_b$ and $\vec{v}_{ab}=\vec{v}_a-\vec{v}_b$; being $\vec{r}$ and $\vec{v}$ the position and the velocity corresponding to the particle (a or b); $\overline{c_{ab}}=c_a+c_b/2$, $\eta^2=0.01\,h^2$, $\alpha=0.1$ and $\beta=0$.

The continuity equation given by Monaghan [29] is used to calculate changes in the fluid density.

$$\frac{d\rho_a}{dt}=\sum_b m_b\vec{v}_{ab}\vec{\nabla}_a W_{ab} \tag{4}$$

This approach based on the time variation of the density is used instead of a weighted summation of mass terms, since the second approach underestimates the density value near fluid interfaces, free-surfaces or boundary limits.

In this SPH approach, the fluid is treated as weakly compressible so an equation of state can be considered to calculate the fluid pressure in terms of density following [30, 31]:

$$P=B\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma}-1\right] \tag{5}$$

DOI: 10.1002/fld

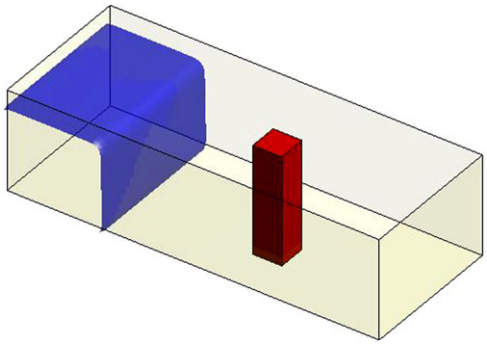

Figure 1. Initial configuration.

with $\gamma=7$ and $B=c_0^2\rho_0/\gamma$, $\rho_0=1000\,\mathrm{kg\,m^{-3}}$ is the reference density and $c_0=c(\rho_0)$, the speed of sound at the reference density.

The XSPH correction [32] is used to change the position of the particles. This approach modifies the velocity of the particle adding a velocity that is calculated following the average velocity in its neighbourhood

$$\frac{\mathrm{d}\vec{r}_\mathrm{a}}{\mathrm{d}t}=\vec{v}_\mathrm{a}+\varepsilon\sum_\mathrm{b}\frac{m_\mathrm{b}}{\overline{\rho_\mathrm{ab}}}\vec{v}_\mathrm{ba}W_\mathrm{ab} \tag{6}$$

with $\varepsilon=0.5$ and $\overline{\rho_\mathrm{ab}}=(\rho_\mathrm{a}+\rho_\mathrm{b})/2$.

The Verlet algorithm [25, 26] is used in the simulations to advance the solution in time. A variable time step, δt, is considered according to [29]. The calculation of the time step involves the Courant condition, the force terms and the viscous diffusion term

$$\delta t=0.3\,\min(\delta t_f,\delta t_{cv}),\quad \delta t_f=\min_\mathrm{a}\left(\sqrt{\frac{h}{|\vec{f}_\mathrm{a}|}}\right),\quad \delta t_{cv}=\min_\mathrm{a}\frac{h}{c_\mathrm{s}+\max_\mathrm{b}\left|\dfrac{h\vec{v}_\mathrm{ab}\vec{r}_\mathrm{ab}}{\vec{r}_\mathrm{ab}^{\,2}}\right|} \tag{7}$$

where f_a is the force term and c_s the speed of sound.

Solid walls are simulated using dynamic boundaries following [26, 33].

2.2. *Case study*

The case under study is the collapse of a dam break due to gravity and its interaction with a rectangular structure. A similar numerical setup was used in previous studies to show the accuracy of the SPHysics model [13] and the marker and cell model [34]. Actually, this testcase is considered as a valuable benchmark for the whole SPH community. The schematic setup is depicted in Figure 1. The container is a rectangular box 160 cm long, 61 cm wide and 75 cm high. The volume of water initially contained at one end of the box was 40 cm long × 61 cm wide × 30 cm high. A tall structure, which was 12 cm × 12 cm × 75 cm in size, was placed 50 cm downstream of the water mass at rest and 24 cm from the nearest sidewall of the tank. The bottom of the container was assumed to be initially dry. Both fluid and solid particles were initially placed on a staggered grid as described in [13].

SPHysics code was validated in [13], where the CLL was implemented. All the procedures tested from now on provide the same numerical accuracy in reproducing the testcase data. The gridding algorithms, the effect of reordering particles and the choice of different neighbour lists give exactly the same neighbourhood, giving rise to the same agreement between experimental and numerical data. The only differences are related to the implementation technique, computational time and memory requirements.

DOI: 10.1002/fld

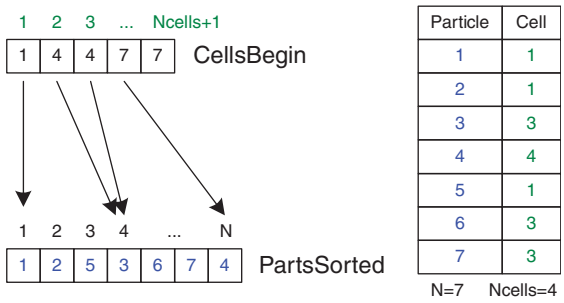

Figure 2. Data definition using *Sliding vector* method.

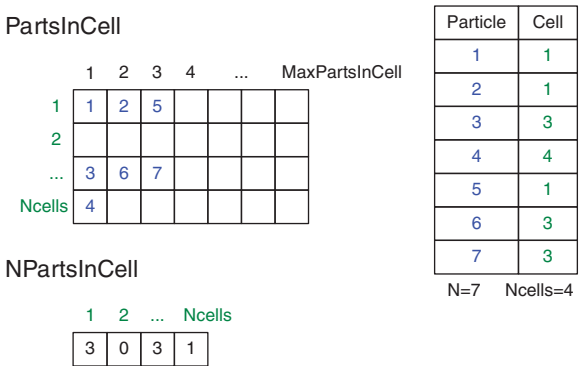

Figure 3. Data definition using *Static matrix* method.

2.3. *Gridding algorithms*

As mentioned above (Table I), the search of potential neighbours involves two initial steps, namely; *cells division* and *particles in cells*. This part of the procedure is common to CLL and VL. Different gridding algorithms will be considered in this section.

