Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2018-0158

Vol. **26**, No. 1, pp. 57-86
July 2019

Improving Linked-Lists Using Tree Search Algorithms for Neighbor Finding in Variable-Resolution Smoothed Particle Hydrodynamics

Shahab Khorasanizade[1] and J. M. M. Sousa[1,*]

[1] *IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais, 1049-001 Lisboa, Portugal.*

Received 10 June 2018; Accepted (in revised version) 24 October 2018

Abstract. Improving linked-lists for neighbor finding with the use of tree search algorithms is proposed here, aiming to cope with highly non-uniform resolution simulations employing a meshless method. The new procedure, coined Quadtree Cells Grid, has been implemented in Smoothed Particle Hydrodynamics (SPH). The SPH scheme employed is adaptive, thus allowing for particle refinement in desired regions of the flow. Owing to the wide range of coexisting particle mass levels, standard linked-list neighbor search algorithms become ineffective. Hence, an alternative is found based on the use of hierarchical data structures, using quadtrees (in 2D problems). The present algorithm exploits the advantages of both linked-lists and quadtree methods with the goal of increasing computational efficiency, when dealing with highly non-uniform particle distributions. Test cases involving two distinct flow problems have demonstrated that the computational cost of the current adaptive neighbor finding algorithm scales linearly with the total number of particles, thus retrieving this characteristic of linked-lists in uniform grid search. Nevertheless, the memory usage increased as a result of the more complex data structure.

AMS subject classifications: 76M28, 68P10, 76D17

Key words: Smoothed particle hydrodynamics, linked-list, quadtree, neighbor finding, variable resolution.

1 Introduction

The particle-based method known as Smoothed Particle Hydrodynamics (SPH) [1] has seen a major increase in its range of applications along the past two decades [2,3]. Briefly, this method typically uses an isotropic smoothing function to calculate field values, and

*Corresponding author. *Email addresses:* `shahab.khorasanizade@tecnico.ulisboa.pt` (Sh. Khorasanizade), `msousa@tecnico.ulisboa.pt` (J. M. M. Sousa) particles carry media and flow data as these evolve according to Lagrangian governing equations. As a rule, the larger number of calculating particles required by more accurate simulations leads to increased computational cost. In the case of SPH, the number of particles demanded to achieve a desired quality of the solution may be reduced by using a variable-mass distribution of particles [4–6]. This strategy becomes especially important when considering engineering problems with truncated boundaries, where different spatial resolutions must be employed to deal with complex and/or moving geometries, such as in the cases of fluid flow through porous media [7] and fluid-structure interaction [8]. Whether uniformly distributed or not, and irrespectively of the criterion chosen for particle splitting or merging (dynamic or regional), a list of neighboring particles must be created for each field particle, to be used in the calculation of the various terms in the equations to be solved. The three main Nearest Neighboring Particle Search (NNPS) algorithms used for that purpose are as follows: all-pair (direct), linked-list (uniform grid) and tree (quadtree in 2D or octree in 3D) [1].

The all-pair search is the simplest but also the most inefficient NNPS algorithm. For a given particle i $(i=1,\cdots,N)$, where N is the total number of particles in the domain, the distance to all other j $(j=1,\cdots,N)$ particles is firstly inspected and only those falling inside a prescribed cut-off radius (the kernel compact support of SPH) will be considered in the next steps for interaction calculations. It is easy to show that the computational effort in this method is of order $\mathcal{O}(N^2)$, thus exposing the ineffectiveness of the algorithm.

In the uniform grid (linked-list) search, the domain is divided into an equally spaced mesh, which has the width of the kernel support in SPH. Subsequently, particles inside each mesh cell are found and, by arranging the particle array based on cell number, NNPS is performed. In this algorithm the search is only performed over the 3, 9 or 27 cells (in 1D, 2D or 3D, respectively) surrounding the target particle cell [1,9]. The computational effort is therefore reduced to the order $\mathcal{O}(N)$, which represents a significant improvement with respect to the method previously described. However, the same performance cannot be obtained using this method in SPH simulations containing particles with variable kernel smoothing lengths (i.e., non-uniform or adaptive SPH). As the level of adaptivity increases, the corresponding computational cost becomes exceedingly high. Hence, aiming to optimize this method for adaptive resolution simulations, an improved algorithm for linked-list neighbor search in single-CPU calculations has been implemented in the latest version of HAdynaSPH [4,10,11], as will be described in this paper.

Hierarchical data structures are particularly useful for the intended purpose because of their ability to focus on the interesting subsets of the data. Quadtree (in 2D and octree in 3D) is a hierarchical data structure based on the principle of recursive decomposition [12,13]. It has been used for the representation of data used for applications in image processing, computer graphics, geographic information systems, and robotics. In the field of Computational Fluid Dynamics (CFD), such structures are specially suitable to deal with adaptive resolution techniques [14–17]. The advantage in the use of the quadtree grids lies in the fact that those grids can be generated automatically, and managed dynamically based on certain criteria, without requiring complex algorithms [14–16, 18]. For example, quadtree grids have been exploited in Lattice-Boltzmann simulations to achieve maximum accuracy for a given CPU time, as well as to optimize memory usage in adaptive resolution algorithms [17, 19, 20]. In all these studies, the division of calculation cells is performed by setting a local criterion, either flow- or spatially-based.

Tree search methods have shown to work well in simulations with particles of variable size. The computational effort involved in this NNPS method is of order $\mathcal{O}(N\log N)$, which is nevertheless higher than that for uniform grid (linked-list) schemes. Xia and Liang [21] have tested quadtree algorithms aiming to speed up SPH calculations of the shallow water equations employing GPUs. In this investigation it has been shown that, compared to the commonly used uniform grid search method, quadtree neighbor searching may allow significant reductions in redundant computations when searching for neighbor particles. However, in some of the studied test cases no improvement was obtained. Awile et al. [22] proposed a NNPS algorithm for general adaptive particle methods, where particles are sorted based on smoothing length and spatial position. Their method showed advantages over conventional schemes (e.g., linked-list) for wide range of adaptive resolutions, as long as coupled with Verlet lists [23]. The computational effort involved without the use of Verlet lists has shown to be higher than that required by both tree and uniform grid search algorithms.

In the present work, a new variant of NNPS is introduced for use in adaptive SPH by retaining some of the features of both uniform grid (linked-list) and tree search algorithms. The proposed methodology reduces to the uniform grid algorithm in simulations with uniform particle resolution, whereas in variable particle mass calculations it allows for remarkable improvements in computational efficiency. These studies have been performed employing the Incompressible SPH (ISPH) method with dynamic adaptivity of HAdynaSPH [4, 6, 10]. The performance of NNPS when various levels of particle mass coexist in SPH is analyzed, comparing the computational cost required by the use of the uniform grid (linked-list) algorithm versus the Quadtree Cells Grid (QCG) method proposed here. This improved algorithm is aimed solely at reducing the computational effort of linked-lists, as it was specially designed to retrieve a cost of order $\mathcal{O}(N)$ in NNPS of adaptive SPH simulations on single-CPU calculations without changing the physical modeling of the problem. Hence, as long as the same spatial resolution is employed, no changes on the flow solution are observed, as shown in this study for incompressible flows past a circular cylinder and a plunging NACA0012 airfoil placed in a free stream. For discussions on stability, consistency, conservation and convergence characteristics [24] of the numerical scheme for the physical modeling employed in HAdynaSPH the reader is referred to previous work [4, 6, 25, 26]. All the results reported herein have been obtained on a desktop machine equipped with an Intel Core i7-2600K @ 3.4 GHz quad-core CPU. The HAdynaSPH package has been developed in `FORTRAN` for double precision calculations and compiled with Intel Fortran Composer XE v12.1 for Linux on a single core.

2 NNPS algorithms

In the case of CFD, the Lagrangian Navier-Stokes equations are solved for each SPH particle representing a parcel of fluid [1]. Such particles are thus moving entities, carrying fluid and flow information, which is obtained at each time instant from discretized volume integration summations over the particles inside their compact support. The radius of this subdomain depends on a weighting function, often referred to in SPH as the kernel function. As particles generally move independently of each other within the flow domain, their neighbor particles change from one time iteration to the other. Hence, an efficient NNPS algorithm is crucial in high-resolution SPH calculations. Some researchers [27,28] have suggested that the neighbor list may be kept unchanged for several time steps of the simulations by simply using a larger subdomain volume. However, this procedure also requires the allocation of larger arrays to store the information, and it leads to calculation errors in rapidly changing flows when variable resolution is employed.

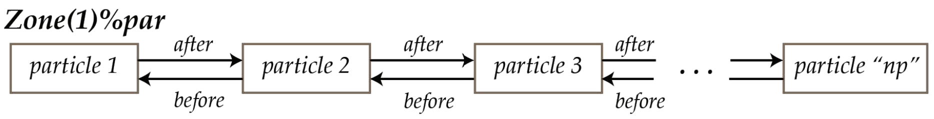
Each and every particle in the HAdynaSPH package is tagged as either Fluid (F), Buffer (O, for Open boundaries), Edge (E), Image (B, for solid Boundary particles) or Solid (S, in two-phase flows). These particles are parts of a linked-list data type, under their corresponding zone with F, O, E, B or S assignment, as depicted in Fig. 1. Each zonal and particle data structure carries physical and numerical entities needed during the simulations, such as mass, density, position. Linked-list information, namely the particle that comes *after* and *before* the current one, is also available there, as seen in Fig. 1. The underlying cells in the proposed NNPS algorithm have a linked-list of `POINTER`s to the particles contained therein. In the following subsections, standard `FORTRAN` syntax, wherever applicable, is used to define the variables.

2.1 Uniform grid (linked-list)

When using this search method, a coarse mesh with the cell size equal to the compact support radius is superimposed to the particle-filled domain. Each particle and grid cell are given unique index numbers, which do not change during the simulation. For each row of the particle index array, the cell index is saved and subsequently this array is sorted by cell number to optimize the neighbor finding search [21, 28]. This sorting is often addressed via Hashing, but it is not applied here due to the use of a different arrangement, as will be discussed below. In the HAdynaSPH package, equally spaced cells are ordered based on z, x and then y directions, as shown in Fig. 2a. This creates $N_{gr} = Grid_x \times Grid_y \times Grid_z$ cells in the domain, where the array *Grid* contains the number of cells in each of those directions. The procedure takes advantage of the interaction symmetry, i.e., if a particle i is found in the vicinity of particle j, vice-versa is also true. Bearing this in mind, each cell has only 5 neighboring cells in 2D problems, rather than the classical 9 neighbors (in 3D problems, 14 neighbors would be considered instead of 27). Traditional and presently used neighboring arrangements are illustrated in Fig. 2b-c,

```
TYPE(zones), DIMENSION(:), ALLOCATABLE :: Zone

TYPE zones
    INTEGER :: index
    INTEGER :: np  !! Number of contained particles
    CHARACTER :: name*12
    CHARACTER :: t*1  !! type of the zone: F, O, E, B, S
    LOGICAL :: compt  !! whether computation should be made on this zone or not
    REAL :: rho  !! density of particles in the current zone
    REAL :: mass  !! characteristic mass or mass of each particles in uniform simulations
    REAL :: hsml !! characteristic smoothing length
    REAL :: nu  !! viscosity
    REAL :: vol !! volume of the zone
    TYPE(particle), POINTER :: par => NULL()
    TYPE(particle), POINTER :: par_final  => NULL()  !! pointer to the last particle
END TYPE zones
```



```
TYPE particle
    INTEGER :: id
    REAL, DIMENSION(dim) :: pos  !! Position
    REAL, DIMENSION(dim) :: vx  !! Velocity
    REAL :: p  !! Pressure
    REAL, DIMENSION(:), ALLOCATABLE :: nu  !! Viscosity in case of turbulence or non-Newtonian
    REAL :: m !! Mass of the particle
    REAL  :: hsml  !! smoothing length
    INTEGER :: mass_level  !! mass level of the particle
    REAL, DIMENSION(:,:), ALLOCATABLE :: sij    !! Strain Tensor
    LOGICAL :: del  !! Deletion status, TRUE in case needed to be removed from the simulation
    TYPE(particle), POINTER :: parent => NULL()  !! Parent of a particle in case of "B" particle
    TYPE(particle), POINTER :: before => NULL()
    TYPE(particle), POINTER :: after => NULL()
END TYPE particle
```

Figure 1: Illustration of zone and particle data structure for a linked-list in a generic $Zone(1)$.

by choosing cell #16 as an example.

