Commun. Comput. Phys.
doi: 10.4208/cicp.010415.110915a

Vol. **19**, No. 3, pp. 770-800
March 2016

# Efficient Implementation of Smoothed Particle Hydrodynamics (SPH) with Plane Sweep Algorithm

Dong Wang[1], Yisong Zhou[2] and Sihong Shao[2,*]

[1] *State Key Laboratory of ASIC and System, School of Microelectronics, Fudan University, Shanghai 201203, China.*
[2] *LMAM and School of Mathematical Sciences, Peking University, Beijing 100871, China.*

Received 1 April 2015; Accepted (in revised version) 11 September 2015

**Abstract.** Neighbour search (NS) is the core of any implementations of smoothed particle hydrodynamics (SPH). In this paper, we present an efficient $\mathcal{O}(N\log N)$ neighbour search method based on the plane sweep (PW) algorithm with $N$ being the number of SPH particles. The resulting method, dubbed the PWNS method, is totally independent of grids (i.e., purely meshfree) and capable of treating variable smoothing length, arbitrary particle distribution and heterogenous kernels. Several state-of-the-art data structures and algorithms, e.g., the segment tree and the Morton code, are optimized and implemented. By simply allowing multiple lines to sweep the SPH particles simultaneously from different initial positions, a parallelization of the PWNS method with satisfactory speedup and load-balancing can be easily achieved. That is, the PWNS SPH solver has a great potential for large scale fluid dynamics simulations.

**AMS subject classifications**: 76M28, 74F10, 35Q30, 68W05, 65D18

**Key words**: Smoothed particle hydrodynamics, meshfree method, neighbour search, plane sweep algorithm, Morton code, segment tree, quadtree, parallelization, dam break.

## 1 Introduction

Smoothed particle hydrodynamics (SPH) is a fully meshfree Lagrangian computational fluid dynamics (CFD) method developed independently by Lucy [1], Gingold and Monaghan [2] in 1977 for astrophysical studies. Due to its robustness in dealing with complex physical problems [3–9], SPH has since been successfully utilized to a large range of fields, such as ocean engineering [10], casting processes [11], semiconductor manufacturing [12] and so on. In SPH, the system is represented by discrete particles carrying their

*Corresponding author. *Email addresses:* wangdong11@fudan.edu.cn (D. Wang), failed.zys@gmail.com (Y. Zhou), sihong@math.pku.edu.cn (S. Shao)

own physical quantities. The quantity of each particle is interpolated from its neighbouring particles through a kernel function with compact support. Thus, efficient searching of particle neighbours is crucial for SPH performance. On account of the compact support property of the kernel function, only neighbour particles within its support domain are involved in calculating the force acting on a given particle. This significantly reduces the computing cost and serves as the start point of dozens of efficient neighbour search algorithms adopted in any SPH implementations over the $\mathcal{O}(N^2)$ direct traversing method ($N$ is the number of SPH particles). An early survey of neighbour search algorithms was given in [13] and a collection of improved ones have been later proposed by the SPH community, including the grid-link-list method [14, 15], the hierarchical tree structured methods [16, 17], the Hilbert-curve decomposition method [18, 19], the Z-order curve indexing method [20], the spatial hashing method [21, 22] and the adaptive-resolution cell lists method [23]. Among them, both the grid-link-list method and the hierarchical tree structured methods are much more popular.

In the grid-link-list method, the complexity of which is $\mathcal{O}(N\log N)$ [14, 15], the entire domain is first divided into uniform grids sized in the radius of the support domain and then the neighbours of a given particle are searched only within its home and adjacent grids. This method is preferred in incompressible or weakly-compressible fluid problems for the smoothing length is often constant or varies in a small range. However, the grid-link-list method is not purely "mesh-free" for the search procedure highly depends on the grids. When the system consists of sparsely distributed particles (e.g., often encountered in adaptive particle refinement/derefinement [24, 25]) or involves a large variation of the support domain, a great many grids may contain no particles, leading to an inefficient use of memory, and then the grid-link-list method is no longer suitable.

Another widely accepted idea for neighbour search comes from hierarchical tree structures and has been applied to SPH by many researchers using different algorithms. Hernquist et al. [16] incorporated the Barnes-Hut algorithm [26] into SPH calculation, where an octree is built hierarchically by dividing the entire domain into small cells and each particle is represented by a leaf node, and then the neighbour search is performed by descending from the top of the tree with a search cube checking whether a node is within the neighbouring area. Benz et al. [17] used a bottom-up tree structure, where the mutual neighbouring particles and tree nodes are organized in higher-level aggregate nodes recursively in a bottom-up hierarchy, and then a binary tree with neighbouring particles grouped together is built. These two tree methods are favoured in the astrophysics community for the ease of computing long-range gravitational forces, the mesh-less property and the flexibility for variational smoothing length. Although the complexity of these tree-based methods is reported to be $\mathcal{O}(N\log N)$ [16], it is shown that the logarithmic term can deteriorate into $\mathcal{O}(dN^{1-1/d})$ with $d$ being the spatial dimension [27]. Moreover, parallelizing these tree-based methods is not an easy task and sometimes requires a redesign of the original algorithm [28].

The neighbour search in SPH is closely connected with the properties of smoothing kernels. Under most circumstances, the first choice of SPH community is the isotropic

smoothing kernel, of which the shape is usually a circle in 2D (sphere in 3D) with the radius $r_c=\kappa h$, where $\kappa$ is the kernel-dominant factor and $h$ is the smoothing length. Nearly all the neighbour search methods are developed for the isotropic kernels. Despite its broad application, it is reported by Yu et al. [29] that an anisotropic kernel with an elliptic shape leads to better results than the isotropic kernel in computer graphics field. However, much to the authors' knowledge, there is no such neighbour search method adaptable to both isotropic and anisotropic kernels. Instead, a neighbour search method for isotropic kernels is used in [29], where the ellipsoidal support domains are covered by uniform grid cells and extra work is needed to filter the particles outside the support domain.

In order to treat the variable smoothing length, the arbitrary particle distribution and the heterogenous kernels as well as to provide effortless parallelization, in this study we propose an $\mathcal{O}(N\log N)$ neighbour search (NS) method based on the plane sweep (PW) algorithm [30]. This algorithm has been successfully applied in a large range of computational geometry problems, such as the VLSI design automation [31] and geographic information systems [32], but rarely used to solve CFD problems. In the resulting neighbour search method, dubbed the PWNS method, neighbour search is performed by a line sweeping over all the SPH particles. That is, no gird is needed any more, i.e., the PWNS method is purely "meshfree", which offers great flexibility to process variable smoothing length and irregular support domain in SPH. Moreover, the raw coordinates of particles are mapped into integer indices without losing their localities and those data structures and algorithms well tuned for integers can be adopted. Also with the help of integer indices, spatial coordinates can be further mapped to the Morton code [33], which provides a more efficient 3D implementation (vide post). Meanwhile, the particle set can be easily divided into several balanced subsets only according to the integer indices. That is, the PWNS method is naturally appropriate for parallelization by sweeping these balanced subsets simultaneously. All of these properties give the PWNS method competitive advantages over other existing neighbour search methods.