In algorithm A (Sliding dynamic vector) particles are stored in an array called *PartsSorted* where they are grouped according to the cell they belong to. Meanwhile, the array called *CellsBegin* indicates the position where particles of each cell starts. Figure 2 presents an example showing how these two arrays are defined. The procedure to build *CellsBegin* and *PartsSorted* consists in inserting each particle according to its cell into *PartsSorted* displacing the pre-existing particles to get a new allocation for the new particle. Meanwhile, *CellsBegin* is re-adjusted to be pointed at the beginning of each cell.

The algorithm B (Preconditioned dynamic vector) is a variant of the previous one. Data are stored following the same procedure as in the Sliding dynamic vector, but *CellsBegin* and *PartsSorted* are built differently in this case. Now extra arrays are implemented to count in advance the number of particles inside each cell so when inserting particles in *PartsSorted*, the memory space is already known and displacing the pre-existing particles is not needed.

In algorithm C (Static matrix) the matrix called *PartsInCell* (see Figure 3) is implemented, each row corresponds to a cell and all particles of the cell are stored. As the maximum number of particles inside a cell is not previously known, a constant value *MaxPartsInCell* is externally fixed. The amount of particles of each cell is stored in the array called *NPartsInCell*. Thus, the main disadvantage of the method is the need of allocating the same memory space for all the cells, independent of the real amount of particles in the cell.

The algorithm D (Linked list) generates a simple linked list using the particles of each cell (see Figure 4). A node represents a particle and contains a pointer to be linked with the next particle of the same cell.

DOI: 10.1002/fld

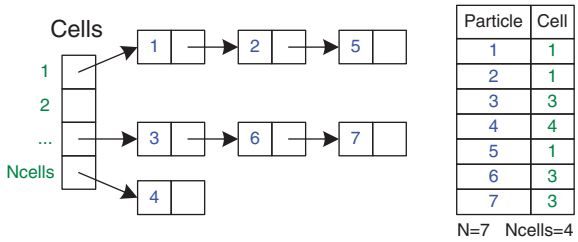

Figure 4. Data definition using the *Linked list* method.

2.4. *Reordering particles*

Particles are stored following an initially given order and once the system evolves their memory allocation of the particle data do not change during the simulation. A potential method to speed up calculations consists in reordering particles. This means that particles must be stored following the order of the cells they belong to and their memory allocation can change at each time step.

Particle data will be closer in the memory space, improving the access to the CPU memory and decreasing the computational time when computing SPH forces. Note that reordering particles does not affect to the force interaction since each particle keeps the same neighbourhood and particle interactions will be exactly the same.

Thus, reordering particles presents two main advantages. The first one is the mentioned improved access to particle data in the CPU memory since the access pattern is more regular and the coalescence is higher. The other advantage is the ease to look for the neighbours of a particle in the adjacent cells since only the first particle of each cell must be known. This means that when the first particle of a cell and the first one of the following cell are known, all the particles contained in the adjacent cell are easily identified. On the other hand, the main drawback of the method is the increase in memory requirements.

Different ways to build the order of the cells can be followed (*SortX*, *SortXY* and *SortXYZ*). The traditional way *SortX* [26] consists in sorting cells depending on the X-coordinate, then the Y-direction and finally in the Z-direction. Hence, the index of a cell *icell* will be defined as

$$icell = cz^{*}ncx^{*}ncy + cy^{*}ncx + cx \tag{8}$$

being cx, cy and cz the position of the cell in the X-, Y- and Z-directions and ncx, ncy the number of cells in the X- and Y-directions. Thus, the proximity among particles is only assured in the X direction.

SortXY and *SortXYZ* are examples of the so-called *Z-Order* (first proposed in 1966 by Morton [35]). In this method, interleaving the bits of the coordinates, a better proximity in all directions is achieved. In *SortXY*, the *Z-Order* is applied in the plane XY getting proximity of the particles that belong to it, while *SortXYZ* looks for proximity in any direction.

2.5. *Neighbour lists*

As mentioned above, there are traditionally two different approaches to generate the neighbour list (see Figure 5 for a sketch of both methods).

A CLL can be calculated by means of the following steps:

(a) The computational domain is divided into cells of side $2h$. This is usually called as kernel support radius.
(b) Particles are stored according to the cell they belong to.

The sketch of this method is shown in Figure 5(a). The possible neighbours (grey coloured dots) of a particle 'a' are placed in the adjacent cells, but only those particles placed at a distance shorter than $2h$ (dark colour) interact with the particle a.

The main advantage of a Verlet list [25] is the possibility of keeping the same list during several consecutive time steps. Although, the technique is well known [23], it has some intrinsic limitations that must be eliminated to obtain a more efficient code. Here, we will first describe the classical

DOI: 10.1002/fld

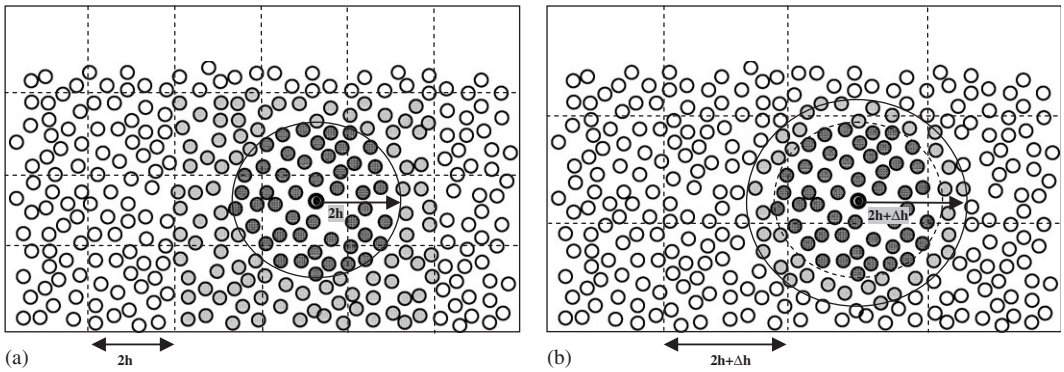

Figure 5. Sketch of the different methods used to create the neighbour list: (a) cell-linked list (CLL) and (b) Verlet list (VL).