The neighbor list of cells is created following Algorithm 1. This is carried out only once, in the beginning of the simulation, as the number of cells and arrangement of the grid do not change during the calculation. Based on the ordering directions defined earlier, a unique spatial record of each cell with index g is saved in dim. The various properties of particles and cells are hereinafter addressed using the standard FORTRAN

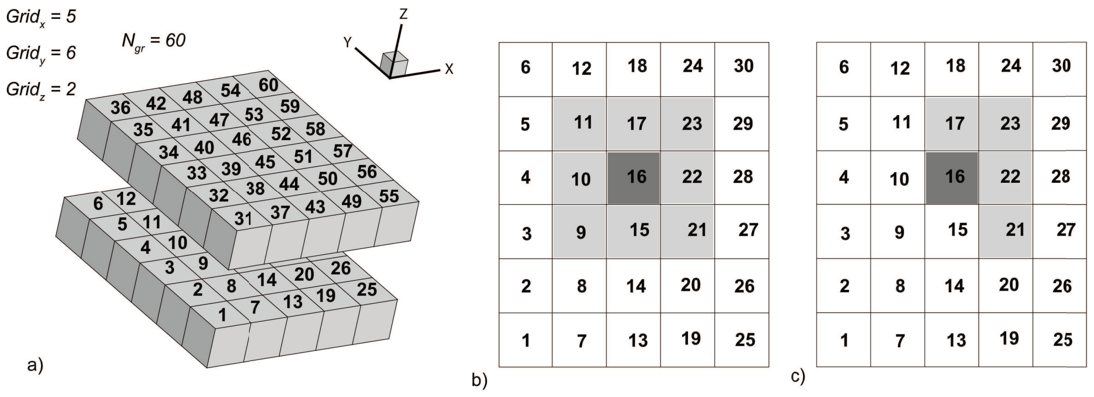


Figure 2: Grid and cell neighbor arrangement: a) general placement of cells in a 3D domain; b) traditional arrangement of neighboring cells [21,28]; c) neighboring cells used in HAdynaSPH [4,10].

convention symbol % for derived data types. For example, for $g=16$ in the example of Fig. 2b-c, we have $g\%dim=(1,3,4)$.

Algorithm 1. Grid cell neighbor finding.

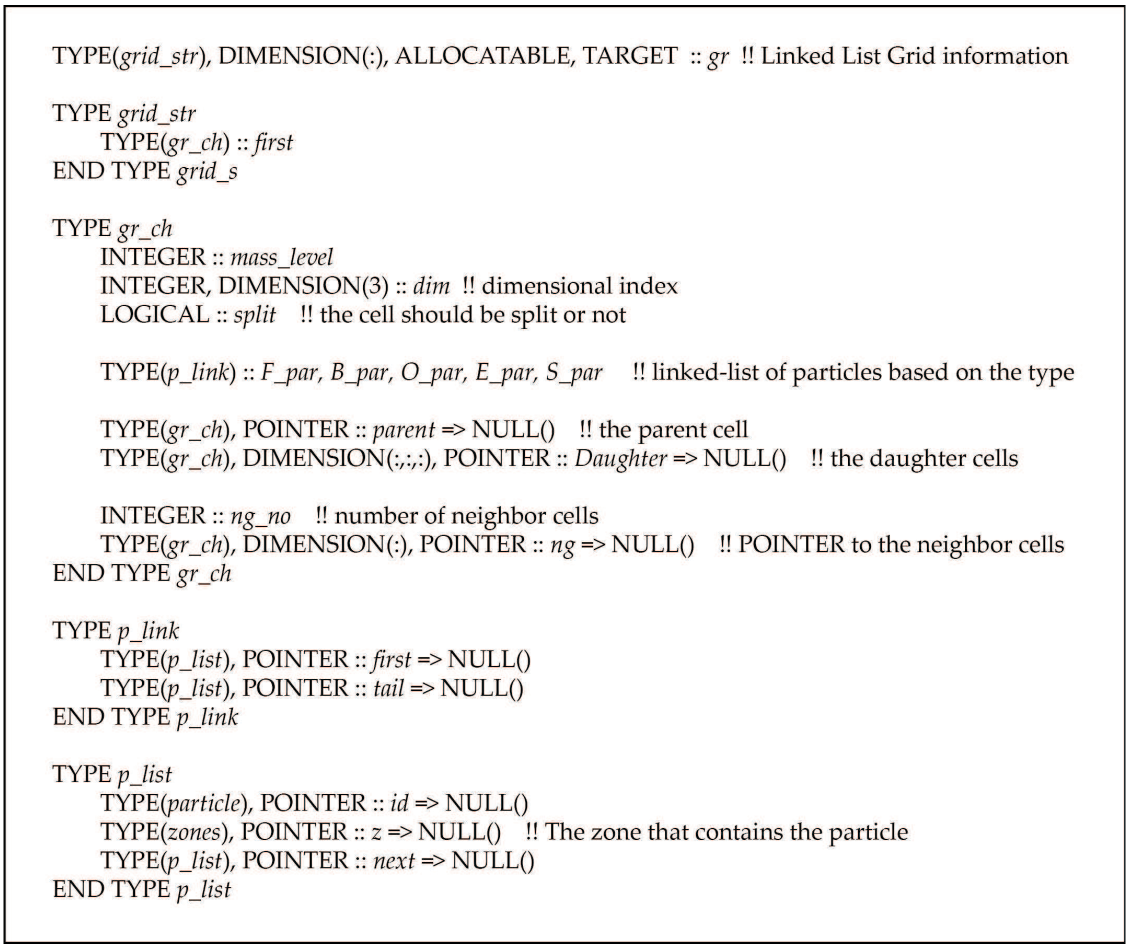
```
Do $g=1,N_{gr}$
    Do $g1=g,N_{gr}$
        $dg=g\%dim-g1\%dim$
        If all the values in $dg$ are either $-1$, 0 or 1
            **Add $g1$ to the neighbor list of $g$**
```

Subsequently, the position vector of each particle is divided by the size of the cell in each direction. Eq. (2.1) is used to calculate the index of the containing cell, as follows:

$$\begin{cases} dg=particle_{pos}/cell_{size}, \\ g=Grid_y\{dg(1)-1+[dg(3)-1]\,Grid_x\}+dg(2), \end{cases} \tag{2.1}$$

where $particle_{pos}$ and $cell_{size}$ denote arrays containing the spatial position of the particle and the cell dimensions, respectively. The target particles are added at the end of the linked-list created for each cell and specific type of particle (F, O, E, B or S, as referenced earlier). This data structure is shown in Fig. 3. Each linked-list contains connections to the first, next and the final entry of the list, which are used to traverse it. For each structure, a `NULL` first record signifies the inexistence of that particle type in the specific cell. These chain-like data types are created to simplify the process of adding and removing entities from the linked-list [29].

Solid wall boundary treatments are carried out in HAdynaSPH employing the SBA algorithm [10]. The procedure initially involves finding the closest E particle to each of the F/O particles. In general, the ratio of the number of F/O particles to the number



```
TYPE(grid_str), DIMENSION(:), ALLOCATABLE, TARGET  :: gr  !! Linked List Grid information

TYPE grid_str
      TYPE(gr_ch) :: first
END TYPE grid_s

TYPE gr_ch
      INTEGER :: mass_level
      INTEGER, DIMENSION(3) :: dim  !! dimensional index
      LOGICAL :: split   !! the cell should be split or not

      TYPE(p_link) :: F_par, B_par, O_par, E_par, S_par    !! linked-list of particles based on the type

      TYPE(gr_ch), POINTER :: parent => NULL()   !! the parent cell
      TYPE(gr_ch), DIMENSION(:,:,:), POINTER :: Daughter => NULL()   !! the daughter cells

      INTEGER :: ng_no   !! number of neighbor cells
      TYPE(gr_ch), DIMENSION(:), POINTER :: ng => NULL()   !! POINTER to the neighbor cells
END TYPE gr_ch

TYPE p_link
      TYPE(p_list), POINTER :: first => NULL()
      TYPE(p_list), POINTER :: tail => NULL()
END TYPE p_link

TYPE p_list
      TYPE(particle), POINTER :: id => NULL()
      TYPE(zones), POINTER :: z => NULL()   !! The zone that contains the particle
      TYPE(p_list), POINTER :: next => NULL()
END TYPE p_list
```

Figure 3: Grid data structure created in HAdynaSPH.

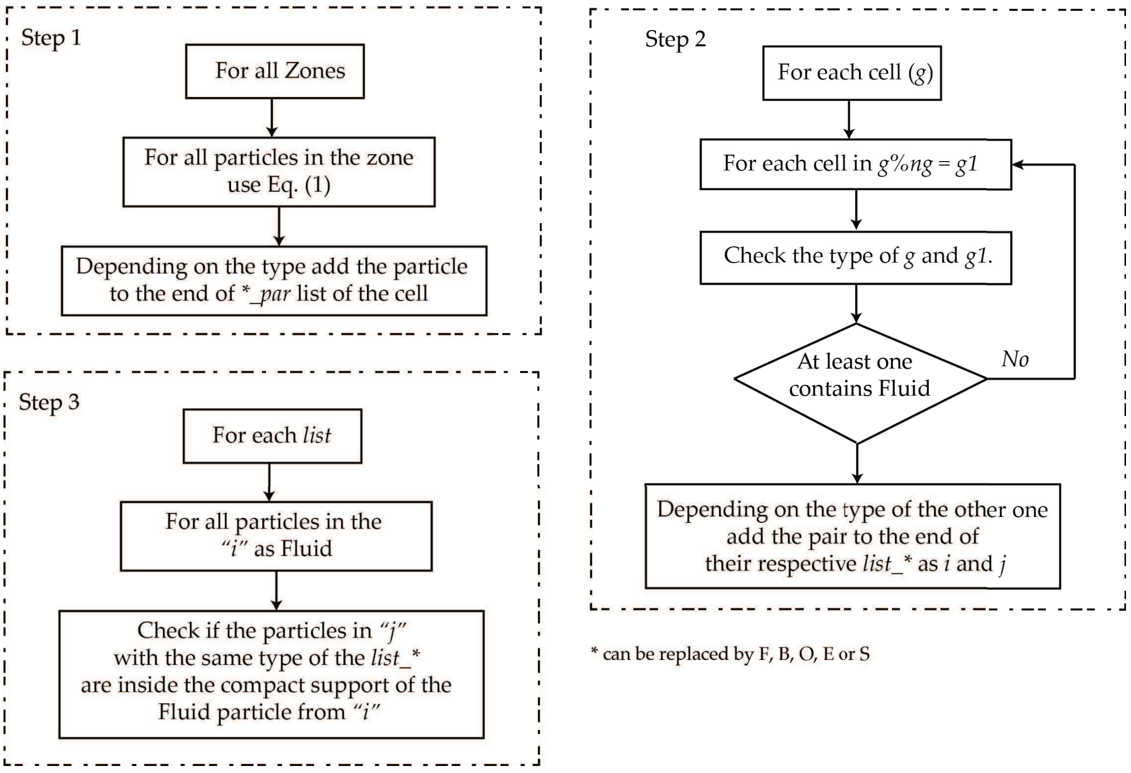
of E particles is high, thus finding the closest edge particle becomes computationally expensive. This search has been optimized by placing F/O and E particles inside the cells using Eq. (2.1). Once the grid information is compiled by filling the `POINTER`s of $\%f\_par$, $\%O\_par$ and $\%E\_par$, a linked-list of cells is constructed by pointing to grid nodes encircling fluid particles and a neighboring cell with Edge attribute. Considering Fig. 2, if at least one F/O particle exists in cell #23 and one E particle exists in cell #29, the data structure depicted in Fig. 4 produces an entry to the Edge finding list as follows: $list\_E\%i$ => #23, $list\_E\%j$ => #29, $list\_E\%next$ => `NULL()`. By traversing this list one may find the Fluid/Open boundary particles in vicinity of an edge. Following these initial steps, B particles are created [10], and these are added to their corresponding cells using Eq. (2.1) and the scheme described earlier.