The rest of the paper is organized as follows. In Section 2, basic formulation of SPH is introduced. Section 3 is devoted into describing the PWNS method with details and corresponding parallel implementations can be found in Section 4. Numerical experiments with discussions are presented in Section 5 for demonstrating the performance of the PWNS method. The paper is concluded in Section 6 with a few remarks.

## 2 Basic formulation of SPH

The basic principle of SPH is that any function $A(\boldsymbol{r})$ can be expressed by an integral over the entire domain $\Omega$ as

$$A(\boldsymbol{r})=\int_\Omega A(\boldsymbol{r}')\delta(\boldsymbol{r}-\boldsymbol{r}')d\boldsymbol{r}', \tag{2.1}$$

where $\boldsymbol{r}$ and $\boldsymbol{r}'$ are the position vectors, $\delta(\boldsymbol{r}-\boldsymbol{r}')$ is the Dirac delta function. Replacing the Dirac delta function by a weight function $W(\boldsymbol{r}-\boldsymbol{r}',h)$, we have the typical interpolation formula as

$$\langle A(\boldsymbol{r})\rangle = \int_\Omega A(\boldsymbol{r}')W(\boldsymbol{r}-\boldsymbol{r}',h)d\boldsymbol{r}', \tag{2.2}$$

where $\langle A(\boldsymbol{r})\rangle$ is used to represent the SPH interpolation in the sense that

$$\lim_{h\to 0} W(\boldsymbol{r}-\boldsymbol{r}',h) = \delta(\boldsymbol{r}-\boldsymbol{r}'), \tag{2.3}$$

subject to the normalization condition

$$\int_\Omega W(\boldsymbol{r}-\boldsymbol{r}',h)d\boldsymbol{r}' = 1. \tag{2.4}$$

Here the weight function $W(\boldsymbol{r}-\boldsymbol{r}',h)$ is usually named as the smoothing kernel with $h$ being the smoothing length. Given that for hydrodynamical quantities long-range forces are usually negligible, we often employ an isotropic smoothing kernels with compact support in practice, i.e., requiring

$$W(\boldsymbol{r}-\boldsymbol{r}',h) = 0 \quad \text{when} \quad |\boldsymbol{r}-\boldsymbol{r}'| \geqslant \kappa h, \tag{2.5}$$

where $\kappa$ is a constant related to the smoothing kernel, and defines the *support domain* (with the radius $r_c = \kappa h$) of the particle at $\boldsymbol{r}$, namely the non-zero area of the smoothing kernel. The isotropic kernel can be written as

$$W(\boldsymbol{r}-\boldsymbol{r}',h) = \frac{\sigma}{h^d} f\Big(\frac{|\boldsymbol{r}|}{h}\Big), \tag{2.6}$$

where $\sigma$ is the normalization factor, $d$ is the spatial dimension, $f(\frac{|\boldsymbol{r}|}{h})$ is usually a symmetric spline with compact support. Recently, Yu et al. [29] proposed an anisotropic kernel as

$$W(\boldsymbol{r},\boldsymbol{G}) = \sigma \det(\boldsymbol{G}) f(\|\boldsymbol{G}\boldsymbol{r}\|), \tag{2.7}$$

where $h$ in Eq. (2.6) is replaced by a $d\times d$ real positive definite matrix $\boldsymbol{G}$.

Based on the Lagrangian specification of the flow field, the SPH method approximates the solutions of the equation of fluid dynamics by replacing the fluid with a set of particles moving with the flow and evaluates hydrodynamical properties of any particle by calculating the smoothing kernel interpolation of the values from other particles. Let us consider a set of $N$ SPH particles such that particle $i$ ($1 \leq i \leq N$) has mass $m_i$, density $\rho_i$ and position $\boldsymbol{r}_i$, we have the particle approximation of Eq. (2.2) as

$$A_i = \sum_j A_j W_{ij} V_j, \tag{2.8}$$

where $i$ and $j$ denote SPH particles, $W_{ij}=W(\boldsymbol{r}_i-\boldsymbol{r}_j,h)$, $V_j$ is the volume of particle $j$. On account of the compact support of the smoothing kernel, the summations in Eq. (2.8) can be reduced from over all the particles to only the neighbouring particles contained in the support domain of the particle $i$. This will greatly increase the efficiency of SPH if all the neighbours of a given particle can be efficiently collected. To this end, we will present and implement a fast neighbour search method base on a plane sweep algorithm.

## 3 Plane sweep neighbour search (PWNS)

A compact support domain in SPH is usually a local subregion of the entire domain. The most commonly used support domain is in the shape of circle (2D) or sphere (3D) with the radius $r_c=\kappa h$. For generality, as mentioned in Section 1, the compact support domain may also have a shape of ellipse [29], square or rectangular. Usually, the SPH particle stands at the centre of its support domain. In this work, since the plane sweep algorithm works in orthogonal systems, the circular (or elliptical) support domain can be transformed to a slightly larger rectangular one, i.e., the circumscribed rectangle, as shown in Fig. 1. As implied by its name, the plane sweep algorithm sweeps all the SPH particles along a specified direction with a sweep line. During the sweep, the line stops at every *event* point to obtain useful information and make necessary operations. No backtracking is needed and the sweep will only be performed once.

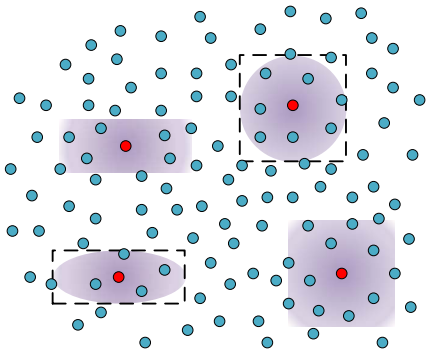


Figure 1: A 2D sketch of support domains (shadow area) with different shapes for different particles (red). For any circular or elliptical support domain, the search for the neighbours is carried out in a slightly larger circumscribed rectangle as the PWNS algorithm prefers a rectangular support domain.