method and then the possible improvements. Let us assume that in the classical Verlet list (VL_C from now on) the list is required to remain fixed for the next C time steps:

(a) The computational domain is divided into cells of side $2H = 2h + \Delta h$, being $\Delta h = v(2V_{\max} \cdot C \cdot dt)$, where $V_{\max}$ is the maximum velocity of any particle of the system, multiplied by 2 since the worst situation appears when two particles with the maximum velocity are moving apart and v is a parameter slightly higher than 1. C is the number of time steps the list is going to be kept. Note that this part of the method is common to CLL when $\Delta h = 0$.
(b) Search for potential neighbours at the adjacent cells. When the distance between the particle of interest, a, and another particle, b, is less than $2H (r_{ab} < 2H)$ that particle is added to the list of potential neighbours. Note that the particle is not only a candidate to interact with 'a' during the following C time steps, but particles with $r_{ab} < 2h$ will interact during the present time step.
(c) The list of potential neighbours is loaded and kept during the following C time steps. Only those particles with $r_{ab} < 2h$ will interact. Note that particles move in time, in such a way that from the initial set of candidates only a percentage of the total interacts each time step, and the interacting particles can change every time step.

The method presents several drawbacks. On the one hand, the list is not checked every time step and particles can leave or enter the neighbourhood without being detected. The imposed condition on Δh depends on $V_{\max}$ and dt, which do not remain constant during the C time steps. $V_{\max}$ can vary due to flow acceleration and dt is variable. This fact can give rise to inaccuracies in calculations. This effect can be prevented by using $v = 1.2$ in the definition of Δh, although this implies higher memory requirements and will slow down the code since the number of 'false' candidates increases. On the other hand, the method is inefficient in terms of computational time. An initial condition is imposed on Δh, but that condition considers the worst situation at the first step from the C steps the list is kept. However, velocity can decrease during the C time steps and the interaction between two particles with the maximum velocity does not necessarily take place. In summary, the list is likely to remain valid for more than C time steps.

The following Verlet list (VL_X from now on) is proposed. In this case Δh is calculated in the same way as in VL_C. However, the number of steps the list is kept (X instead of C) is only tentative, assumed at the first time step, but it can be longer or shorter depending on the calculation. The position of all particles in the domain is also recorded at the first time step. During the following time steps the position of the particles is checked. When the distance travelled by any particle from the first step is longer than $\Delta h/2$ the Verlet list is recalculated and assumed to last for X time steps. Note that the drawbacks mentioned above disappear with this approach. When a particle enters or leaves the neighbourhood of particle a the list is recalculated, even when the number of steps is less than X. Furthermore, the list can be kept even when the number of steps is higher than X if no particle has left or entered the neighbourhood of any particle a. Finally, $v = 1.0$ is

DOI: 10.1002/fld

assumed in Δh calculation because no extra distance is added to $2H$ since the real position of the particles is checked every time step.

The sketch of this method is shown in Figure 5(b). The possible neighbours (grey coloured dots) of the particle are placed in the adjacent cells. Only those particles placed at a distance shorter than $2H$ (dark colour) will potentially interact with particle a and will be included in the list. Note that during the first time step only the particles marked with black dots will interact with particle a.

3. RESULTS

3.1. Comparison among gridding algorithms

As mentioned above, gridding is common to CLL and VL methods in such a way that the comparison among algorithms will be independent of the method. Here, we will compare the runtime and memory requirements of the four algorithms assuming a neighbour search based on CLL.

Figure 6 shows the computational runtimes using the different gridding methods as a function of the number of particles N. The runtime increases with the number of particles, although at a different rate depending on the algorithm. Note that the computational time (in hours) of algorithm A (Sliding dynamic vector, darker line in Figure 6) is considerably higher than the rest, e.g. the method is more than three times slower than the rest for a simulation with 1 50 000 particles. While the runtime increases quasi-linearly with the number of particles for the rest of the cases, it increases almost exponentially for algorithm A. The algorithm D (Linked list) becomes the faster one, which is especially patent when N increases.

Figure 7 shows the memory needed to allocate the arrays concerning the neighbour list. Obviously, the memory requirements increase with the number of particles. Note that memory requirements in case C (Static matrix) is considerably higher (at least 20 times) than in the rest of the cases. For the rest, the method A (Sliding vector) provides the best use of memory.

In summary, the algorithm A is easy to implement and provides an optimal use of the memory; however, it is slow since inserting particles penalizes the computational time. Algorithm B has higher memory requirements than A due to the use of the auxiliary arrays, however, it is much faster than algorithm A. The algorithm C (Static matrix) is easy to be implemented but requires a big amount of memory space since a fixed memory space of particles for each cell is stored at the beginning of the simulation. Finally, the algorithm D optimizes the use of memory since there is no wasted memory and useless memory space is not reserved as in previous cases.

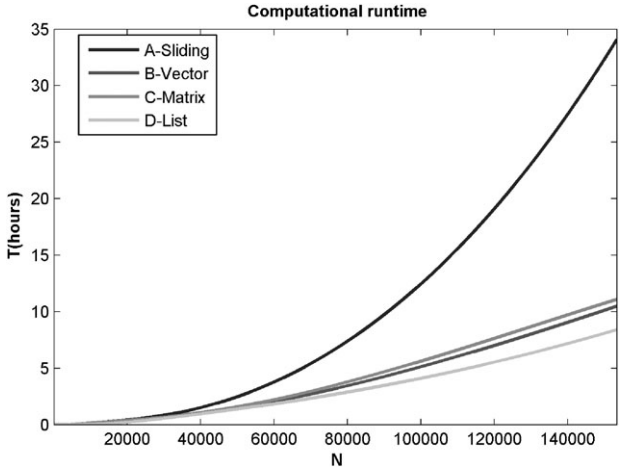

Figure 6. Computational runtime for the different gridding algorithms.