In SPH calculations, particle interactions must be found for F-F, F-O and F-B (and also for F-S in the case of two-way coupling in multiphase flow). The underlying optimization

```
TYPE(gr_first) :: list_F      !! List of fluid seeing grid index
TYPE(gr_first) :: list_O      !! List of buffer seeing grid index
TYPE(gr_first) :: list_E    !! List of edge seeing grid index
TYPE(gr_first) :: list_S      !! List of solid seeing grid index
TYPE(gr_first) :: list_B   !! List of Image seeing grid index

TYPE gr_first
    TYPE(grid_list), POINTER :: first => NULL()
    TYPE(grid_list), POINTER :: tail => NULL()
END TYPE gr_first

TYPE grid_list
    TYPE(gr_ch), POINTER :: i => NULL()
    TYPE(gr_ch), POINTER :: j => NULL()
    TYPE(grid_list), POINTER :: next => NULL()
END TYPE grid_list
```



Step 1
For all Zones
For all particles in the zone use Eq. (1)
Depending on the type add the particle to the end of *_par list of the cell
Step 2
For each cell (g)
For each cell in g%ng = g1
Check the type of g and g1.
At least one contains Fluid
No
Depending on the type of the other one add the pair to the end of their respective list_* as i and j
Step 3
For each list
For all particles in the "i" as Fluid
Check if the particles in "j" with the same type of the list_* are inside the compact support of the Fluid particle from "i"
* can be replaced by F, B, O, E or S

Figure 4: Cell type listing data structure and the interaction finding steps in the linked-list algorithm.

concept is similar to that of edge particle finding. Also based on the interaction type, a linked-list of F-containing cells, which includes a POINTER to the neighbor cell encircling any of the aforementioned types of particles, is made. This linked-list is looped over the cells, with an inner loop over the particles contained in each cell, thus inspecting the particles in the pointed cell to find all the interactions. The specific interaction cell list and overall particle interaction finding steps are shown in Fig. 4. Step 1 consists in placing each particle at the end of the respective linked-list type in the containing cell. This is followed by Step 2, where the interaction finding is firstly optimized by sorting the cells and their neighbors based on the particles they contain. Particle interaction are found and saved at each iteration by looping inside the *list* linked-lists, as described in Step 3. Traversing the linked-list throughout HAdynaSPH is done using standard `FORTRAN` implementations [29].

Nevertheless, even after all these improvements, this method proved not to be adequate for variable-resolution SPH simulations due to the excessive computational cost of the NNPS. Hence, a new procedure retaining some of the features of uniform grid (linked-list), but making extensive use of tree search algorithms, is proposed next.

2.2 Quadtree Cells Grid (QCG)

Uniform grid algorithms are known to be a suboptimal strategy for handling the requirements of variable resolution, and their application to SPH does not make an exception. This is illustrated here based on Fig. 5, where the particle arrangement around a NACA0012 airfoil is shown. For example, using a quintic kernel and setting the ratio of the kernel smoothing length to the initial particle spacing to the (usual) value of 1.3, one may expect approximately 16 uniformly-distributed particles (of $mass_{level}=1$) in each cell (Fig. 5a). However, as each of the original particles is split into 4 smaller ones in a regional refinement procedure [4], one would find approximately 64 particles of the next mass level ($mass_{level}=2$) in the same cell. Further, if these particles themselves split into a higher level following the same scheme, approximately 256 calculating nodes (of $mass_{level}=3$) would result in that region, and so on. This splitting pattern is shown in Fig. 5b, where the largest axis of the airfoil chord C is used as length scale. As the number of particle mass levels increases, the uniform grid (linked-list) NNPS diverges from being optimal.

One of the possible ways to optimize the NNPS in variable-resolution calculations is via tree-search algorithms [1], in particular using quadtree (2D) or octree (3D) data structures. In these, particles are firstly sorted in a hierarchical tree. This tree is constructed with the whole computational domain forming its main root. The root is divided into 4 daughter cells in 2D problems (or by 8, in 3D problems), which may contain more than one particle. This division process is carried out, and repeated for all the cells encompassing more than one particle until each one contains no more than one particle. In other words, the branch splitting is performed until the leaves on the tree correspond to individual particles. The result of such process for a typical particle field is depicted in Fig. 6. Unlike the uniform grid searching method, particles in tree methods are sorted by space-filling curves or z-indexing. This way, particles spatially near each other are also placed close to each other in memory [21]. After building a tree, such as the one shown in Fig. 6b, the neighbors are found by traversing the tree through desired paths. These paths

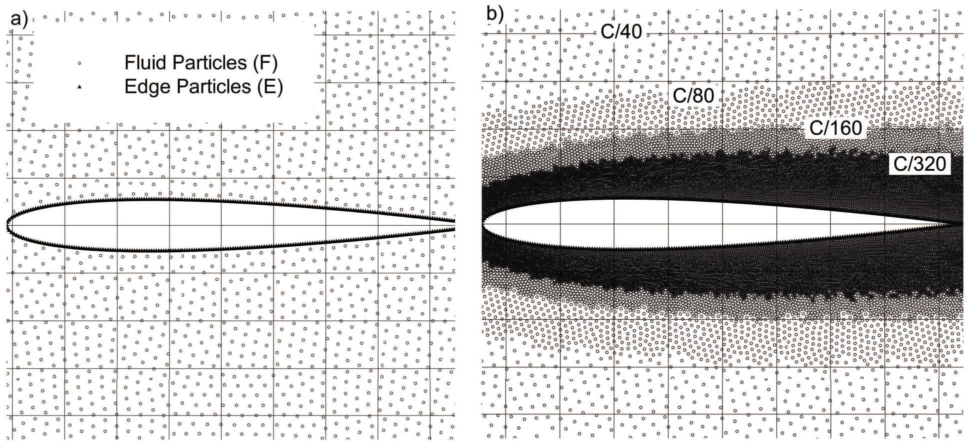


Figure 5: Particles in cell arrangement for: a) uniform resolution $C/40$; b) variable resolution with regional refinement and coarsest resolution $C/40$. The number of coexisting $mass_{level}$ shown in (b) is 4.

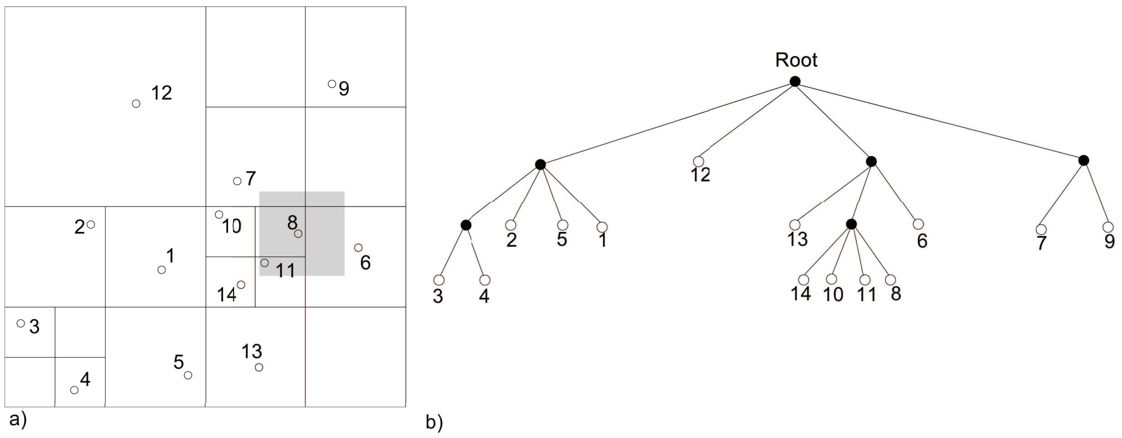


Figure 6: Original quadtree NNPS: a) particle arrangement and grid creation; b) tree structure.

are chosen by verifying if the cells are overlapping with a square enclosing the target particle (shaded area around particle #8 in Fig. 6a). The aforementioned condition must be met until the search reaches the leaf of the tree containing a single particle, so that it may be considered as a neighbor of the target particle. For additional details on this search algorithm the readers are referred to the work of Xia and Liang [21]. One should emphasize that the computational effort of this method is of order $\mathcal{O}(N\log N)$, and even in the implementation of Awile et al. [22] it is of order $\mathcal{O}(max_{level}N\log N)$, where max_{level} denotes the maximum number of coexisting $mass_{level}$. Hence, retrieving the computational cost of linked-list in uniform grids, i.e., $\mathcal{O}(N)$, is increasingly beneficial as the number of particles (or calculation nodes) grows.

In the variable-resolution method implemented in the HAdynaSPH package, a quad-tree is created for the foregoing grid instead of building a tree based on the particles. This has been done with the specific objective of achieving a computational effort of order $\mathcal{O}(N)$ in the NNPS. Hence, one should not misunderstand the proposed procedure with a tree search method, as the underlying concepts of the proposed algorithm are still linked-list alike. During the process of particle placement in the cells, the mass levels of particles and the cells are compared. If the particle $mass_{level}$ is larger than that of the target cell, that cell is tagged for splitting and it will be added to the end of *split_grid_list* to be divided in the next flow calculation iteration. This linked-list contains `POINTER`s to those cells that need to be split at the end of current time step. The physical domain, which encompasses also the boundaries and extended regions in case of free-surface flows, is firstly divided into an initial uniform grid (as shown in Fig. 5) with the value of $mass_{level}=1$ assigned to each one of its cells. After a prescribed number of iterations (every 10 time steps in the present study), the code loops over *split_grid_list* to split the tagged cells (g in Algorithm 2). In the 2D problem illustrated in Algorithm 2, each parent cell is divided into four daughter cells with their unique spatial index (%*dim*) for the new $mass_{level}$. As these smaller cells contain smaller particles, their mass level is increased by one unity and their splitting condition is set to `FALSE`.

Algorithm 2. Grid cell splitting using QCG. For the sake of simplicity only a 2D problem is illustrated, but the same concept applies to 3D problems as well. Note also that the standard `FORTRAN` symbol => indicates that a `POINTER` variable is used to store the memory location of another variable.

```
Do $g = split\_grid\_list$ (1 to $end$)
    Allocate $g\%Daughter(2,2)$ (a two-by-two cell array for the number of daughter cells)
    Do $i = 1,2$
        Do $j = 1,2$
            $g1$ => $g\%Daughter(i,j)$
            $g1\%dim(1) = (g\%dim(1)1) \times 2 + i$
            $g1\%dim(2) = (g\%dim(2)1) \times 2 + j$
            $g1\%split =$ FALSE
            $g1\%parent$ => $g$
            $g1\%mass_{level} = g\%mass_{level} + 1$
```

After the completion of Algorithm 2, the neighbor list of the new (smaller) daughter cells must be assigned. It must be noted that contrary to the uniform grid algorithm described in the previous subsection, and for reasons that will be discussed later, each cell in the initial uniform grid has again 9 neighbor cells (Fig. 2b) instead of 5. The steps on how the daughter cell neighbor list is created and updated are illustrated in Algorithm 3. The search is limited to the neighbors of its parent cell, as the daughters neighbor cells cannot lay beyond those cells. This algorithm has been assembled in two main sections: Part 1 assigns the neighbor list for the recently created daughter cells ($g1$); Part 2 updates the neighbor list of daughter cells $(g3)$ holding a $mass_{level}$ equal to that of $g1$. These cells have been split in previous time steps and are children of the cells in the neighborhood of cell g (the parent of $g1$).