To demonstrate the PWNS method, we take a 2D system as example. The positive direction of the $x$-axis is chosen as the sweep direction. The $x$-coordinate of particle $i$ is denoted by $x_i$, the size of the rectangular support domain for particle $i$ is characterized by the vector $\boldsymbol{r}_i=(a_i,b_i)$ with $a_i$, $b_i$ being the half length of the edge in $x$-direction, $y$-direction respectively. The values of $a_i$, $b_i$ can be set freely to wrap any size and any shape of support domains (see Fig. 1). That is, the variable smoothing length is naturally supported. Meanwhile, both isotropic and anisotropic kernels are fully covered in the

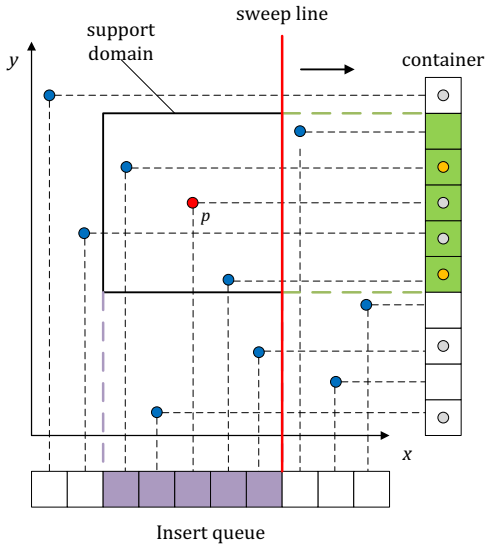



Figure 2: A diagram of the 2D PWNS method. The red line sweeps from the leftmost particle to the rightmost one. All the particles before the sweep line have been inserted into the container. For a given particle $p$ (red), only those particles (yellow) standing inside the vertical projection (green zone) and inserted between $e_p^{\text{limit}}$ and $e_p^{\text{query}}$ (purple zone) are regarded as its neighbours.

PWNS method. The left (resp. right) boundary of the support domain of particle $i$ is $x_i-a_i$ (resp. $x_i+a_i$). We put $x_i-a_i$, $x_i$ and $x_i+a_i$ for all $1\leq i\leq N$ together, sort them in ascending order and finally obtain an *event queue* $\{e_k \mid 1\leq k\leq 3N\}$. All events in $\{e_k\}$ can be classified into three categories according to their locations in support domains: $e^{\text{limit}}$ for left boundaries like $x_i-a_i$, $e^{\text{insert}}$ for particle positions like $x_i$ and $e^{\text{query}}$ for right boundaries like $x_i+a_i$. Those events of different types but sharing identical coordinate are arranged in the following specified order: $e^{\text{limit}}<e^{\text{insert}}<e^{\text{query}}$. For a given particle $i$, the related three events are marked respectively as $e_i^{\text{limit}}$, $e_i^{\text{insert}}$ and $e_i^{\text{query}}$. In consequence, the $x$-axis in space is transformed into the time domain. Then the neighbour search is performed with the line touching each event chronologically. In order to record what happens during the sweep, a container $\mathcal{T}$ representing the remaining spatial dimension should be constructed. The procedure of the PWNS method is presented as follows:

1. All the SPH particles with their insert events touched by the sweep line are inserted into the container $\mathcal{T}$.

2. When the line touches $e_i^{\text{query}}$, all the particles in the container will be checked and only those particles both located inside the projection of the support domain in $y$-axis and inserted between $e_i^{\text{limit}}$ and $e_i^{\text{query}}$ will be chosen as the neighbours of particle $i$.

A diagram of the 2D PWNS procedure is shown in Fig. 2 and the skeleton is described in Algorithm 1. As shown in Algorithm 1, a "smart" container is of great importance

**Algorithm 1:** The skeleton of the PWNS method for SPH.

**Input** : The number of SPH particles $N$, the particle set $\{i\,|\,1\leqslant i\leqslant N\}$
**Output**: The neighbour list of each particle

```
Merge all the elements in {x_i − a_i | 1⩽i⩽N}, {x_i | 1⩽i⩽N}, {x_i + a_i | 1⩽i⩽N}
together and sort them in ascending order to obtain the event queue
{e_k | 1⩽k⩽3N};
Initialize T;
for k = 1,2,...,3N do
    if e_k = e_i^insert then
        Insert i into T;
    end if
    if e_k = e_i^query then
        Query all the neighbours of i inside T;
    end if
end for
```

to the efficiency of the PWNS method. More specifically, such container should provide fast implementation of the following operations: one is the insertion of a particle into the container, the other is the query of all the neighbours of a particle inside the container. In the following, we will detail how to construct the container $\mathcal{T}$ as well as how to perform those insert and query operations efficiently in $\mathcal{T}$.

### 3.1 2D implementation

In the plane sweep algorithm, the $x$-axis is converted into the time domain. The dimension of the container is one less than that of the entire system, i.e., a 1D container $\mathcal{T}$ is needed for 2D implementation. For constructing $\mathcal{T}$, there are numerous candidates in the literature and most of them have tree structure such as the segment tree, the interval tree and the priority search tree etc. In this work, we choose the segment tree due to its highly efficient performance of insert and query operations for continuous integer data. The segment tree is a kind of balanced binary tree used to store segments [34]. The root of a segment tree is usually constructed by giving it an segment representing the entire domain. Then the children get their segments by dividing the parent's segment from the middle and a node is regarded as a leaf node if its segment cannot be divided any more. A simple implementation of segment tree is shown in Fig. 3. In the PWNS method, the root node of $\mathcal{T}$ is used to represent the entire range of particle coordinates and the particles are stored in the leaf nodes. $\mathcal{T}$ is constructed in keeping with $y$-coordinates of particles, i.e., the location of each particle in the tree is only determined by its $y$-coordinate. The insert operation can be implemented as shown in Algorithm 2. In order to obtain all the neighbours of particle $i$, one needs to traverse $\mathcal{T}$ from the root in a top-down manner

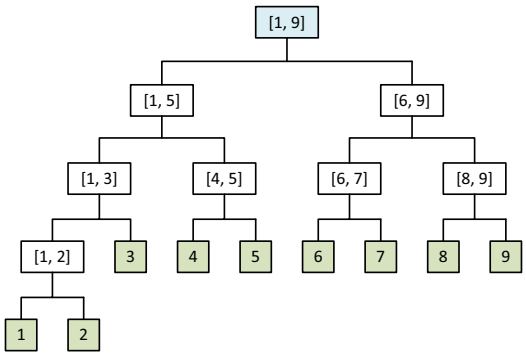



Figure 3: A 1D segment tree representing the segment $[1,9]$. The two children of a given node $[a,b]$, where $a$ and $b$ are integers, are defined as $[a,\lfloor\frac{a+b}{2}\rfloor]$ and $[\lfloor\frac{a+b}{2}\rfloor+1,b]$. Note that the floor function is used to determine the boundary of the child node.