DOI: 10.1002/fld

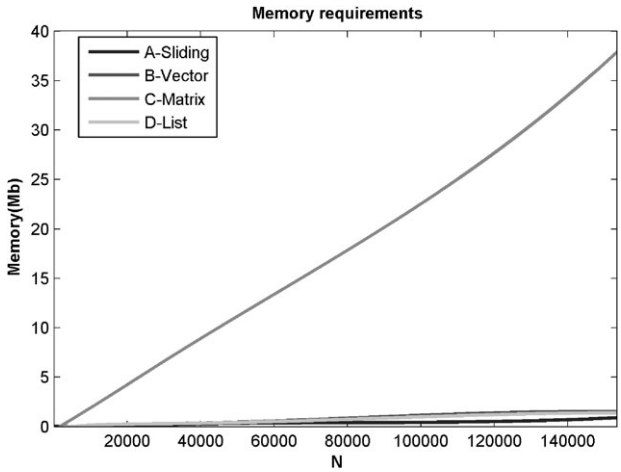

Figure 7. Memory requirements for the different gridding algorithms.

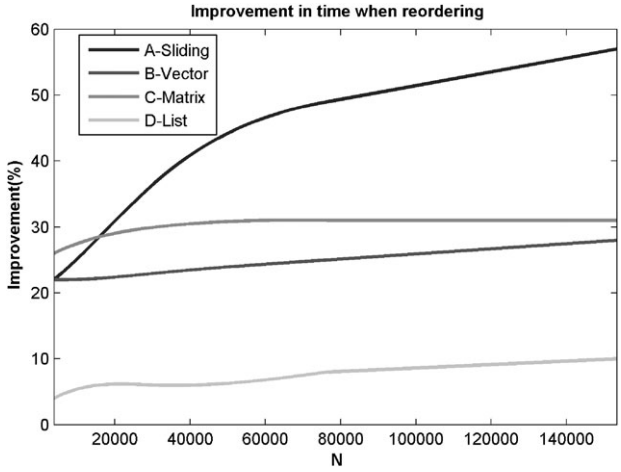

Figure 8. Improvement in time when reordering particles for the different algorithms.

3.2. Reordering particles

Reordering can potentially speed up calculations due to the reasons described in Section 2.4. Figure 8 shows the improvement achieved by reordering particles in the four gridding algorithms. In algorithms B, C and D the improvement in efficiency is higher than 20%. Even, in some particular cases (e.g. algorithm A with 1 50 000 particles) the improvement is close to 60%. The algorithm D (Linked list) presents the lowest improvement since the particle data structure was quite optimized by construction. Note that the improvement compares each sorted method with its previous version. Improvement values after particle reordering were chosen to be shown instead of the absolute times (as was done in Figure 6) because the achieved computational runtimes for the algorithms B, C and D are now equivalent and differences are not observed. Hence, even if the improvement achieved by one algorithm is much higher than for the rest, this does not necessarily mean that the algorithm is better. This is the case, for example, of algorithm A, which can be improved by about 60% by reordering but it should be kept in mind that this algorithm was initially three times slower than the others as shown in Figure 6. These particular results were obtained while using the *SortX* algorithm but similar behaviour was obtained using the other sorting methods (not shown).

The comparison among the different sorting methods is shown in Figure 9 where the gridding algorithm B (Dynamic vector) was used before reordering (Figure 8).

DOI: 10.1002/fld

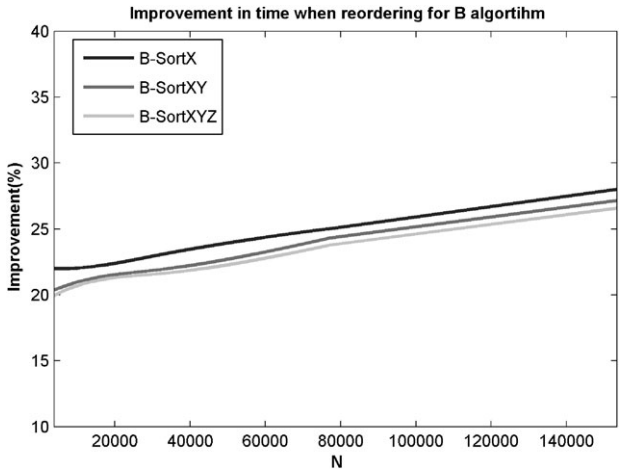

Figure 9. Improvement in time when reordering particles for the algorithm B.

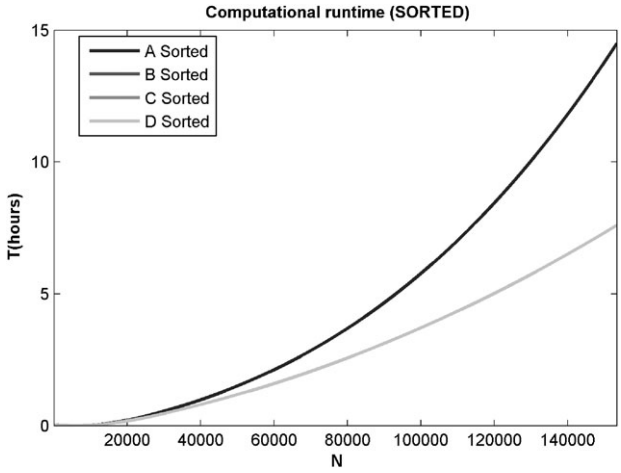

Figure 10. Computational runtime for the different algorithms after reordering particles.

In all cases, the improvement achieved with *SortX* is always higher than using the methods based on *Z-Order*. This method presents some intrinsic limitations in this case. For example the calculation of the index of the cell starting from its coordinates (Equation (8)) is much faster ($\sim$ 100 times) for the traditional case *SortX* than using the *Z-Order* solution. The *Z-Order* calculation of indices is only efficient when carrying out the calculation by means of Boolean operations. However, this can only be achieved when the maximum number of bits of the coordinates is known before compilation in such a way that *ad hoc* coding is needed for every particular case.

Another disadvantage is the need to compute more cells than necessary because the number of cells in each direction should be a power of two. Thus, additional cells must be added in each direction.