Algorithm 3. Daughter cell neighbor list creation and update using QCG.

```
Do $g = split\_grid\_list$ (1 to $end$)                                   ↓ Part 1
    Do $i = 1,2$
        Do $j = 1,2$
            $g1 => g\%Daughter(i,j)$
            Do $g2 => g\%ng$ (traversing the linked-list)
                If $g\%mass_{level} = g2\%mass_{level}$
                    If $g2$ has Daughters
                        Do $m = 1,2$                                          Part 1.1
                          Do $n = 1,2$
                            $g3 => g2\%Daughter(m,n)$
                            $dg = g1\%dim - g3\%dim$
                            If all the values in $dg$ are either -1, 0 or 1
                              Add $g3$ to the end of neighbor list of $g1(g1\%ng)$
                    Else
                        Status = TRUE                                         Part 1.2
                        Do $m = 1,2$
                          If Status is TRUE
                            Do $n = 1,2$
                              If Status is TRUE
                                $Fake\_dim(1) = [g2\%dim(1) - 1] \times 2 + m$
                                $Fake\_dim(2) = [g2\%dim(2) - 1] \times 2 + n$
                                $dg = g1\%dim - Fake\_dim$
                                If all the values in $dg$ are either $-1$, 0 or 1
                                  Add $g2$ to the end of neighbor list of $g1(g1\%ng)$
                                  Status = FALSE
Do $g2 => g\%ng$ (traversing the linked-list)                             ↓ Part 2
    If $g2$ has Daughters AND not being split
        Do $m = 1,2$
            Do $n = 1,2$
                $g3 => g2\%Daughter(m,n)$
                $k = 0$
                Do $g4 => g3\%ng$ (traversing the linked-list)
                    If $g4 = g$
                        Do $i = 1,2$
                            Do $j = 1,2$
                                $g1 => g\%Daughter(i,j)$
            $dg = g1\%dim – g3\%dim$
            If all the values in $dg$ are either $-1$, 0 or 1
                **Add** $g1$ **to the end of neighbor list of** $g3(g3\%ng)$
                $k = k+1$
    If $k > 1$
        **Remove** $g$ **from the neighbor list of** $g3(g3\%ng)$
```

Fig. 7a shows a schematic of the grid where the split cells are marked using dashed lines. In turn, Fig. 7b illustrates the tree structures for the split cells in Fig. 7a. As mentioned in Section 2.1, the array *Grid* contains the number of cells in each of the x, y or z directions. In the proposed QCG algorithm, and contrasting to its uniform grid counterpart, the *Grid* array is defined per each $mass_{level}$, thus producing the cell indices shown in Fig. 7b as explained next. This $Grid^{mass\_level}$ contains the maximum possible of cells that may exist for a specified $mass_{level}$. Each $mass_{level}$ has its corresponding $Grid^{mass\_level}$ with values given by the dimension of its previous level in each direction multiplied by two, as each parent cell breaks into a 2-by-2 grid in 2D problems. Taking the example in Fig. 7, $Grid^{(1)}$ (e.g., for $mass_{level} = 1$) takes the values (4,4), whereas $Grid^{(2)}$ takes the values (8,8), despite the fact that at the instant shown in the figure not all of those smaller cells exist. In a general adaptive simulation one might never reach that number of cells for a known $mass_{level}$, and these values only denote the upper limits. After making available the array $Grid^{mass\_level}$, Eq. (2.2) is used to determine the cell index, as follows:

$$INDEX = Grid_y^{mass\_level}\left\{g\%dim(1) - 1 + \left[g\%dim(3) - 1\right] Grid_x^{mass\_level}\right\} + g\%dim(2), \quad (2.2)$$

where $INDEX$ is the cell number shown in Fig. 7b. Taking again the 2D problem illustrated in Fig. 7, for $mass_{level} = 1$ and $g\%dim = (3,2)$ the value of $INDEX$ is 10, whereas for $mass_{level} = 3$ and $g\%dim = (7,6)$ it takes the value of 102. Contrary to tree search schemes previously used in SPH [21], where only one particle is placed per leaf of the tree, the cells in this improved linked-list method (the present QCG algorithm) contain more than one particle, as the branches are in fact constructed to reduce the search area for the pair-finding. It must be noted that the arrangement in Fig. 7 is independent of the particle arrangement itself, as far as small particles exist in smaller cells. In addition, the spatial index of each cell starts from 1 for each $mass_{level}$ and it is indeed unique within a specific mass level, but the same cell index may eventually exist within the whole domain for a different $mass_{level}$. Hence, each cell is identified by both the $mass_{level}$ and its unique spatial index within that $mass_{level}$. Hereinafter, a particular cell g will be referred to as $\#INDEX^{mass\_level}$.

As mentioned earlier, the first part of Algorithm 3 is assigning the neighbor list of daughter cells $(g1)$ created from any cell g. Initially, the neighbor cells of g are looped over to check whether the $mass_{level}$ of such cells is equal to that of g. This part of the algorithm was designed to ensure that the cells find their neighbors characterized by a size equal or one level above that of themselves. Upon passing this check, there are two sub-sections where the neighbor list of $g1$ is created. If $g2$ (i.e., the neighbor of g)

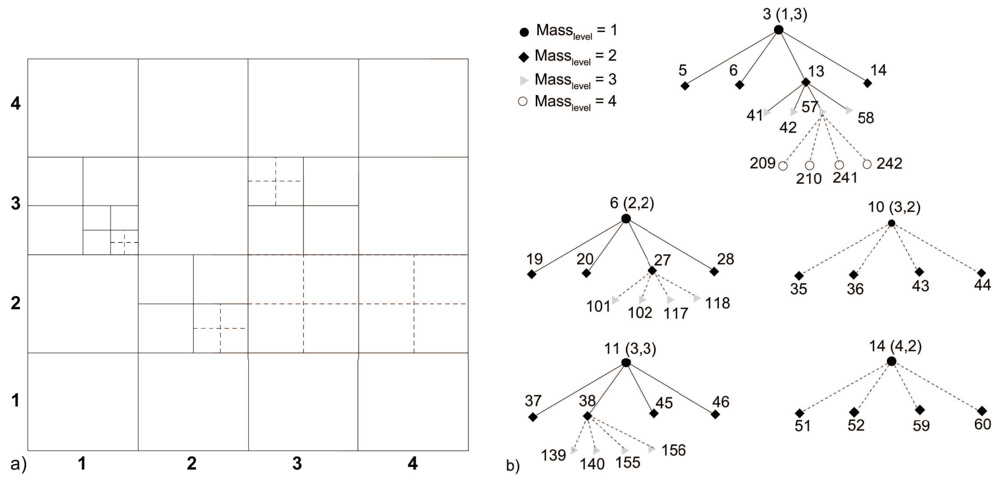


Figure 7: Example of application of the QCG algorithm: a) grid structure with dashed lines representing recently split cells; b) tree representation of the grid, where only the cells that have daughters are shown.

has daughters, it must be verified whether such cells are the neighbors of $g1$, following Part 1.1 of the algorithm. Once more as illustrated in Fig. 7, for $g1=\#44^2$ under $g=\#10^1$, Algorithm 3 (Part 1.1) goes through $g2$=$\#11^1$, and assigns $g3$=$\#37^2$ and $\#45^2$ as neighbors of $g1=\#44^2$. Also, for $g2=\#14^1$, daughter cells of $g3=\#51^2$ and $\#52^2$ are considered as neighbors of $g1=\#44^2$. On the other hand, Part 1.2 of the algorithm checks for the neighbors of g, namely $g2$, that do not have any daughters. Let us assume that $g2$ has been split and each of these imaginary daughters has dimensions calculated as *Fake_dim*. If any of these fake daughters could be considered as neighbors of $g1$, then $g2$ would be added to the neighbor list of $g1$. For instance, for $g1=\#44^2$ in Fig. 4, $g2=\#15^1$ with $g\%dim=(4,3)$ would be in the neighbor list.

The neighbors of recently split cells are assigned using, again, Part 1 of the algorithm. However, the neighbor list of the daughter cells that have been split in the previous iterations of the flow calculations needs to be updated for the new daughters to be considered. This is carried out using Part 2 of Algorithm 3, and the procedure is explained for the previously split cell $g2=\#11^1$, as sketched in solid lines in Fig. 7a. Prior to this, some of the daughters of $g2$, for example $g3=\#45^2$ with $m=2$ and $n=1$ in Algorithm 3, would have g=$\#10^1$ in their neighbor list ($g4$). However, as g had been split in the current iteration of the flow calculations, the neighbor list of $g3$ must be updated to contain the $g\%Daughter$ instead of g. By looping over the neighbors of g that have not been split at the current time-step ($g2$), one finds the daughters ($g3$) of such cells that have g as their neighbor. Once those cells have been found, it must be checked whether $g\%Daughter$ are in the vicinity of $g3$. Then, these new cells are added and g is removed from the neighbor list of $g3$. Hence, following the steps above in Part 2 of Algorithm 3, $g1=\#36^2$ and $\#44^2$ (i.e. the daughters of $g=\#10^1$) would be added, while their parent $g=\#10^1$ would be removed from the list of $g3=\#45^2$.

It was also mentioned earlier that cells in linked-list are split after being tagged during the particle placement in the cells. However, the particle arrangement becomes much more complicated once dealing with variable resolution and, therefore, particle placement in the cells could not be done by simply using Eq. (2.1). Hence, in the present method, the particles are placed in the cells following Algorithm 4.

Algorithm 4. Particle placement in cells using QCG. Here only F particles are considered as an example. The same algorithm is applied to other types of particles. The fluid containing property in the present example is saved in f_self for each cell.

```
Do $F = Zone(1)\%par$ (traversing the fluid particles linked-list)
    Use Eq. (2.1) to find the cell index for $mass_{level} = 1, g$
    $Aux\_g = g$
    Do while $Aux\_g$ has Daughters
        $Aux\_g\%f\_self$ = TRUE
        EXIT this loop If $Aux\_g\%mass_{level} = F\%mass_{level}$
        $dg = F_{pos}/cell_{size}$ (corresponding to the current mass level)
        $daughter_d = dg\,(Aux_g 1) \times 2$
        $Aux_g$ => $Aux_g\%daughter_d$
    **Add** $F$ **to** $Aux_g$ **particle list** $(Aux_g\%F\_par)$
    If $Aux\_g\%mass_{level} \neq F\%mass_{level}$, $Aux_g\%split$ = TRUE **and add to the end of** $split\_grid\_list$
```

In Section 2.1 it was explained that linked lists of cells are constructed for each and every type of SPH particle interactions. Subsequently, these structures are used to find all the interactions. Based on Algorithm 4, particles are placed in the cells of the same $mass_{level}$. So, if by traversing down the quadtree of the cells one reaches a cell that has a lower $mass_{level}$ than the target particle, this cell is tagged to be split at the end of this iteration. Here, interactions between particles of various $mass_{level}$ need also to be considered, as even a cell without daughters may have particles with a higher $mass_{level}$. Generally, the particle placed in a cell goes through the NNPS for all the particles placed in the neighbor cell and all its subsequent daughters. Using again Fig. 7, cell #10^1 sees cell #6^1 and both of them are characterized by $mass_{level} = 1$. Note that the cell #6^2, as a Daughter of cell #3^1, is different from the cell used in the current example and is not seen by cell #10^1. For this situation, the large particles in cell #10^1 must check all the particles placed in cell #6^1 and the smaller cells created from it, i.e., dead-end cells such as #20^2, #28^2 and smaller ones such as #101^3, #118^3, among others. The same concept applies also when dealing with cell #35^2. This cell is a neighbor of cell #27^2, which has the same size, and apart from checking for the particles of this $mass_{level}$, the algorithm must check for the particles placed in higher $mass_{level}$ cells created from cell #27^2 as well. Another example can be given using cell #20^2, which has cell #13^2 as a neighbor. Particles in cell #20^2 must check all the particles in cell #13^2 and all the levels below it, no matter the depth. Hence, each cell has a linked-list of neighbor cells even if particles do not exist inside those cells, and particles in each cell must loop over those neighbor cells to check if the particles that may be inside these are contained within their computational compact support.