**Algorithm 2:** The 2D insert algorithm.

**Input** : The container tree $\mathcal{T}$, particle $i$, index $k$ of $e_i^{\text{insert}}$

**Function** `INSERT(`$\mathcal{T}$`,i,k)`

- $\nu \leftarrow$ the root node of $\mathcal{T}$;
- `/* Optimization discussed in Section 3.2 */`
- $t_\nu^{\text{penult}} \leftarrow t_\nu^{\text{last}}$;
- $t_\nu^{\text{last}} \leftarrow k$;
- **if** *R is not leaf* **then**
  - **if** $y_i$ *locates in the left part of* $\nu$ **then**
    - `INSERT(`$\mathcal{T}^{\text{left}}$`,i,k);` `// `$\mathcal{T}^{\text{left}}$`: the left subtree of` $\nu$
  - **else**
    - `INSERT(`$\mathcal{T}^{\text{right}}$`,i,k);` `// `$\mathcal{T}^{\text{right}}$`: the right subtree of` $\nu$
  - **end if**
- **else**
  - Insert $i$ into $\nu$;
- **end if**

**end**

according to the following two criteria as already shown in Fig. 2:

- *Vertical Criterion (VC)*: Only those nodes with their segments overlapping the vertical projection of the support domain $[y_i-b_i,y_i+b_i]$ are traversed and the valid leaf nodes in $[y_i-b_i,y_i+b_i]$ should be chosen.
- *Horizontal Criterion (HC)*: Only the particles stored in valid leaf nodes with $e^{\text{insert}}$ greater than $e_i^{\text{limit}}$ are selected as the neighbours of particle $i$.

**Algorithm 3:** The 2D query algorithm.

**Input** : The container tree $\mathcal{T}$, particle $i$, index $k$ of $e_i^{\text{limit}}$
**Output**: The neighbour list of $i$

```
1  Function QUERY(T,i,k)
2      v ← the root node of T;
3      if t_v^last ⩾ k then
4          if v is leaf then
5              foreach particle j in v do
6                  t ← the index of e_j^insert;
7                  if t ⩾ k then
8                      Add j to the neighbour list of i;
9                  end if
10             end foreach
11         end if
           /* Optimization discussed in Section 3.2                    */
12         if t_v^penult < k then
13             QUERY(T^leaf,i,k);          // T^leaf: the only leaf meets HC
14         end if
15         s ← the support domain of i;
16         if s intersects with the left part of v then
17             QUERY(T^left,i,k);
18         end if
19         if s intersects with the right part of v then
20             QUERY(T^right,i,k);
21         end if
22     end if
23 end
```

The algorithm of the query operation is given in Algorithm 3. It should be noted that the underlined operations in Algorithms 2 and 3 are optimization techniques which will be described in Section 3.2.

## 3.2 Optimization techniques

As mentioned above, there are two criteria, i.e., $VC$ and $HC$, for eliminating unqualified candidates in the container $\mathcal{T}$ during the query process. However, it will be inefficient if only these basic criteria are implemented. Since the structure of the segment tree is predefined and only depends on $y$-coordinates, the leaves in $[y_i-b_i,y_i+b_i]$ will actually cover the entire $x$-domain $[x_{\min},x_{\max}]$ (see Fig. 4). But in practice, the support domain of a SPH particle is very small compared with the entire domain, i.e., $2a_i \ll x_{\max}-x_{\min}$. Therefore,

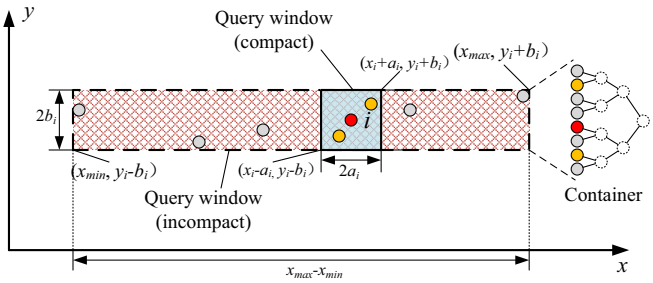



Figure 4: Comparison of the compact query window (support domain) with the incompact query window. If the incompact query window is adopted, those particles (grey) located outside $[x_i-a_i,x_i+a_i]$ will also be visited, leading to a significant increase of query time.

if the query window is not compact, most of the visited leaves satisfying $VC$ may not meet $HC$, which significantly increases the computing cost. To avoid these unnecessary traversals, several optimization techniques below are adopted.

### 3.2.1 Horizontal criterion ($HC$) combined traversal

The query operation should be combined with checking $HC$ when visiting any node in the container $\mathcal{T}$. This is done by adding a time flag $t^{\text{last}}$ to each node of $\mathcal{T}$. For a given node $\nu$, $t_\nu^{\text{last}}$ records the time of the last insertion via $\nu$. If a particle is inserted at the time $k$, the flags of all nodes on the path from the root to the leaf will be marked with $k$ (see Fig. 5). That is, $t_\nu^{\text{last}}=k$ simply indicates that all the particles in the subtree $\mathcal{T}_\nu$ are inserted either before or at the time $k$. Therefore, one should examine the flag $t_\nu^{\text{last}}$ before checking the intersection between the support domain and the node $\nu$. If $t_\nu^{\text{last}}$ is smaller than $e^{\text{limit}}$, which means no particle in the subtree $\mathcal{T}_\nu$ satisfies $HC$, then $\mathcal{T}_\nu$ can be bypassed.

The basic $HC$ combined optimization limits the query traversal to the qualified leaves. Compared with all the leaves in the entire query range, the number of these qualified leaves are quite small. Usually, there is only one leaf $\lambda$ satisfying $HC$ in a certain subtree $\mathcal{T}_\nu$. If both the root $\nu$ of such subtree $\mathcal{T}_\nu$ and the particular path from $\nu$ to $\lambda$ in $\mathcal{T}_\nu$ are determined, then we can jump straightforwardly from $\nu$ to $\lambda$, i.e., there is no need to visit those midway nodes. To this end, we introduce to each node an additional time flag $t^{\text{penult}}$ which records the time of the penultimate insertion. Then it can be easily checked that the conditions $t_\nu^{\text{last}}\geqslant e^{\text{limit}}$ and $t_\nu^{\text{penult}}<e^{\text{limit}}$ together implies that there is only one leaf $\lambda$ in $\mathcal{T}_\nu$ satisfying both $HC$ and $t_\lambda^{\text{last}}=t_\nu^{\text{last}}$. Generally, selecting $\lambda$ from all the leaves of $\mathcal{T}_\nu$ is not an easy task. Fortunately, when inserting a particle to its position (leaf), we can record the path from the top root to the target leaf at the same time (see Fig. 5). Since $\nu$ is on this path, the position of $\lambda$ is known to $\nu$ and then the expected jump can be readily completed. Fig. 6 shows a cartoon of the range query operation for valid leaves. There we can see clearly that, with the help of the $HC$ combined traversal, all of the unnecessary traversals during the query can be completely avoided.