In fact, *Z-Order* is efficient when the random computation of any cell is carried out. However, in the case of SPHysics, a sequential sweep of cells is executed to compute the neighbouring forces, instead of a random one. Thus, even when the implementation of *Z-Order* is optimized for the SPHysics code, the obtained speed up does not reveal a better performance than obtained when using *SortX*.

Figure 10 shows the computational runtime for the different algorithms after reordering particles. Once again, the neighbour search is based on CLL, in such a way that results can be compared with those shown in Figure 6.

DOI: 10.1002/fld

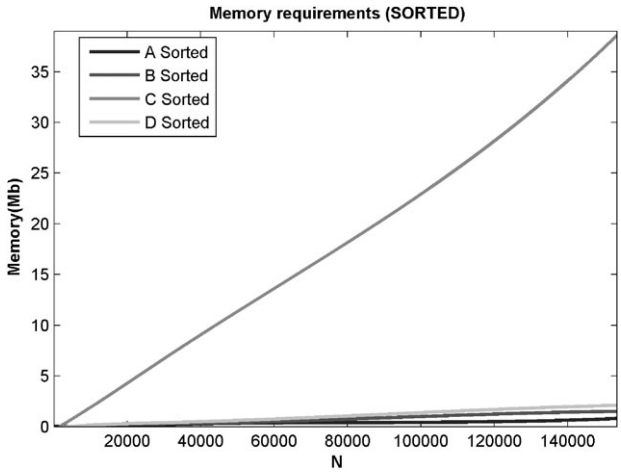

Figure 11. Memory requirements for the different algorithms after reordering particles.

Now, the differences in time among the algorithms B, C and D are negligible (three lines coincide in Figure 10). This is due to the fact that the total runtime of the simulation is dominated by the *Force Computation* (see Table I). Thus, once the particle data access from the CPU memory has been improved by reordering, the computational time dedicated to the neighbour list is similar in the three cases and the total runtime is also similar. Algorithm A is the exception because it is still twice slower due to the insertion of particles when building the Sliding dynamic vector.

In Figure 11, the memory required by the different algorithms increases linearly with N as previously depicted in Figure 7. Quantitatively, the memory requirements have increased slightly for the algorithms A, B and D, while the percentage of memory increase in the algorithm C (Static matrix) is not relevant since the previous allocated memory was huge in comparison to this increase. Once again, the algorithm A is the most efficient in terms of memory requirements although the method is completely inefficient due to its high computational cost as shown in Figure 10. It is necessary to note the fact that the memory allocated for algorithm B is lower than allocated for algorithm D (Linked list). The opposite behaviour was previously observed in Figure 7.

In summary, one can affirm that reordering particles gives rise to a considerable improvement in execution runtime with small additional requirements in memory. This suggests the need of reordering to improve the efficiency of the code. In addition, the three gridding algorithms (B, C and D) have a similar efficiency in terms of runtime, although the algorithm B is the most efficient in terms of allocated memory. This advantage and the fact that algorithm B is easier to be implemented than D makes it a good candidate to be used to elaborate the list of neighbours.

3.3. Comparison between lists

As mentioned above, the list can be created in two different ways (CLL and VL). In addition, the sorted version of method B has been proven to be the most efficient method to grid particles. Thus, this will be the only method used from now on to compare the efficiency of both lists.

Differences in the computational runtime can be observed in Figure 12, where both neighbour lists were applied in SPHysics to run case study described in Section 2. In both simulations, particles have been sorted according to the cells. VL is slower than CLL for any number of particles. In particular, the method is about 13% slower for 1 50 000 particles. This difference is due to the time needed to create the real neighbour list in VL. However, the power of this method has not been properly exploited since the same list can be kept for several time steps, which alleviates the additional burden associated with the creation of the Verlet list.

DOI: 10.1002/fld

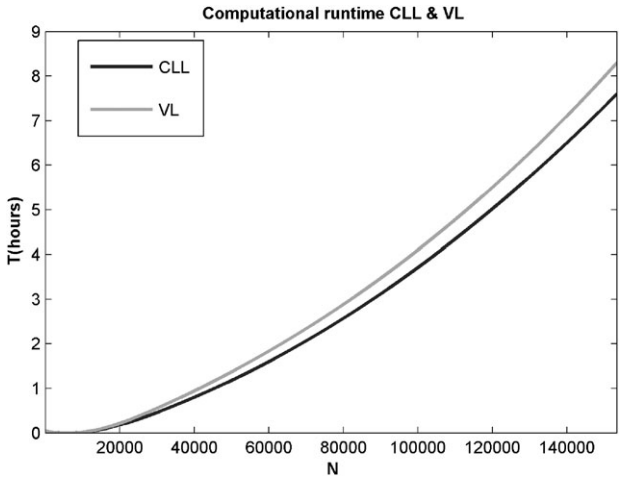

Figure 12. Computational runtime of different approaches for neighbour list.

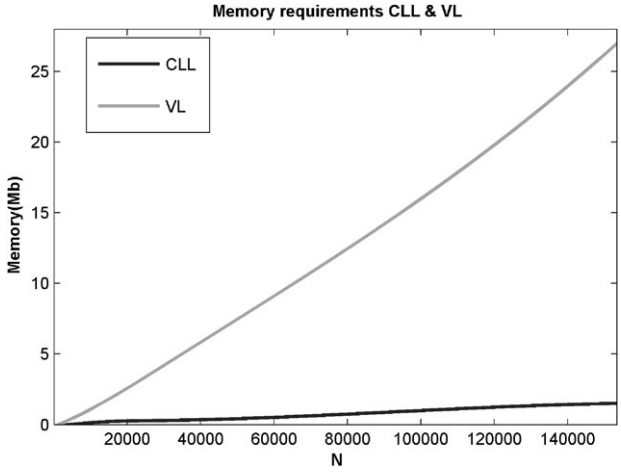

Figure 13. Memory requirements of different approaches for neighbour list.

From the point of view of memory requirements, VL is less efficient than CLL (Figure 13). Thus, for example, the allocated memory is 18 times higher in VL when using 1 50 000 particles. In addition, this ratio increases almost linearly with N.