3 Implementation results

Fluid flow simulations were conducted here with the sole objective of assessing the performance of the proposed implementation for NNPS in adaptive SPH, rather than to improve any flow solution, as the numerical accuracy and convergence of the SPH method used has been demonstrated elsewhere [4,6,10,11]. All the simulations described herein were carried out using HADynaSPH [4,11], applying a Wendland C6 kernel [30] with a ratio of the smoothing length to the initial particle spacing of 1.95, thus making the compact support 3.9 times larger than the particle spacing. This combination results, on average, in 44 particles inside the compact support.

3.1 Steady flow around a circular cylinder

The QCG algorithm was tested in the steady flow over a circular cylinder placed in a free-stream [31–34], at a Reynolds number $Re=20$. The cylinder is placed at the center of a $20D\times 20D$ domain, where the characteristic length D is the diameter of the solid body. Flow simulation data have been nondimensionalized using the diameter D and the free-stream velocity U_∞. Aiming to illustrate the advantages of the use of the QCG algorithm, adaptive particle refinement has been applied in ISPH to place the calculated values of (total) drag coefficient C_d and angle of separation θ_s within the limits given by previous numerical [31,32] and experimental [33] investigations. By considering a computational domain as large as the one used here, a uniform distribution of particles corresponding to a relatively high resolution $D/80$ would create more than 2.5 million particles. This simulation would be computationally very expensive and, clearly, such a high spatial resolution is not required in the far-field. The characteristics of the present test simulations and the resolutions employed, in both uniform and adaptive particle arrangements, are shown in Table 1.

The initial number of particles in the computational domain is indicated in Table 1 for each case. As the flow has open boundaries all around in order to simulate the free-stream, the number of particles in the domain varies only slightly during a simulation due to small imbalances in the instantaneous flow rate, even when using the uniform particle arrangement. Concerning the adaptive procedure, splitting the particles as these enter prescribed regions of the flow produces a large rise in the number of particles. The benchmark flow data for the present test case are listed in Table 2 for various studies, both experimental and numerical. The results of the present simulations reported in Table 1 confirm the convergence of C_d to the range of values given in Table 2 as the particle resolution is increased. It must also be mentioned that the aforementioned parameter was calculated at the simulation time $T_{sim}=5$, for which all the SPH simulations had converged.

The data presented in Table 1 demonstrated the independency of the drag coefficient for the simulations with particle resolutions finer than $D/20$. The convergence to an accurate flow solution can be observed in a detailed data comparison made against the

Table 1: Characteristics of test simulations and resulting (total) drag coefficient C_d.

Case	Particle arrange.	NNPS algorithm	Spatial resolution	Mass levels	No. of particles, N			C_d
					Initial	1000 it.	$T_{sim}=5$	
A	Uniform	Linked-list	$D/5$	1	9976	10140	10028	1.93
B			$D/10$	1	39920	40298	40125	2.05
C			$D/20$	1	159668	160068	160066	2.13
D	Adaptive	Linked-list	$D/5 \rightarrow D/5$	1	9976	10002	9964	1.98
I		QCG						
E		Linked-list	$D/5 \rightarrow D/10$	2	10937	12109	11466	2.08
J		QCG						
F		Linked-list	$D/5 \rightarrow D/20$	3	13916	15957	15747	2.13
K		QCG						
G		Linked-list	$D/5 \rightarrow D/40$	4	18028	19196	22616	2.13
L		QCG						
H		Linked-list	$D/5 \rightarrow D/80$	5	33156	33930	45107	2.13
M		QCG						

Table 2: Benchmark values of the angle of separation θ_s and the (total) drag coefficient C_d for steady flow past a circular cylinder placed in a free-stream at $Re=20$.

Referenced work	θ_s (deg)	C_d
Coutanceau and Bouard [33] [1]	42.3	–
Tritton [34] [1]	–	2.22
Taira and Colonius [32]	43.3	2.06
Calhoun [31]	45.5	2.19
Marrone et al. [35] [2]	43.0	2.20
Sen et al. [36] [2]	43.5	2.02

[1] Experimental study.
[2] Results in a confined domain.

numerical simulations of Sen et al. [36], who investigated the flow past a circular cylinder at various blockage ratios and Reynolds numbers. These authors have not provided data for $Re=20$ concerning the pressure distribution, hence a comparison with the pressure coefficient Cp published for the bounding values of $Re=15$ and 30 is made instead in Fig. 8a. A direct data comparison at $Re=20$ was nevertheless possible regarding nondimensional vorticity ω in Fig. 8b. Minor discrepancies found at some values of the angle θ defining the position along the edge of the circular cylinder may be attributed to a small effect of blockage, despite the choice of the lowest available value in the data set [36].

In all the simulations performed here employing the adaptive arrangement, the base resolution (i.e., the particle resolution far from the object) was $D/5$. This coarse resolution was then increased using regional criteria. Such regions were defined as ellipses in the

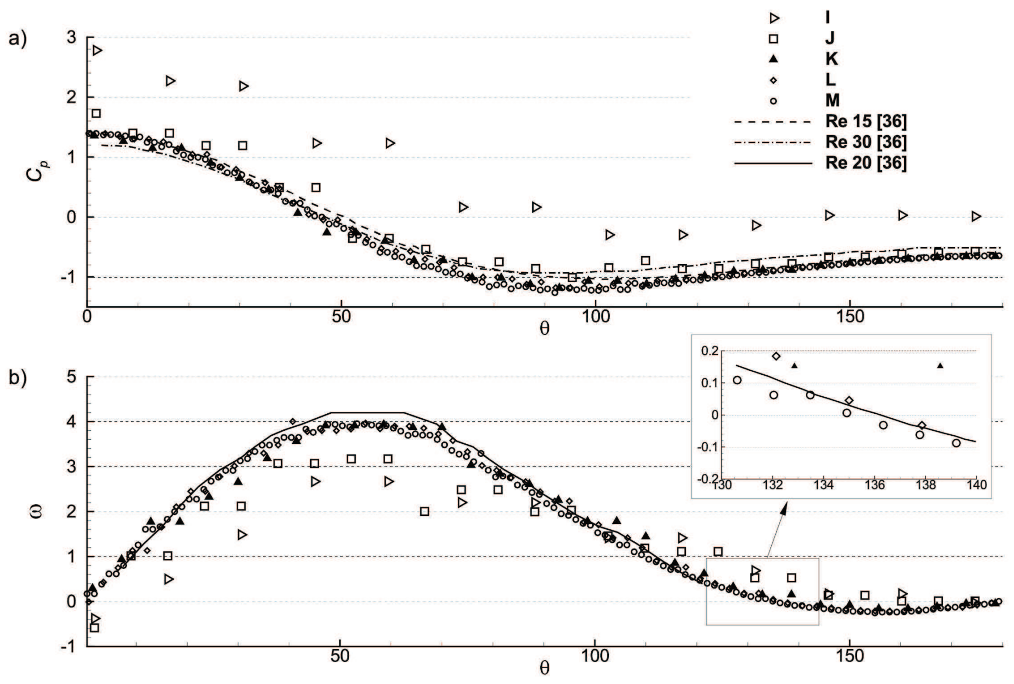


Figure 8: Comparison of present results with data published by Sen et al. [36] along the edge of the circular cylinder: a) distributions of Cp at $Re=20$ compared to the bounding values at $Re=15$ and 30; b) distributions of nondimensional vorticity at $Re=20$.

present simulations and finer resolutions were naturally employed near the object. Fig. 9 portrays the initial particle arrangement for the whole computational domain, as well as in vicinity of the circular cylinder (inset), for the simulation cases H and M.

The QCG algorithm has been introduced to improve the computational cost of the SPH simulations employing variable-mass particle distributions. The total CPU-time (including all operations) required for the test cases in Table 1 to achieve 1000 time iterations is shown in Fig. 10a. Looking only at the low-resolution simulations (test cases D and I, or E and J), it can be seen that the use of the QCG algorithm leads to a slightly higher computational cost than the linked-list due to the extra operations, as detailed in Section 2.2. However, as the number of coexisting $mass_{level}$ increases (leading also to a larger number of particles), the superiority of the QCG algorithm becomes evident. Namely, for the adaptive simulations employing the finest particle resolution (test cases H and M), the use of the QCG algorithm in NNPS reduces the total CPU-time by a factor of 2.73 with respect to the use of the linked-list.

The values of total CPU-time (including all operations) corresponding to the reference flow time $T_{sim}=5$ are shown in Fig. 10b also for the test cases in Table 1. However, depending on the particle resolution, the number of iterations it took the simulations to reach $T_{sim}=5$ differs. For example, those with the finest resolution up to $D/20$ only (cases

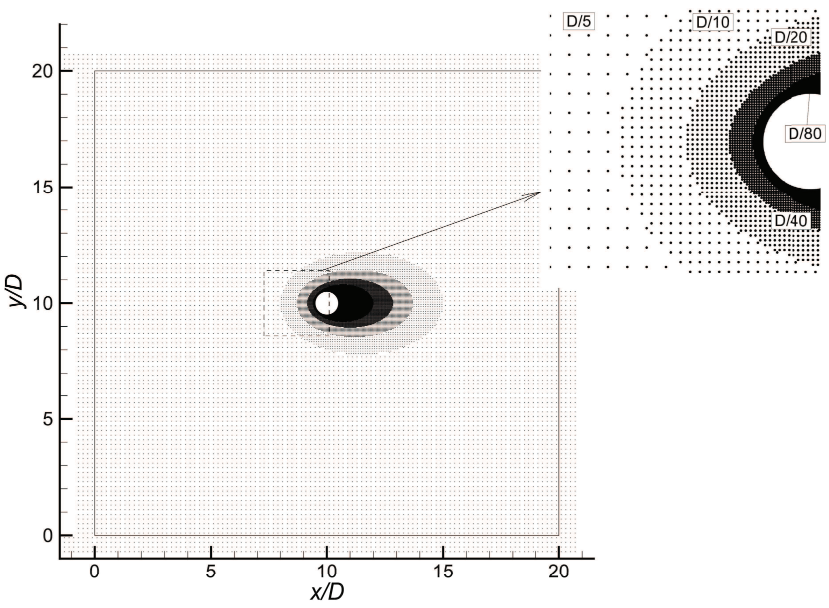
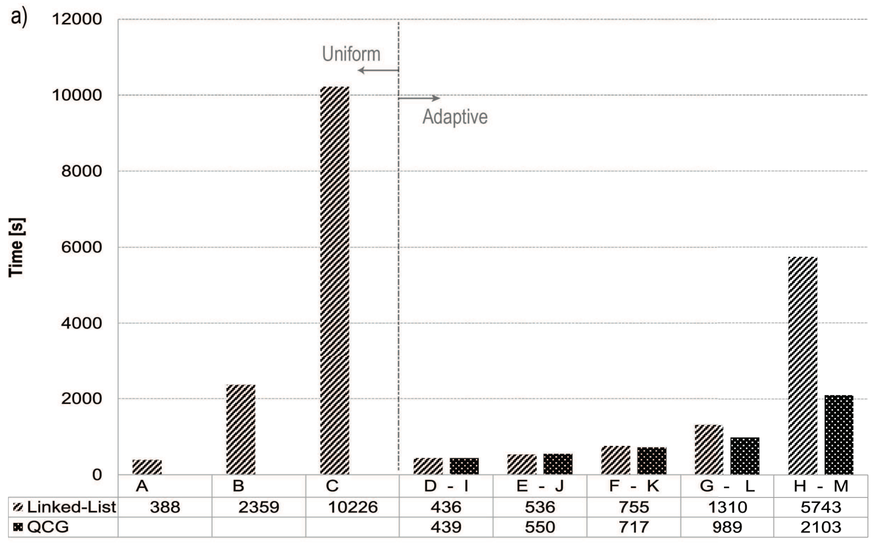