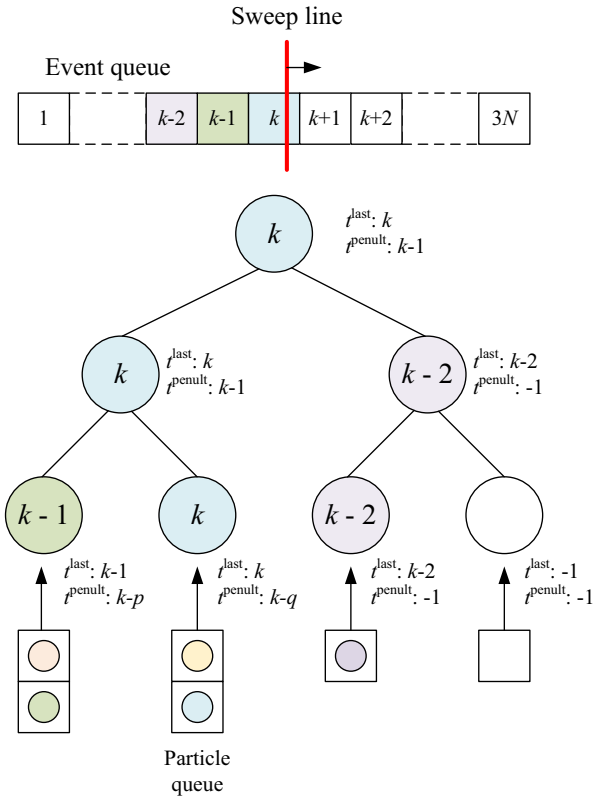



Figure 5: Insertion of a particle at event $e_k$ into the tree. The initial state of the time flag for any node is $-1$. During the insertion, the flags of all the nodes along the path from the root to the leaf are marked with $k$. Meanwhile, the penultimate time flag will also be updated. If the particle queue of the current leaf is not empty, the new particle will be added to the tail of the queue.

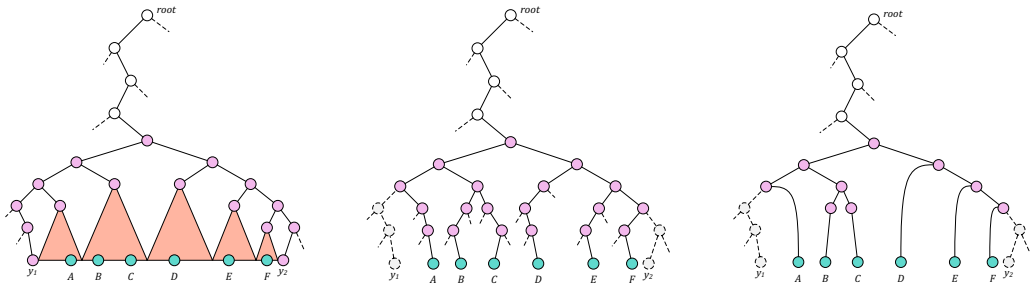



Figure 6: The range query operation for valid leaves $A$, $B$, $C$, $D$, $E$ and $F$ in $[y_1,y_2]$. (a) Range query without any optimization: the entire subtree between $y_1$ and $y_2$ will be traversed; (b) $HC$-combined traversal without jump: only those branches to valid leaves will be traversed; (c) $HC$-combined traversal with jump: If there are only one leaf satisfying $HC$ in a subtree, we can jump from the root of this subtree to the target leaf.

### 3.2.2 Indexing particles

Another optimization is for efficient implementation of the data structure. On one hand, it has been known that computers process floats more slowly than integers. On the other hand, both the plane sweep algorithm and the operations on the segment tree function efficiently with integers. Thus we usually map the coordinates of particles and support domains into integer indices while preserving the localities. This map can be achieved by the following steps:

1. In each dimension, sort the particle coordinates in ascending order and mark them by sequential indices $1,2,3\cdots,M$. Those identical coordinates are represented by the same integer, which makes $M\leqslant N$.

2. Using the particle coordinates as keys, assign each coordinate of the support domain with the index of its closest key.

The resulting integer index set is a surjection of the particle set and then the segment tree constructed from the index set will not have any empty leaves (i.e., leaves containing no SPH particles) which also results in a save of memory. Actually, such indexing procedure can be regarded as a specified ranked list technique and the interested readers are referred to [35, 36] for more discussions on other ranked list techniques.

## 3.3 3D implementation

Intuitively, we have at least two way to implement the 3D PWNS method. One is using a plane parallel to the $yz$-plane to sweep the entire domain along the $x$-axis and make window queries on the $yz$-plane, as shown in Fig. 7. We denote this kind of implementation by $\text{PWNS}_{1+2}$, meaning we perform the sweep on ONE dimension and make range queries on the remaining TWO dimensions. The other one is performing the sweep procedure with a line perpendicular to the $xy$-plane along a certain path on the $xy$-plane and doing range queries on an 1D container in the $z$-axis like we have done for 2D systems. Similarly, we denote this method by $\text{PWNS}_{2+1}$.

Actually, $\text{PWNS}_{1+2}$ is a straightforward extension of the 2D PWNS method into 3D systems. This extension can be implemented without major modifications to Algorithm 1. A slight difference is that we now need a 2D container which should support efficient insert and window query operations in the $yz$-plane. Usually, any type of spatial partitioning tree fits the job, such as the quadtree [37], the k-d tree [38] etc. In our study, two types of containers, the quadtree and the BUBtree [39], are chosen. The BUBtree is a variant of the UBtree [40], i.e., a combination of both B-tree [41] and the Z-order curve, which has been adopted in many commercial databases. Note in passing that the optimization techniques discussed in Section 3.2 can also be applied to these data structures. However, the complexity of the window query on these 2D containers cannot be guaranteed to be logarithmic [27].

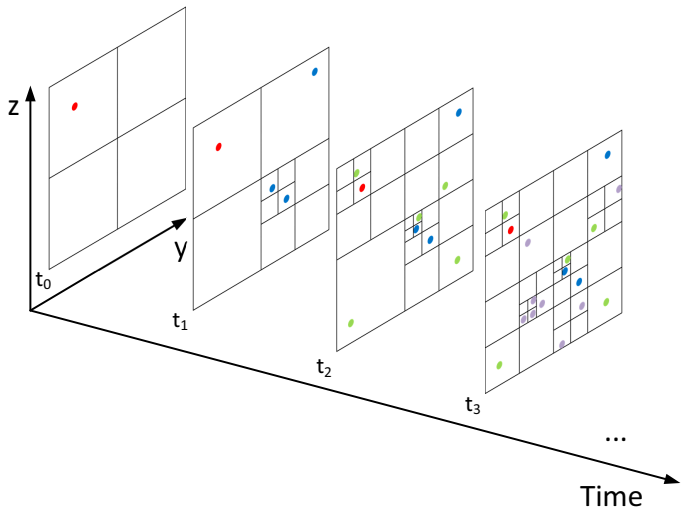



Figure 7: The quad-tree partitioning of the $yz$-plane with particles inserted chronologically. The red, blue, green, purple particles are inserted into the tree at $t_0$, $t_1$, $t_2$, $t_3$ respectively. The domain is subdivided until each leaf of the quad tree contains at most a single particle.