Figure 14 shows the different performances of VL_C and VL_X in terms of runtime depending on the number of time steps (n_s) the list is kept (C or X depending on the approach). Both VL_C and VL_X showed to be less efficient than CLL for low and high n_s values. However, there is an intermediate region ($3 \leqslant n_s \leqslant 15$), where both methods showed to be faster than CLL. In particular, the most efficient region is obtained for $n_s \sim 7$ time steps, where VL_X is about 4% faster than CLL and VL_C 3% faster. In addition, the method VL_X has shown to be faster than VL_C for any n_s. Obviously, the particular location of the maximum and the interval where the methods based on the Verlet list are more efficient depend on the case under study, although other calculations with different test cases obtained from the SPHysics website (http://www.sphysics.org) showed a similar behaviour. A similar figure can be obtained for different values of N.

The memory requirements of the different methods are shown in Figure 15. Thus, while CLL always requires the same amount of memory, the increase is almost parabolic with n_s in VL_C and VL_X. In the present case, $N = 31\,239$ particles, the memory allocated for CLL is on the order of 0.25 Mb and it can even be on the order of 25 and 40 Mb for VL_X and VL_C, respectively. In the

DOI: 10.1002/fld

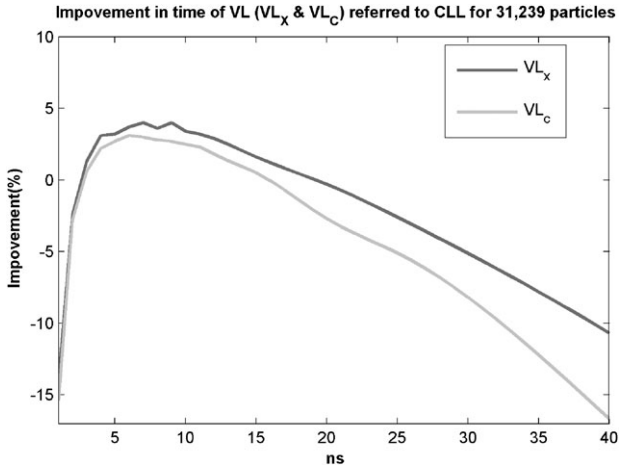

Figure 14. Improvement in time using VL_C and VL_X compared with CLL. All cases were calculated with $N = 31\,239$.

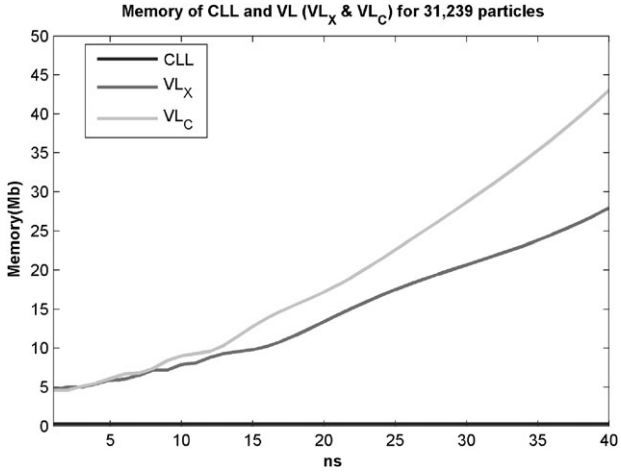

Figure 15. Allocated memory in CLL, VL_X and VL_C. All cases were calculated with $N = 31\,239$.

region where both methods are efficient ($n_s \sim 7$) the allocated memory is about 30 times higher than in CLL. In addition, it should be noted that VL_C has higher memory requirements than VL_X for any n_s. This is a direct consequence of the different values of v considered in both approaches (see Δh definition).

Figure 16 shows the comparison between VL_C and VL_X in terms of runtime improvement compared with CLL. This comparison was carried for $n_s = 7$ time steps which corresponds to the most efficient value for the methods based on a Verlet list as shown in Figure 14. For any number of particles, both methods have shown to be faster than CLL, with an improvement that tends to increase with N. In addition, VL_C was observed to be slower than VL_X for any N. The maximum improvement ($\sim 5.7\%$) was obtained for VL_X with 1 50 000 particles.

The observed improvement in velocity is moderate, especially when the memory requirements of the Verlet list are on the order of 30 times higher than for CLL in the most efficient region (see Figures 14 and 15). However, this result can be improved in those cases where the loop over particles should be carried out more than once per time step. This is the case, for example, of different improvements in classical SPH formulation as MLS filters [12, 36], kernel and kernel gradient corrections [37–40] or Riemann solvers [41]. In CLL the potential neighbours placed in

DOI: 10.1002/fld

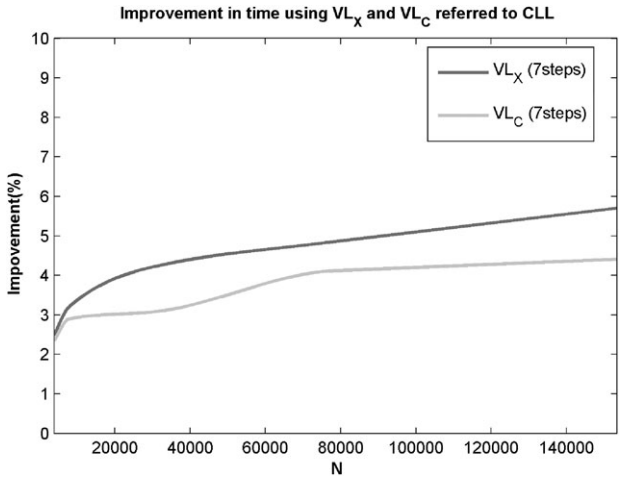

Figure 16. Comparison between VL_X and VL_C.