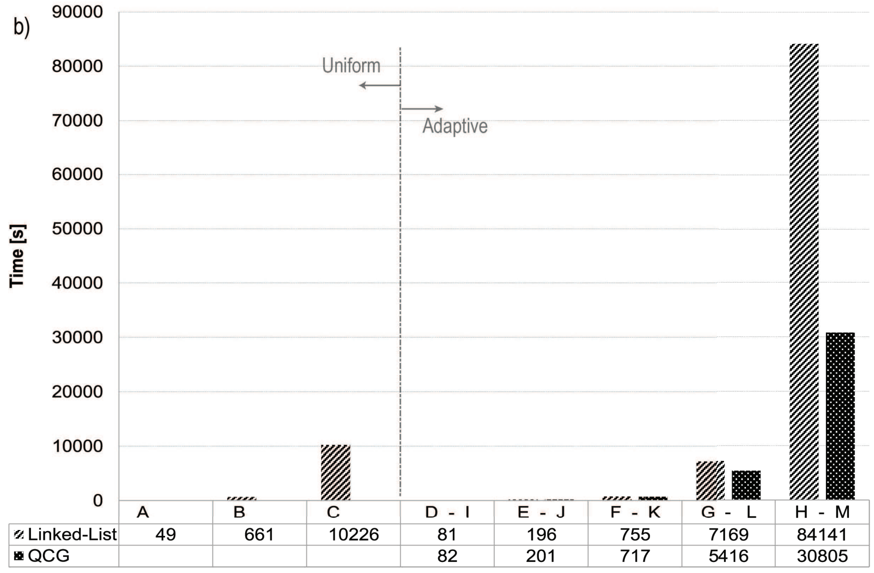
Figure 9: Initial particle arrangement in the computational domain and in vicinity of the circular cylinder (inset). The solid-line square represents the free-stream boundaries. Particles outside these boundaries are the so-called buffer particles employed in the open boundary treatment [11].

C, F and K) reached the foregoing time instant in less than 1000 iterations, but even for this moderate resolution the advantage of using the QCG algorithm for the adaptive particle arrangements is already noticeable. Drastic increases in such benefits are nevertheless obtained for the finest resolutions up to $D/40$ and $D/80$ (cases G and L, and H and M, respectively), though the simulation time grows significantly in those simulations, as expected due to (mainly) the larger number of particle interactions.

It was demonstrated that significant reductions in the total computational time required by adaptive SPH simulations may be achieved with the use of the QCG algorithm. However, it must be emphasized that most of the computational effort in ISPH simulations is on account of the iterative solver. Often, the calculation time for these solvers does not scale up linearly, thus biasing the analysis of the results in Fig. 10 concerning the mere performance of the NNPS algorithms. This issue may be assessed in detail in Fig. 11a, which shows the average calculation time per iteration to reach the first 1000 iterations for the various operations involved in the ISPH method, namely: a predictor step, in which particles are advanced to an intermediate position, and a velocity (u^*) is calculated based on the governing equations; the solution of a Pressure Poisson Equation (PPE), using the foregoing velocity field to compute the pressure field; a corrector step, using the pressure filed to determine the final velocity of the particles (u^{n+1}); and, apart from the already mentioned NNPS, a fifth contribution denoted here by others, corresponding to memory allocation and deallocation, matrix manipulation, and particle shifting.



	A	B	C	D - I	E - J	F - K	G - L	H - M
▨ Linked-List	388	2359	10226	436	536	755	1310	5743
■ QCG				439	550	717	989	2103



	A	B	C	D - I	E - J	F - K	G - L	H - M
▨ Linked-List	49	661	10226	81	196	755	7169	84141
■ QCG				82	201	717	5416	30805

Figure 10: Total CPU-time for the test cases in Table 1: a) first 1000 iterations; b) required to reach $T_{sim}=5$.

The comparison in Fig. 11a reveals the dominant cost of the PPE up to a moderate number of particles. However, as the amount of coexisting mass levels increases in the simulations (depicted in Fig. 11b), the cost of the NNPS grows dramatically, and only the use of the QCG algorithm avoids that this contribution becomes eventually the largest one in the set. The proposed algorithm ensures that the relative computational cost of NNPS with respect to components other than the PPE remains approximately constant, thus yielding that its use may be even more beneficial in fully explicit Lagrangian methods that are not limited by the computational cost of an implicit solver.

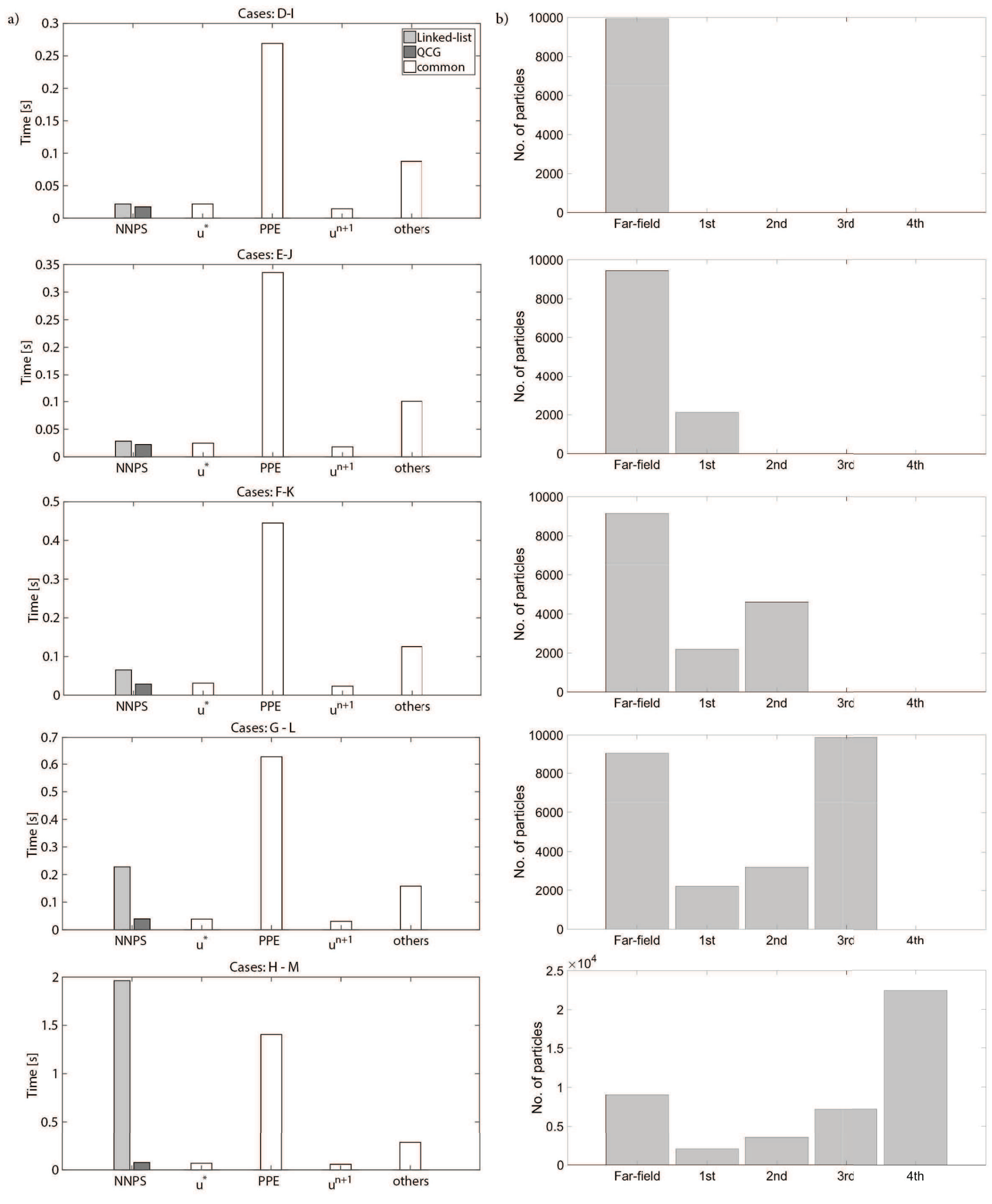


Figure 11: Computational statistics of the method and simulations. a) averaged calculation time for each component of the ISPH method for the first 1000 iterations; b) number of particles in the flow field at $T_{sim}=5$, sorted by mass level (e.g., 1st represents the first split level with respect to the uniform far-field resolution, and so on).

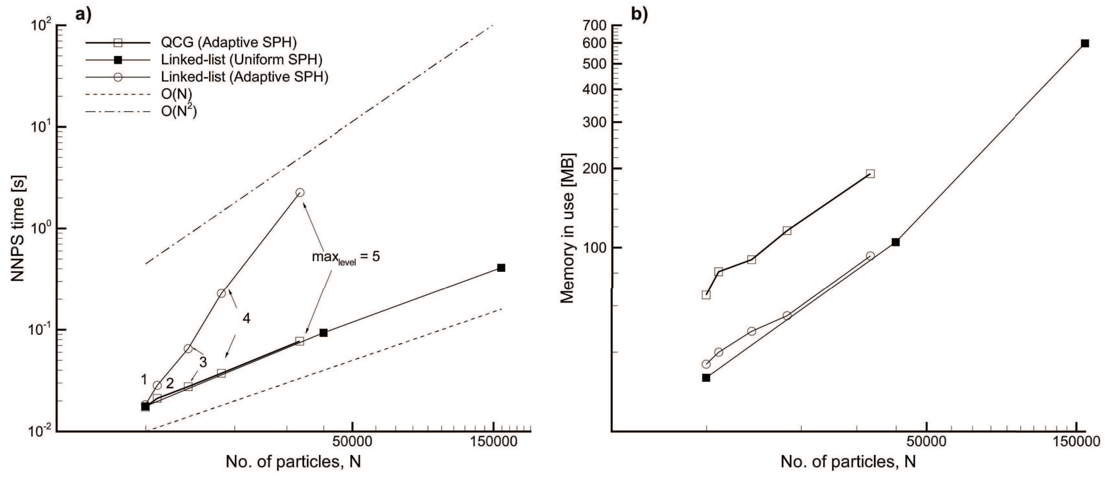


Figure 12: Averaged computational data per iteration for various implementations during the first 1000 iterations for the initial number of particles shown in Table 1. a) NNPS calculation time. Slopes of order $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ are also shown in dashed lines for reference. b) Memory in use vs. number of particles.

The performance of the QCG algorithm in adaptive SPH is compared with that of linked-list in adaptive and uniform SPH in Fig. 12. In Fig. 12a, slopes of order $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ are also shown in this figure for reference. It can be concluded that the goal of retrieving a computational effort of the NNPS of order $\mathcal{O}(N)$ with the QCG algorithm, similarly to that of linked-list in uniform resolution simulations, has been fully achieved. In the case of adaptive simulations using the classical linked-list algorithm, the computational effort corresponding to NNPS diverges sharply from $\mathcal{O}(N)$, thus making these calculations highly costly as the number of particles increase via the increment of $mass_{level}$ (up to 25.4 times more expensive with five coexisting mass levels, i.e. $max_{level}=5$). Contrasting with this improvement in computational cost, and due to the `POINTER` structures used in QCG implementation, the average memory in use per iteration more than doubled, as shown in Fig. 12b. Advancing from uniform to an adaptive implementation, the required storage would naturally increase to accommodate the extra variables required in the calculations, as the linked-list results demonstrate in Fig. 12b.