While in PWNS$_{2+1}$, the query operations can work in a logarithmic manner due to the 1D container (e.g., the segment tree). To perform the sweep along a certain path on the $xy$-plane, we employ the Morton code [33] to map the coordinates of particles in the $xy$-plane to 1D indices. As illustrated in Fig. 8, the Morton code is a 1D representation

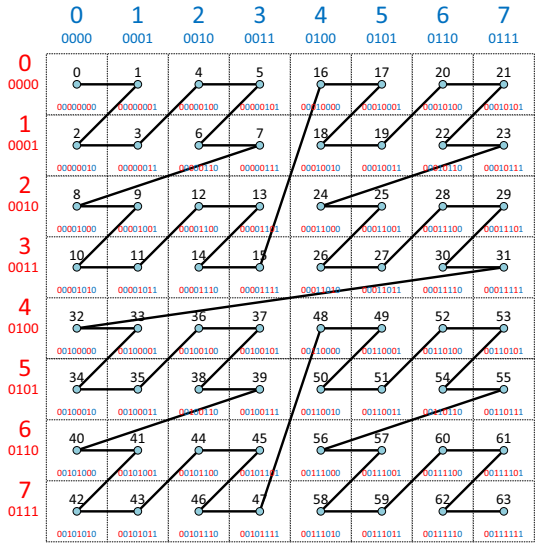



Figure 8: The Morton code map for particles with integer indices from 0 to 7.

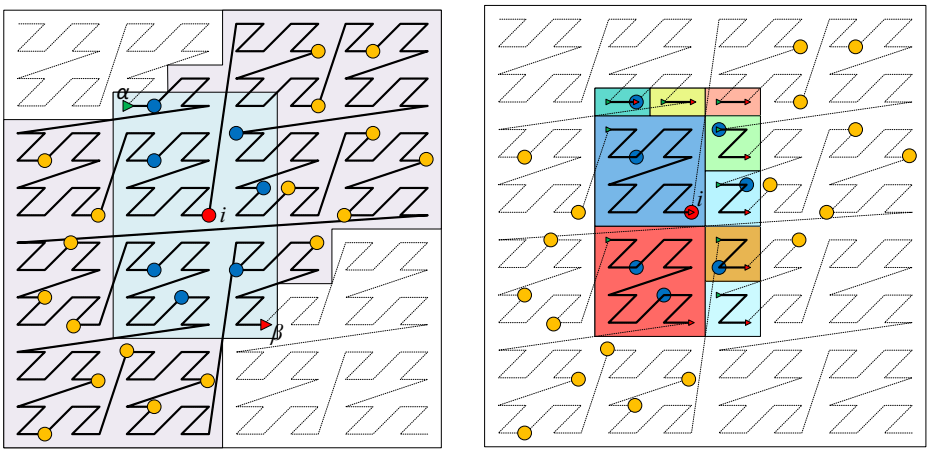



Figure 9: Comparison of $\mathcal{S}_i$ without decomposition and with BIGMIN decomposition. (a) Without decomposition: Plenty of particles (in the shadow area) outside the projection of the support domain (blue) are also cover by $\mathcal{S}_i$; (b) With BIGMIN decomposition: The resulting subsegments (after a depth-5 decomposition) with green (resp. red) arrow representing $\mathcal{M}_{\text{BIGMIN}}$ (resp. $\mathcal{M}_{\text{LITMAX}}$) only cover those particles inside the projection of the support domain.

of 2D data by interleaving the binary values of coordinates in each dimension. Connecting all the codes leads to the well known Z-order curve. An important property of the Z-order curve is that although the points on the curve are mapped into 1D indices, their localities are well preserved. The distances between a given particle and its neighbours are then close on the curve. Another crucial feature is that the order of the Morton code implicitly reflects the sequence of traversing quadtree leaves. Given two particles $i$ and $j$ whose Morton code are respectively $\mathcal{M}_i$ and $\mathcal{M}_j$, $\mathcal{M}_i<\mathcal{M}_j$ simply means we have at least $x_i<x_j$ or $y_i<y_j$. Accordingly, any particle $k$ in the support domain bounded by $(x_{\min},y_{\min})$ and $(x_{\max},y_{\max})$ will definitely locate between $\mathcal{M}_{\min}$ and $\mathcal{M}_{\max}$ on the curve. Then we can project the support domain of particle $i$ to the $xy$-plane and perform the plane sweep along the Z-order curve segment $\mathcal{S}_i=[\mathcal{M}_\alpha,\mathcal{M}_\beta]$, where the index $\alpha$ (resp. $\beta$) corresponds to the point $(x_i-a_i,y_i-b_i)$ (resp. $(x_i+a_i,y_i+b_i)$). Those particles located inside the support domain will be visited during the sweep on $\mathcal{S}_i$. Correspondingly, the events become $e_i^{\text{limit}}=\mathcal{M}_\alpha$, $e_i^{\text{insert}}=\mathcal{M}_i$ and $e_i^{\text{query}}=\mathcal{M}_\beta$. Note in passing that, the segment tree and all related optimization techniques adopted for the 2D PWNS method can still be used in $\text{PWNS}_{2+1}$. Since the query range is considerably small compared with the size of the segment tree, a logarithmic complexity for query operations can be guaranteed. However, the Morton code only preserves the information of particle position "locally". If the projection of the support domain of particle $i$ spans many local areas and grows "global", the region covered by $\mathcal{S}_i$ may become far larger than the projection, as illustrated in Fig. 9(a). We can see clearly there that the majority of particles with $\mathcal{M}$ in $[\mathcal{M}_\alpha,\mathcal{M}_\beta]$ are actually outside the support domain and will be queried when the query

event is touched by the sweep line from Algorithm 1. Even worse, these particles completely satisfy $VC$ and $HC$ and will be added to the neighbour lists. Once the number of these particles becomes too large, the cost of query operations in the container will also grow considerably. To solve this problem, the BIGMIN method [42] is adopted to decompose the puffy $\mathcal{S}_i$ into "local" subsegments without sweep line walking cross the border of the support domain. After the decomposition, the original puffy $\mathcal{S}_i$ is divided into many subsegments (shown in Fig. 9(b)), each subsegment with a $\mathcal{M}_{\text{BIGMIN}}$ being $e^{\text{limit}}$ and a $\mathcal{M}_{\text{LITMAX}}$ being $e^{\text{query}}$, where $\mathcal{M}_{\text{BIGMIN}}$ (resp. $\mathcal{M}_{\text{LITMAX}}$) denotes the point where the curve walks into (resp. outside) the support domain. By performing the sweep procedure on these subsegments, we have the skeleton of our 3D PWNS method shown in Algorithm 4.