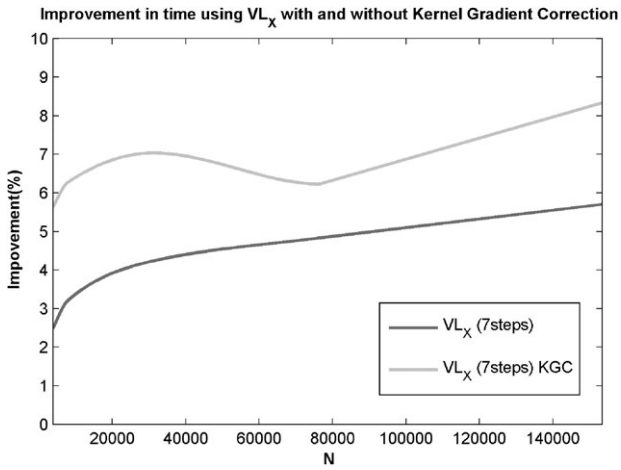

Figure 17. Comparison between VL_X with and without kernel gradient correction (KGC). The improvement is referred to CLL.

adjacent cells are checked several times every time step, while in the methods based on Verlet list the same list is loaded more than once but not recalculated several times every time step.

Figure 17 shows the comparison between VL_X in terms of runtime improvement compared with CLL. Two different approaches have been considered in this case. The light line corresponds to the SPHysics configuration described in Section 2.1. This line coincides with the light line as shown in Figure 16. The dark line corresponds to the same model with the kernel gradient correction described in [38]. In the case with kernel gradient correction, the velocity improvement was calculated comparing the runtime using VL_X with the runtime using CLL and the same gradient correction. Comparison was carried out assuming the most favourable case ($n_s = 7$, see Figure 14). Obviously, the improvement is higher in the corrected case, reaching a percentage higher than 8% for $N = 150\,000$ particles.

4. SUMMARY

This manuscript studies the use of neighbour lists in SPH models. In particular, calculations have been carried out using SPHysics. The performance of these neighbour lists has been previously

DOI: 10.1002/fld

commented in the literature [23] in such a way that realistic simulations cannot be performed by means of a brute force evaluation of summations in SPH method. The use of a list decreases the number of interactions per time step from N^2 to $N \cdot \ln(N)$. Keeping this in mind, different approaches have been considered to improve the performance of the method.

Traditionally, there are two methods to create a neighbour list. The CLL method creates a list linked to every cell which is used to calculate the possible interactions among particles placed in the same and adjacent cells. The VL method creates a list linked to every particle. The first step in the creation of a list is similar in both methods since particles should be initially organized in cells. Thus, the gridding procedure plays a key role in both approaches. Four gridding methods have been proposed, namely: (A) Sliding dynamic vector; (B) Preconditioned dynamic vector; (C) Static matrix method and (D) Linked list. In the first approach, the best method was D since it provides the fastest runtime with a moderate use of memory. Algorithm D uses 1.4 MB of memory simulating 1 50 000 particles. This algorithm is 20% faster than algorithm B, 24% than C and 75% faster than algorithm A. Thus, algorithm B was the second one. Algorithm A had to be discarded since it is the slowest procedure and C (37.9 MB) was also discarded due to the huge requirements in terms of memory (37.9 MB for 1 50 000 particles).

The performance of the methods can be improved by reordering the particles in such a way that particles from adjacent cells are close in memory, which gives rise to a more regular access pattern. This improvement was added to the four methods mentioned above. Reordering was observed to improve the runtime in more than 20% in most of the gridding algorithms. Once again, A and C had to be discarded when comparing the different methods due to the reasons mentioned in the last paragraph. B and D showed a similar performance in runtime although B was more efficient in terms of allocated memory. Due to this fact and to the ease to code the method, algorithm B was adopted to carry out the comparison between CLL and VL.

In general, VL needs much more memory than CLL, which is the main drawback of the method. In terms of runtime, the main advantage of VL is the possibility of keeping the same list during several consecutive time steps. The improved version (VL_X) of the classical Verlet list showed to be dependent on the number of steps, n_s, that the list is kept. For low and high values of n_s the CLL method was faster than VL_X. Only in an intermediate region VL_X was faster than CLL with a maximum improvement close to 6% for $n_s = 7$ time steps. This runtime improvement is rather moderate, especially when considering the memory requirements of the method compared with CLL. A better improvement in terms of runtime can be achieved when SPH has to loop over the particles more than once per time step, since the same list is kept in VL_X but the code is forced to a new search when using CLL. An improvement in runtime higher than 8% was obtained when using a SPHysics formulation with a kernel gradient correction, which implies a double loop every time step. Further improvement is expected when the number of loops per time step increases. To sum up, the choice of the neighbour list approach (CLL or VL_X) depends on the specific simulation under study. CLL is suggested for use when running a serial code since the number of particles is high and the memory requirements in VL_X are too expensive to be balanced by a runtime improvement on the order of 10%. However, in a parallel code each node deals with a low number of particles, which can constitute a well-suited case to implement VL_X since the burden is shared between the available nodes.

Although results are dependent on the case under study, test carried out with different initial configurations (see SPHysics test cases at http://www.sphysics.org) showed similar results both in 2D and 3D.

Finally, results are only valid for CPU calculations since different constraints must be taken into account in GPU (Graphics Processing Unit) codes. New research and *ad hoc* methods should be developed for GPU-based calculations.

ACKNOWLEDGEMENTS

The authors are grateful to Dr Ben Rogers for his helpful suggestions. This work was partially supported by Xunta de Galicia under the project PGIDIT06PXIB383285PR. The authors acknowledge the ESPHI

DOI: 10.1002/fld

(An European Smooth Particle Hydrodynamics Initiative) project supported by the Commission of the European Communities (Marie Curie Actions, contract number MTKI-CT-2006-042350).