On the other hand, much less computational time was required by the QCG algorithm in an adaptive SPH simulation producing the same value of C_d as a uniform simulation using the classical linked-list algorithm, due to the ten-fold reduction in the number of particles employed (see Table 1). These savings allow for higher particle refinement to be used in the vicinity of the object, as local features of the flow field around the circular cylinder such as the flow separation angle $\theta=\theta_s$ are very sensitive to spatial resolution. This is illustrated in Fig. 13, where the flow patterns for cases A, B, C (uniform SPH) and I, J, K, L, M (adaptive SPH) are portrayed. The shaded sector inside the cylinder section provides a graphical representation of the maximum and minimum values reported in the literature for the angle of separation, as listed earlier in Table 2. It can be seen that

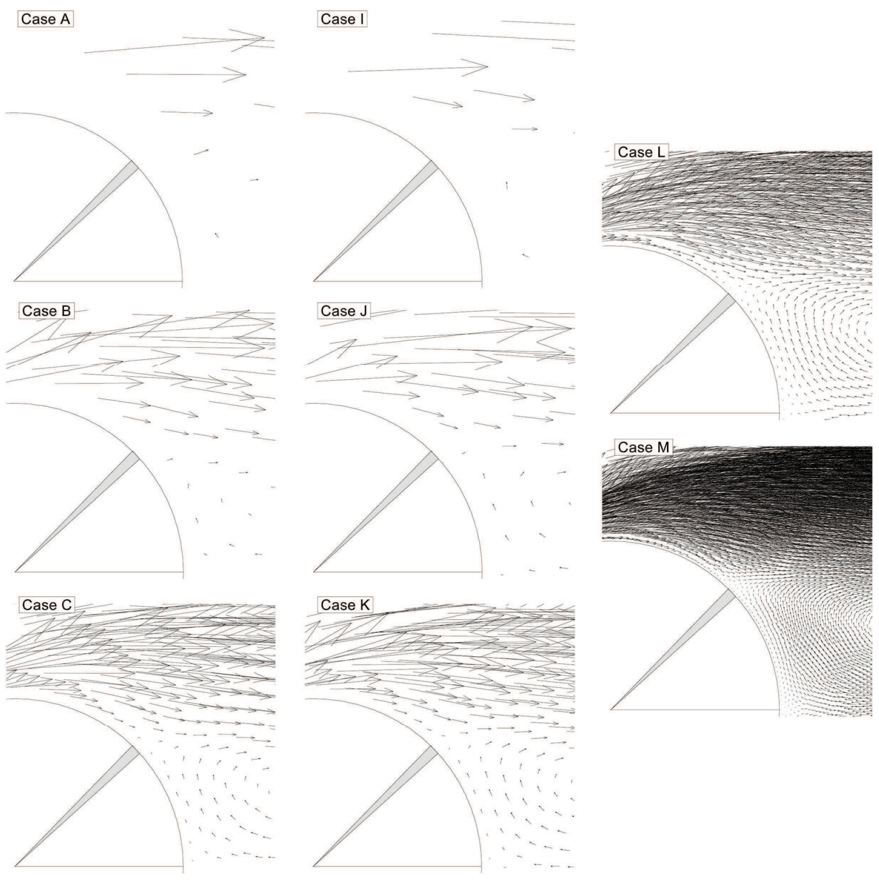


Figure 13: Effect of particle resolution on the converged velocity vector pattern around a circular cylinder placed in a free-stream at $Re=20$. The shaded sector within the cylinder section represents the maximum and minimum values reported in the literature for the angle of separation, as listed in Table 2.

reasonable values for θ_s may be computed already using the uniform particle arrangement with a resolution of $D/20$ (case C), but the associated uncertainty is high. Increased accuracy may be obtained by employing the adaptive particle arrangement with higher resolutions (cases L and M), together with the QCG algorithm, for a small fraction of the computational effort that would be required by comparable uniform simulations.

3.2 Plunging NACA0012 airfoil

The steady state study is complemented here by an investigation of unsteady flow around a NACA0012 airfoil plunging vertically in a free-stream at $Re=252$ [26,37]. Examples of the application of SPH to the analysis of streamlined bodies can be found in the work of Shadloo et al. [38]. In their comprehensive investigation, the flow over a stationary airfoil at various angles of attack and Reynolds numbers was studied, eventually giving rise to the establishment of self-sustained oscillations characterized by a Strouhal number. Conversely, in the case of the plunging airfoil considered here, the streamlined body is heaving at a specified frequency rather than remaining stationary. A Strouhal number characterizing this motion, which in turn controls the dynamics of the flow, can also be formed. However, as the foregoing parameter is associated to prescribed oscillations, it bears no direct relation with its counterpart describing naturally-occurring flow oscillations.

Whereas the previous flow problem allowed a detailed comparison between the performance of the linked-list method and that of the presently proposed algorithm, this additional flow problem is aimed at a sensitivity analysis of QCG algorithm to the complexity of the particle arrangement in the presence of a moving body. The airfoil is placed horizontally at a location $10C$ downstream from the inflow of a computational domain of size $30C \times 20C$. The free-stream (inlet) velocity U_∞ has been selected to obtain the intended value of the Reynolds number for an airfoil of chord length C. The two-dimensional symmetric NACA0012 airfoil plunges vertically around the mid-plane of the domain and its motion is described in time t by the following expression:

$$y(t) = h\sin(kt + \pi/2), \tag{3.1}$$

where the y-direction is perpendicular to the undisturbed stream, k stands for the reduced frequency, and h denotes the amplitude of the plunging motion. In Eq. (3.1) all quantities have been made nondimensional using C and U_∞ as length and velocity scales, respectively. The amplitude of the plunging motion has been kept fixed as $h=0.12$, whereas two distinct values of the reduced frequency, namely $k=4$ and 8, were chosen to generate the shedding of vortical structures exhibiting different characteristics. For additional information regarding this flow problem, the reader is invited to numerical and experimental studies published in the literature [26,37,39,40].

In the previous section, the superiority of the QCG algorithm with respect to classic linked-list was demonstrated, for calculations employing a moderate number of particles. Here, the focus is on the assessment of the performance of QCG for a complex flow problem involving a moving body. Details of the test cases considered, and the corresponding average number of particles present in the computational domain, are given in Table 3. The local refinement used for each of these cases is similar to that used in the flow problem previously considered, though enlarged to better capture vortex shedding phenomena from the body. In case Q only, this refinement scheme is coupled to a dynamic, vorticity-based splitting. Following the adaptive scheme described by Khorasanizade and Sousa [4], particles with absolute non-dimensional vorticity value above 25 are broken into four daughter particles. This would increase the resolution for the regions attached to the surface as well as at the center of the vortices.

Table 3 demonstrates the ability of SPH to predict the averaged drag coefficient with fair accuracy, and minor discrepancies may be due to the long transients reported by Yucel et al. [37]. A negative value of the drag coefficient indicates generation of thrust.

Table 3: Characteristics of the cases considered for the flow past a plunging NACA0012 airfoil in a free-stream at $Re=252$.

Case	h	k	Spatial resolution [1]	No. of particles	Present C_d	C_d [37]
N	0.12	4	$C/5 \to C/80$	55×10^3	0.158	0.189
O			$C/5 \to C/160$	76×10^3	0.162	
P		8	$C/5 \to C/160$	78×10^3	-0.182	-0.147
Q			$C/5 \to C/160+C/320$	94×10^3	-0.175	
R			$C/5 \to C/320$	136×10^3	-0.172	

[1] Regional (local) + dynamic particle refinement [4] where applicable.

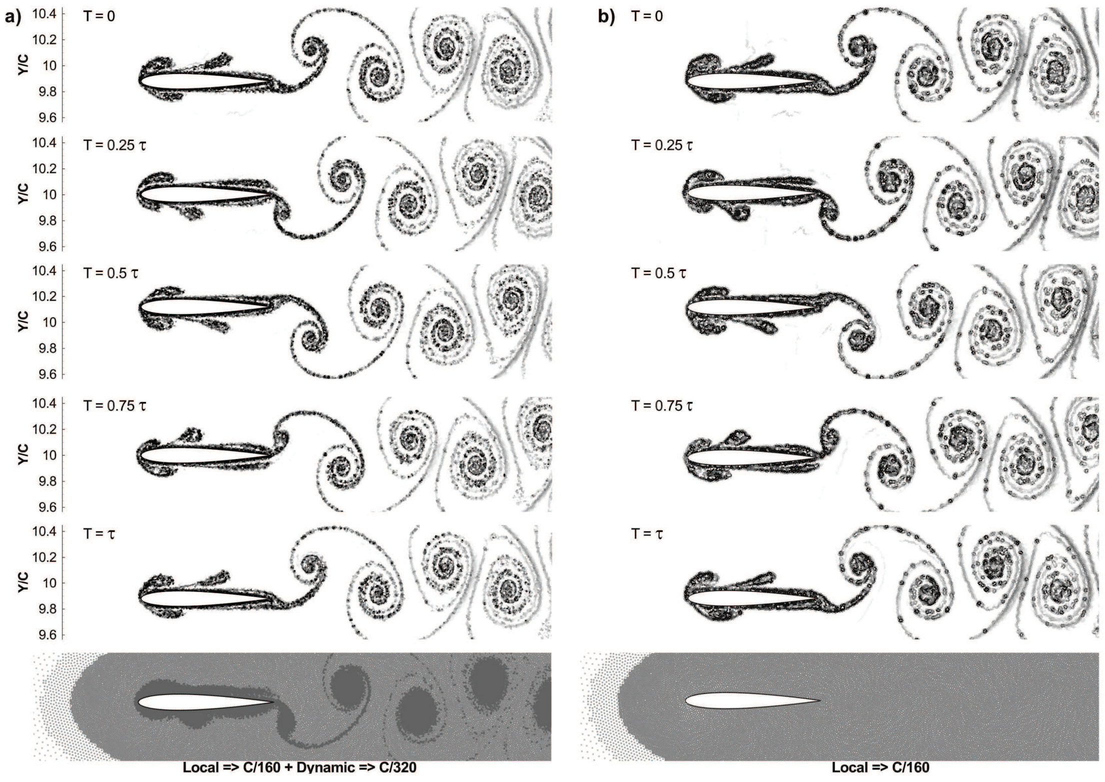


Figure 14: Effect of particle resolution in the vicinity of the plunging airfoil for $h=0.12$, $k=8$ and $Re=252$. Note: τ is the period of plunging calculated from k. a) LCS of case Q in a full cycle of plunging; b) LCS of case P in a full cycle of plunging. Corresponding particle distributions are shown at the bottom.

As expected, increasing the particle resolution around the moving airfoil improved numerical accuracy. Particle distribution and the effect of resolution on the flow structures are depicted in Fig. 14. Lagrangian Coherent Structures (LCS) have been calculated for the adaptive particle arrangement using the method introduced by Khorasanizade and Sousa [26]. It can be seen that, by increasing the particle resolution from $C/160$ to $C/320$

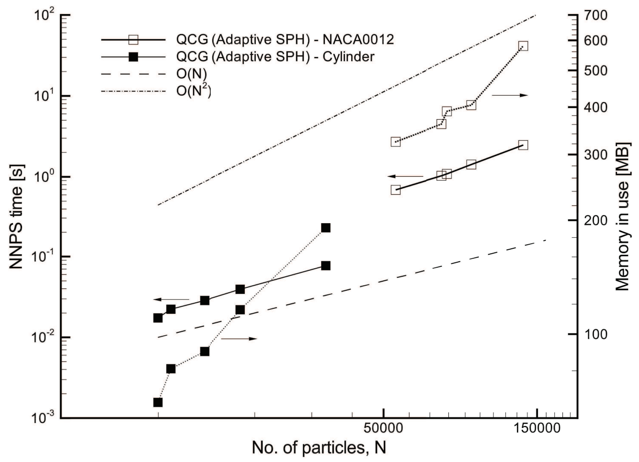


Figure 15: Average NNPS time and memory in use for each iteration vs. the number of particles for the flow cases shown in Table 3. Results of the circular cylinder are taken from Figure 12. For each test case, the solid lines indicate time, whereas the dotted lines depict the amount of memory used by the process at each iteration.

employing the dynamic refinement procedure, a better definition of the edges of the LCS is achieved, which is in-line with the data presented in Table 3.