**Algorithm 4:** The PWNS$_{2+1}$ algorithm.

**Input** : The number of SPH particles $N$, the particle set $\{i \,|\, 1 \leqslant i \leqslant N\}$
**Output**: The neighbour list of each particle

Generate Morton codes for $(x_i,y_i)$, $(x_i-a_i,y_i-b_i)$ and $(x_i+a_i,y_i+b_i)$, $i=1,2,\cdots,3N$;
**for** $i=1,2,\cdots,N$ **do**
    Divide $[\mathcal{M}_\alpha,\mathcal{M}_\beta]$ into subsegments by the BIGMIN method;
    Add all resulting $\mathcal{M}_{\text{BIGMIN}}$ and $\mathcal{M}_{\text{LITMAX}}$ to the event set $E$;
**end for**
Sort all the events in $E$;
Initialize $\mathcal{T}$;
**foreach** *event* $e_k$ *in* $E$ **do**
    **if** $e_k = e_i^{\text{insert}}$ **then**
        Insert $i$ into $\mathcal{T}$;
    **end if**
    **if** $e_k = e_i^{\text{query}}$ **then**
        Query all the neighbours of $i$ inside $\mathcal{T}$;
    **end if**
**end foreach**

As shown in Algorithm 4, some pre-operations, i.e., the Morton code generation and the Z-order curve decomposition, must be conducted in PWNS$_{2+1}$. The Morton code generation can be finished by a fast look-up table (LUT) method with only marginal costs for the coordinates of particles have already been mapped into integer indices. The situation for the BIGMIN decomposition is a bit more complicated. Since the Morton code is a $N^2$ mesh-like representation of the 2D domain with only $N$ particles in the domain, the distribution of the particles is considerably sparse. Hence, if the resulting Z-order curve subsegments become too small, there may exist a large number of subsegments covering no particles, leading to a significant increase in the runtime of event sorting and sweeping. Note that dividing any $\mathcal{S}_i$ by the BIGMIN method is in fact a binary space

partitioning, and we find that the decomposition depth of 4 or 5 is enough for most cases (see Appendix A for more details).

### 3.4 Complexity analysis

The time complexity is usually the major concern when we evaluate an algorithm. In this section, the complexity of the PWNS method will be studied, provided that there are $N$ particles in total. As shown in Algorithms 1 and 4, the search procedure can be divided into two stages: *Preparation stage* and *Sweep stage*. We will analyse below the complexity of these two stages separately.

At the beginning of *Preparation stage*, the coordinates of all the particles are mapped into integer indices. Since the indices are obtained from sorted coordinates, the complexity of this procedure can be achieved in $\mathcal{O}(N\log N)$ by any of the comparison sort methods, such as the quicksort algorithm [43]. For the 3D case, extra work for the Morton code generation must be conducted based on these indices. This can be achieved in $\mathcal{O}(N)$ with the LUT method. Afterwards, the event set is obtained. Since each particle has three basic events, i.e., $e^{\text{limit}}$, $e^{\text{insert}}$ and $e^{\text{query}}$, the complexity for constructing events will be $\mathcal{O}(3N)$. For the 3D implementation, extra operations for decomposing the event ranges will be conducted. The decomposition can be achieved in $\mathcal{O}(D\log D)$ with $D$ being the bit length of the Morton code. Since the depth of decomposition is bounded in our PWNS method, only part of the bits will be used, which reduces the complexity of decomposition to a constant level. Thus for all the particles, the decomposition can be done in $\mathcal{O}(C_{\text{BIGMIN}}N)$ and the constant $C_{\text{BIGMIN}} < D\log D$. To sort the events with integer time flags, an $\mathcal{O}(N)$ non-comparison sorting algorithm can be used, such as the bucket sort and the radix sort [44]. The last step of *Preparation stage* is constructing the container tree. Since the structure of the segment tree is known once the integer indices are obtained, the construction can be done in a bottom-up manner with a cost of $\mathcal{O}(2N)$ (almost equals to $2N$). It must be emphasized here that this $\mathcal{O}(2N)$ construction protects the PWNS method from the complex construction and maintenance of a dynamical tree structure, such as the kd-tree, the R-tree, etc. (the cost for these dynamical trees is usually $\mathcal{O}(N\log N)$), especially when moving a particle in an upper layer, which often requires updating the entire subtree. As a result, we have the total cost of *Preparation stage* to be $\mathcal{O}(N\log N)$.

In *Sweep stage*, one needs to perform $N$ insert operations in the container tree. Since each insert operation will cost $\mathcal{O}(\log N)$ time, the total cost of all insert operations is $\mathcal{O}(N\log N)$. The derivation of the complexity of query operations is slightly more complicated than insert operations. Usually, when performing one range query on a segment tree, the complexity can be $\mathcal{O}(\log N+K)$, where $K$ is the number of leaves in the range. As mentioned in Section 3.2, without optimizations, there may be quite a lot of "unnecessary" traversals in the PWNS method and $K$ can be far larger than the actual number of neighbours, leading to a waste of CPU time. However, if those optimizations are adopted, taking the 2D implementation as an example, for any particle $i$, $K$ can be

reduced from $\frac{2b_i}{A_y}N$ to $\frac{2a_i}{A_x}\frac{2b_i}{A_y}N$, where $A_x$ (resp. $A_y$) are the projecting length of the entire domain in $x$-axis (resp. $y$-axis) (See Fig. 4). Since $a_i$ is usually far less than $A_x$, the proposed optimization techniques can significantly reduce the time costs of query operations. Furthermore, as the number of neighbours of each particle is approximately constant [9], we have $\frac{2a_i}{A_x}\frac{2b_i}{A_y}N \sim k$, where $k$ is the number of neighbours. Meanwhile, the traversal on the segment tree can jump from a certain level to the leaf directly. The skipped depth can be described as $\log\frac{K}{k}=\log\frac{A_x}{2a_i}$, where $\frac{K}{k}$ represents the width of query range per neighbour on the tree and $K \gg k$. Consequently, we have a profound reduction in the query depth from $\log N$ to $\log N-\log\frac{K}{k}=\log\frac{kN}{K}$ and the complexity of query operations becomes $\mathcal{O}(N\log\frac{kN}{K}+kN)$. Totaling the complexity terms of both insert and query operations up, the cost of *Sweep stage* is also $\mathcal{O}(N\log N)$.

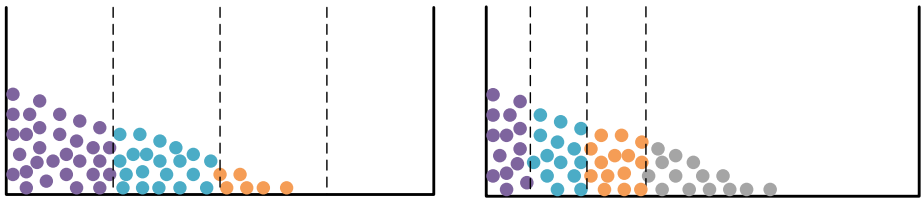
In summary, the complexity of the PWNS method can be guaranteed to be $\mathcal{O}(N\log N)$. In actual simulations, multiple sweeps over the same set of particles are usually unavoidable and we are able to make further improvement in these multiple sweeps by virture of a Verlet list [45]. Actually, by slightly enlarging the support domain of each particle, one could use the PWNS method to generate a Verlet list and then reduce the complexity of neighbour search in each SPH iteration to $\mathcal{O}(kN)$. As a result, the PWNS method could be performed every dozens of SPH iterations when the Verlet list is required to be updated, and thus the complexity of the entire SPH simulation is further reduced.