REFERENCES

1. Lucy L. A numerical approach to the testing of fusion process. *Journal Astronomical* 1977; **82**:1013–1024.
2. Gingold RA, Monaghan JJ. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society* 1977; **181**:375–389.
3. Benz W, Asphaug E. Impact simulations with fracture. I. Methods and tests. *Icarius* 1994; **107**:98–116.
4. Benz W, Asphaug E. Simulations of brittle solids using smoothed particle hydrodynamics. *Computer Physics Communications* 1995; **87**:253–265.
5. Monaghan JJ. Simulating free surface flows with SPH. *Journal Computational Physics* 1994; **110**:399–406.
6. Monaghan JJ. Gravity currents and solitary waves. *Physica D* 1996; **98**:523–533.
7. Monaghan JJ, Kos A. Solitary waves on a Cretan beach. *Journal of Waterway*, *Port*, *Coastal*, *and Ocean Engineering* 1999; **125**(3):145–154.
8. Monaghan JJ, Kos A. Scott Russell's wave generator. *Physics of Fluids* 2000; **12**(3):622–630.
9. Shao SD. SPH simulation of solitary wave interaction with a curtain-type breakwater. *Journal of Hydraulic Research* 2005; **43**(4):366–375.
10. Shao SD, Lo EYM. Incompressible SPH method for simulating Newtonian and non-Newtonian flows with free surface. *Advanced Water Resources* 2003; **26**(7):287–800.
11. Gotoh H, Shao SD, Memeita T. SPH-LES model for numerical investigation of wave interaction with partially immersed breakwater. *Coastal Engineering Journal* 2004; **46**(1):39–63.
12. Colagrossi A, Landrini M. Numerical simulation of interfacial flows by smoothed particle hydrodynamics. *Journal of Computational Physics* 2003; **191**:448–475.
13. Gómez-Gesteira M, Dalrymple RA. Using a 3D SPH method for wave impact on a tall structure. *Journal of Waterway*, *Port*, *Coastal*, *and Ocean Engineering* 2004; **130**(2):63–69.
14. Gómez-Gesteira M, Cerqueiro D, Crespo AJC, Dalrymple RA. Green water overtopping analyzed with an SPH model. *Ocean Engineering* 2005; **32**(2):223–238.
15. Dalrymple RA, Rogers B. Numerical modeling of water waves with the SPH method. *Coastal Engineering* 2006; **53**:141–147.
16. Crespo AJC, Gómez-Gesteira M, Dalrymple RA. 3D SPH simulation of large waves mitigation with a dike. *Journal of Hydraulic Research* 2007; **45**(5):631–642.
17. Crespo AJC, Gómez-Gesteira M, Dalrymple RA. Modeling dam break behavior over a wet bed by a SPH technique. *Journal of Waterway*, *Port*, *Coastal*, *and Ocean Engineering* 2008; **134**(6):313–320.
18. Rogers BD, Dalrymple RA. SPH modeling of tsunami waves. *Advanced Numerical Models for Tsunami Waves and Runup*, vol. 10. Advances in Coastal and Ocean Engineering. World Scientific: Singapore, 2008.
19. Monaghan JJ. Smoothed particle hydrodynamics. *Reports on Progress in Physics* 2005; **68**(8):1703–1759.
20. Cleary PW, Prakash M, Ha J, Stokes N, Scott C. Smooth particle hydrodynamics: status and future potential. *Progress in Computational Fluid Dynamics* 2007; **7**:70–90.
21. Cleary PW, Prakash M. Discrete element modelling and smooth particle hydrodynamics: potential in the environmental sciences. *Philosophical Transactions of Royal Society* 2004; **362**:2003–2030.
22. Gomez-Gesteira M, Rogers BD, Dalrymple RA, Crespo AJC. State-of-the-art of classical SPH for free-surface flows. *Journal of Hydraulic Research* 2010; **48**:113–134.
23. Viccione G, Bovolin V, Carratelli EP. Defining and optimizing algorithms for neighbouring particle identification in SPH fluid simulations. *International Journal for Numerical Methods in Fluids* 2008; **58**:625–638.
24. Stellingwerf RF, Wingate CA. Impact Modelling with SPH. *Memorie della Societa Astronomia Italiana* 1994; **65**:1117.
25. Verlet L. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard–Jones molecules. *Physical Review* 1967; **159**:98–103.
26. Gómez-Gesteira M, Rogers BD, Dalrymple RA, Crespo AJC, Narayanaswamy M. User Guide for the SPHysics Code v2.0, 2010.
27. Monaghan JJ, Lattanzio JC. A refined method for astrophysical problems. *Astronomy and Astrophysics* 1985; **149**:135–143.
28. Monaghan JJ. SPH without a tensile instability. *Journal of Computational Physics* 2000; **159**:290–311.
29. Monaghan JJ. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics* 1992; **30**: 543–574.
30. Monaghan JJ, Cas RAF, Kos AM, Hallworth M. Gravity currents descending a ramp in a stratified tank. *Journal of Fluid Mechanics* 1999; **379**:39–70.
31. Batchelor GK. *Introduction to Fluid Dynamics*. Cambridge University Press: U.K., 1974.
32. Monaghan JJ. On the problem of penetration in particle methods. *Journal of Computational Physics* 1989; **82**:1–15.
33. Crespo AJC, Gómez-Gesteira M, Dalrymple RA. Boundary conditions generated by dynamic particles in SPH methods. *CMC*: *Computers*, *Materials*, *and Continua* 2007; **5**(3):173–184.

DOI: 10.1002/fld

34. Chen S, Johnson DB, Raad PE, Fadda D. The surface marker and microcell method. *International Journal for Numerical Methods in Fluids* 1997; **25**:749–778.
35. Morton GM. A computer oriented geodetic data base; and a new technique in file sequencing. *Technical Report*, Ottawa, Canada, IBM Ltd., 1966.
36. Dilts G. Moving-least-square-particle hydrodynamics—I. Consistency and stability. *International Journal for Numerical Methods in Engineering* 1999; **44**:1115–1155.
37. Belytschko T, Krongauz Y, Dolbow J, Gerlach C. On the completeness of meshfree particle methods. *International Journal for Numerical Methods in Engineering* 1998; **43**:785–819.
38. Bonet J, Lok T-SL. Variational and momentum preservation aspects of smoothed particle hydrodynamic formulations. *Computer Methods in Applied Mechanics and Engineering* 1999; **180**:97–115.
39. Vila JP. On particle weighted methods and SPH. *Mathematical Models and Methods in Applied Sciences* 1999; **9**:161–210.
40. Chen JK, Beraun JE. A generalized smoothed particle hydrodynamics method for nonlinear dynamic problems. *Computer Methods in Applied Mechanics and Engineering* 2000; **190**:225–239.
41. Marongiu JC, Leboeuf JC, Caro J, Parkinson E. Free surface flows simulations in Pelton turbines using an hybrid SPH–ALE method. *Journal of Hydraulic Research* 2010; **48**:Extra Issue 40–49.

DOI: 10.1002/fld