The complexity of the particle arrangement, yielding larger ratios between the various $mass_{level}$, increases the load on NNPS algorithms. The average time required to conduct a NNPS and the amount of memory in use per iteration for the test cases of Table 3 are plotted in Fig. 15. For a better understanding, the results presented earlier in Fig. 12 are shown again. The amount of memory required for the test cases encompassing a plunging airfoil increases with the number of particles without a defined trend. The large ratio between the number of particles existing in each $mass_{level}$ is responsible for the sudden change observed between Q and R, despite $max_{level} = 7$ remaining the same in both cases. In addition, the average time for case N (plunging airfoil) is almost one order of magnitude larger than that of case M (circular cylinder). This is mainly due to the use of larger splitting zones (as mentioned earlier), so that the vortical structures shed from the moving body could be resolved with a higher accuracy. Naturally, this implies an additional load on any neighbor search procedure. As QCG is also a NNPS algorithm, one can only expect that its performance depends on the particle splitting strategy. However, once a strategy for that purpose has been selected, the computational cost still scales linearly with the total number of particles, as illustrated in Fig. 15 for both the circular cylinder and the plunging airfoil problems.

4 Conclusions

An improved and optimized linked-list based NNPS algorithm coined Quadtree Cells Grid (QCG) has been introduced for use in adaptive SPH simulations. This algorithm couples the simplicity and efficiency of linked-list search with a hierarchical structure of cells akin to quadtrees in 2D problems. SPH particles are placed in each cell based on their mass level and the corresponding level that each cell may contain. The mass of the coarsest particle is used as a reference to define the mass levels. As the SPH particles are split into smaller ones, their associated mass level is increased and cells are tagged to be split as well if they contain a particle characterized by a higher mass level. Subsequently, these marked entities produce smaller daughter cells containing those smaller particles, and the linked-list grid structure is updated. The neighbor cells list of each cell is constructed based on the cells in the vicinity of their parent, as the cells are split in desired regions. This algorithm has been tested on various solutions of a fluid flow problem employing multiple levels of particle mass.

The first problem chosen to illustrate the capabilities of the QCG algorithm was the two-dimensional flow over a circular obstacle of diameter D placed in a free-stream at $Re=20$. Several SPH simulations were carried out in a computational domain of size $20D\times 20D$. Owing to the high computational cost, uniform resolution calculations using the classical linked-list algorithm have been conducted up to a particle spacing of $D/20$ only. The advantages of using the QCG algorithm were demonstrated in adaptive SPH simulations with a base resolution of $D/5$ and a finest particle resolution of $D/80$ in the vicinity of the body. Convergence of integral and local flow results to benchmark data was verified. It was concluded that the proposed algorithm reaches the peak of its performance as the number of mass levels increases. In the most refined adaptive simulation considered, corresponding to 5 coexisting mass levels, the QCG algorithm allowed for a reduction in total computational cost by a factor of 2.73, when compared to the traditional linked-list algorithm. However, if only the computational effort required by the NNPS is accounted for, speedups up to 25.4 were obtained as the goal of retrieving the order $\mathcal{O}(N)$ with the proposed algorithm was achieved.

A second problem, described by the flow generated by a vertically plunging NACA0012 airfoil in a freestream with $Re=252$ was studied as well. A larger domain of $30C\times 20C$ was chosen to reduce the effect of the boundaries on the results. The plunging motion of the studied airfoil was characterized by a non-dimensional amplitude $h=0.12$ and reduced frequencies $k=4$ and 8. A total number of particles as large as 136,000 was used in the most refined case, with a maximum number of coexisting mass levels of 7. Due to the complexity of the flow and particle arrangement, the neighbor finding for this flow problem required significantly higher computational times. However, the order $\mathcal{O}(N)$ of the QCG algorithm was again achieved.

Altogether these results provided firm support to the conclusion that the QCG algorithm is computationally efficient. One should emphasize that, due to the necessary `POINTER`s and structures used in the implementation, the QCG algorithm shows higher values of memory use with respect to the traditional linked-list. Finally, it must be noted that although the present test simulations were two-dimensional only, the algorithm has been designed irrespectively of the dimensions of the system.

Acknowledgments

This work has been supported by Fundação para a Ciência e a Tecnologia (FCT), through IDMEC, under LAETA, project UID/EMS/50022/2013. The financial support via FCT grant SFRH/BSAB/114588/2016 is also acknowledged.

References

[1] G. R. Liu and M. B. Liu, Smoothed Particle Hydrodynamics: A Meshfree Particle Method, World Scientific Publishing Co. Pte. Ltd., Singapore, 2003.

[2] H. Gotoh and A. Khayyer, Current achievements and future perspectives for projection-based particle methods with applications in ocean engineering, J. Ocean Eng. Mar. Energy, **2** (2016), 251278.

[3] M. S. Shadloo, G. Oger and D. Le Touzé, Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges, Comput. Fluids, **136** (2016), 1134.

[4] Sh. Khorasanizade and J. M. M. Sousa, Dynamic flow-based particle splitting in smoothed particle hydrodynamics, Int. J. Numer. Methods Eng., **106** (2016), 397410.

[5] J. Feldman and J. Bonet, Dynamic refinement and boundary contact forces in SPH with applications in fluid flow problems, Int. J. Numer. Methods Eng., **72** (2007), 295324.

[6] Sh. Khorasanizade and J. M. M. Sousa, An adaptive fully-Lagrangian meshless method for incompressible laminar flow airfoil studies, Aerosp. Sci. Technol., **64** (2017), 161170.

[7] A. Khayyer, H. Gotoh, Y. Shimizu, K. Gotoh, H. Falahaty and S. Shao, Development of a projection-based SPH method for numerical wave flume with porous media of variable porosity, Coast. Eng., **140** (2018), 122.

[8] A. Khayyer, H. Gotoh, H. Falahaty and Y. Shimizu, An enhanced ISPHSPH coupled method for simulation of incompressible fluidelastic structure interactions, Comput. Phys. Commun., **232** (2018), 139164.

[9] J. M. Domínguez, A. J. C. Crespo and M. Gómez-Gesteira, Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method, Comput. Phys. Commun., **184** (2013), 617627.

[10] Sh. Khorasanizade and J. M. M. Sousa, A two-dimensional Segmented Boundary Algorithm for complex moving solid boundaries in Smoothed Particle Hydrodynamics, Comput. Phys. Commun., **200** (2016), 6675.

[11] Sh. Khorasanizade and J. M. M. Sousa, An innovative open boundary treatment for incompressible SPH, Int. J. Numer. Methods Fluids, **80** (2016), 161180.

[12] H. Samet, The Quadtree and Related Hierarchical Data Structures, ACM Comput. Surv., **16** (1984), 187260.

[13] A. Klinger and C. R. Dyer, Experiments on picture representation using regular decomposition, Comput. Graph. Image Process., **5** (1976), 68105.

[14] M. Yerry and M. Shephard, A Modified Quadtree Approach To Finite Element Mesh Generation, IEEE Comput. Graph. Appl., **3** (1983), 3946.

[15] M. S. Shephard and M. K. Georges, Automatic three-dimensional mesh generation by the finite octree technique, Int. J. Numer. Methods Eng., **32** (1991), 709749.

[16] P. L. Baehmann, S. L. Wittchen, M. S. Shephard, K. R. Grice, and M. A. Yerry, Robust, geometrically based, automatic two-dimensional mesh generation, Int. J. Numer. Methods Eng.,

24 (1987), 10431078.
[17] B. Crouse, E. Rank, M. Krafczyk and J. Tölke, A LB-based approach for adaptive flow simulations, Int. J. Mod. Phys B, **17** (2003), 109112.
[18] Z. J. Wang, A Quadtree-based adaptive Cartesian/Quad grid flow solver for Navier-Stokes equations, Comput. Fluids, **27** (1998), 529549.
[19] S. Foroughi, S. Jamshidi and M. Masihi, Lattice Boltzmann method on quadtree grids for simulating fluid flow through porous media: A new automatic algorithm, Physica A, **392** (2013), 47724786.
[20] Y. Chen, Q. Kang, Q. Cai and D. Zhang, Lattice Boltzmann method on quadtree grids, Phys. Rev. E, **83** (2011), 26707.
[21] X. Xia and Q. Liang, A GPU-accelerated smoothed particle hydrodynamics (SPH) model for the shallow water equations, Environ. Model. Softw., **75** (2016), 2843.
[22] O. Awile, F. Büyükkeçeci, S. Reboux and I. F. Sbalzarini, Fast neighbor lists for adaptive-resolution particle simulations, Comput. Phys. Commun., **183** (2012), 10731081.
[23] L. Verlet, Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules, Phys. Rev., **159** 1967), 98103.
[24] A. Khayyer, H. Gotoh and Y. Shimizu, Comparative study on accuracy and conservation properties of two particle regularization schemes and proposal of an optimized particle shifting scheme in ISPH context, J. Comput. Phys., **332** (2017), 236256.
[25] Sh. Khorasanizade and J. M. M. Sousa, Using a fully-Lagrangian meshless method for the study of aerosol dispersion and deposition, Aerosol Sci. Technol., **50** (2016), 926936.
[26] J. M. M. Sousa and Sh. Khorasanizade, A Fully Lagrangian Approach to Study the Flow Past Heaving Airfoils Placed in a Freestream, 56th AIAA Aerosp. Sci. Meet., American Institute of Aeronautics and Astronautics, Kissimmee, 2018.
[27] S. Shao, Incompressible SPH simulation of water entry of a free-falling object, Int. J. Numer. Methods Fluids, **59** (2009), 91115.
[28] J. M. Domínguez, A. J. C. Crespo, M. Gómez-Gesteira and J. C. Marongiu, Neighbour lists in smoothed particle hydrodynamics, Int. J. Numer. Methods Eng., **67** (2011), 20262042.
[29] J. C. Adams, W. S. Brainerd, R. A. Hendrickson, R. E. Maine, J. T. Martin and B. T. Smith, The Fortran 2003 Handbook, Springer-Verlag, London, 2009.
[30] Y. Jiang, L. Fang, X. Jing, X. Sun and F. Leboeuf, A second-order numerical method for elliptic equations with singular sources using local filter, Chinese J. Aeronaut., **26** (2013), 13981408.
[31] D. Calhoun, A Cartesian Grid Method for Solving the Two-Dimensional Streamfunction-Vorticity Equations in Irregular Regions, J. Comput. Phys., **176** (2002), 231275.
[32] K. Taira and T. Colonius, The immersed boundary method: A projection approach, J. Comput. Phys., **225** (2007), 21182137.
[33] M. Coutanceau and R. Bouard, Experimental determination of the main features of the viscous flow in the wake of a circular cylinder in uniform translation. Part 1. Steady flow, J. Fluid Mech., **79** (1977), 231256.
[34] D. J. Tritton, Experiments on the flow past a circular cylinder at low Reynolds numbers, J. Fluid Mech., **6** (1959), 547567.
[35] S. Marrone, A. Colagrossi, M. Antuono, G. Colicchio and G. Graziani, An accurate SPH modeling of viscous flows around bodies at low and moderate Reynolds numbers, J. Comput. Phys., **245** (2013), 456475.
[36] S. Sen, S. Mittal and G. Biswas, Steady separated flow past a circular cylinder at low Reynolds numbers, J. Fluid Mech., **620** (2009), 89119.

[37] S. B. Yucel, M. Sahin and M. F. Unal, Strong transient effects of the flow around a harmonically plunging NACA0012 airfoil at low Reynolds numbers, Theor. Comput. Fluid Dyn., **29** (2015), 391412.

[38] M. S. Shadloo, A. Zainali, M. Yildiz and A. Suleman, A robust weakly compressible SPH method and its comparison with an incompressible SPH, Int. J. Numer. Methods Eng., **89** (2012), 939956.

[39] J. C. S. Lai and M. F. Platzer, Jet Characteristics of a Plunging Airfoil, AIAA J., **37** (1999), 15291537.

[40] K. D. Jones, C. M. Dohring and M. F. Platzer, Experimental and computational investigation of the Knoller-Betz effect, AIAA J., **36** (1998), 12401246.