## 4 Parallel implementation

The CFD problems usually involve large-scale data processing. The capability of dividing a large problem into small parts and solving them simultaneously is a basic requirement for modern CFD solvers. Based on the plane sweep algorithm, the parallelization of the PWNS method is equivalent to sweeping over all the SPH particles synchronously with multiple lines. Therefore the resulting SPH solver can thoroughly be parallelized.

Taking the 2D case for instance, in the PWNS SPH solver, the $x$-coordinates of SPH particles are sorted in ascending order and converted to integer indices, from which a temporal event queue is constructed. Then a line perpendicular to the $x$-axis is used to sweep the entire domain by scanning the chronologically ordered events. Once the line has passed over all the events, the complete neighbour list is obtained and then the inter-particle forces can be calculated. Finally, the related physical information of all SPH particles is updated.

To implement the parallel PWNS method, the particle set is first divided into $n$ subsets based on the indices of particles. Then for each subset, we can construct its own event queue and perform independent PWNS. Finally, each subset will have its own neighbour list and the following processes can be conducted within this parallelism framework. Note that unlike those geometry-dominant parallel algorithms (see Fig. 10(a)), our index-dominant dividing is practically not restricted by the geometry of the system. Since the sweep operation only depends on the well ordered index array $\{1,2,\cdots,N\}$, where $N$



(a) Geometry-dominant dividing (b) Index-dominant dividing

Figure 10: Comparison of two dividing methods. (a) Geometry-dominant dividing, where the space is equally divided while the number of particles in each subset varies; (b) Index-dominant dividing, where the number of particles in each subset is almost equal, i.e. the computing cost on each node is balanced.

is the number of particles, we can simply divide the particles into $n$ balanced sets as $\{\{(i-1)\frac{N}{n}+1,(i-1)\frac{N}{n}+2,\cdots,i\frac{N}{n}\}|i=1,2,\cdots n\}$ without considering the geometrical distribution of SPH particles, as shown in Fig. 10(b). This is yet another demonstration of the purely meshfree property of the PWNS method. Because the cost of the PWNS method is dominated by the number of SPH particles, the costs of each subsets are almost the same. Therefore, load balancing can be easily achieved in our parallel PWNS SPH solver.

While implementing the parallel PWNS method, several efficiency issues should be remarked below.

1. With the aim of achieving sound load balancing and reducing communication costs in most computing processes, all the particles in the global set must be sorted before dividing into subsets. To this end, a parallel sort algorithm based on regular sampling [46] is adopted here. Although this algorithm requires unavoidable communications between processors, experimental results have shown its encouraging speedup on present multiprocessor architectures [46].

2. When constructing the event queue, the edges of each support domain are also included. In consequence, as illustrated in Fig. 11(a), if the support domain of particle $i$ in Subset A wraps other particles in Subset B, those particles in Subset B will not be included in the event queue of Subset A. Thus the plane sweep in Subset A will not give the neighbours of $i$ in Subset B and leads to an incomplete neighbour list. To fix this problem, a ghost support domain of particle $i$ is introduced in Subset B (see Fig. 11(b)) and then the event queue of Subset B will take account of the ghost domain, i.e., $e_i^{\text{limit}}$ and $e_i^{\text{query}}$ will effect during the sweep of Subset B. Finally, the neighbours of particle $i$ in Subset B can be found and will be added to the neighbour list of Subset A before the force calculation. Compared with the entire system, the amount of particles with support domain across subsets is almost negligible in large-scale simulations, thus the communication cost of neighbour transfer between different subsets is marginal.

3. As discussed in Section 3.4, the query depth can be reduced from $\log N$ to $\log\frac{kN}{K}$ once the optimization techniques introduced in Section 3.2.1 are applied. Accord-

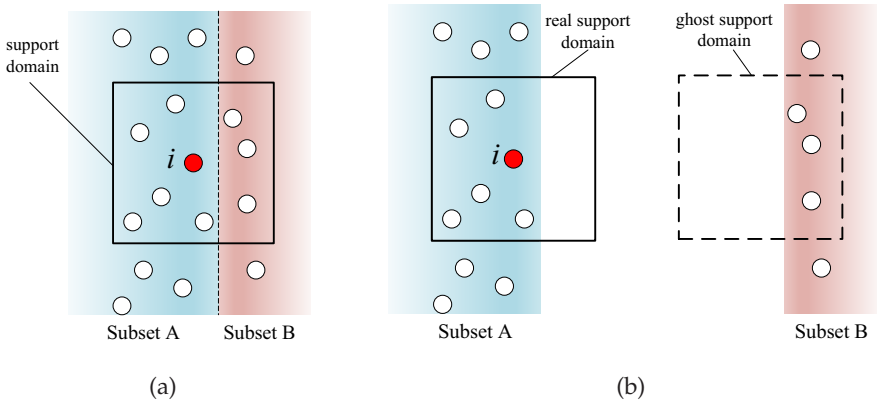



Figure 11: (a) The support domain of particle $i$ in Subset A wraps other particles in Subset B. (b) A ghost support domain of particle $i$ is introduced to find the neighbours within Subset B.

ingly, for any $p$-thread parallel implementation, the resulting query depth is $\log\frac{N'}{p}$ with $N'=\frac{kN}{K}$. Since $N'$ is apparently smaller than $N$, the descending rate of $\log\frac{N'}{p}$ will be larger than $\log\frac{N}{p}$ as $p$ grows. This will result in a *superlinear* decrease of the complexity for the parallel implementation. This phenomenon is especially common for randomly distributed particles, where the neighbours of a given particle are more sparsely scattered than uniformly distributed cases.

## 5 Numerical experiments

In this section, we present several test cases to evaluate the PWNS method which was implemented in C++ with OpenMP as the parallelization platform. All the cases were run on a server with 32 processors (2.67GHz Intel Xeon E7-8837).

### 5.1 Efficiency evaluation

To evaluate the efficiency of the PWNS method, we conducted two groups of experiments with different particle distributions. In the first group, the particles are placed in a $n^d$ uniform Cartesian mesh, where $d$ is the space dimension. Thus the number of particles $N=n^d$. This type of distribution is popularly used as the initial particle distribution in SPH simulations. Here the initial particle spacing $\Delta x=0.01$ and the smoothing length $\kappa h=0.03$. A sample 2D distribution with $20\times 20$ particles is shown in Fig. 12(a). In the second group, the particles are randomly distributed in the same domain (see Fig. 12(b)). In order to mimic the practical SPH simulations, the smoothing length is allowed to vary and obeys a uniform distribution $\mathcal{U}(0.8\kappa h,1.2\kappa h)$. Eventually, the evaluations of both the

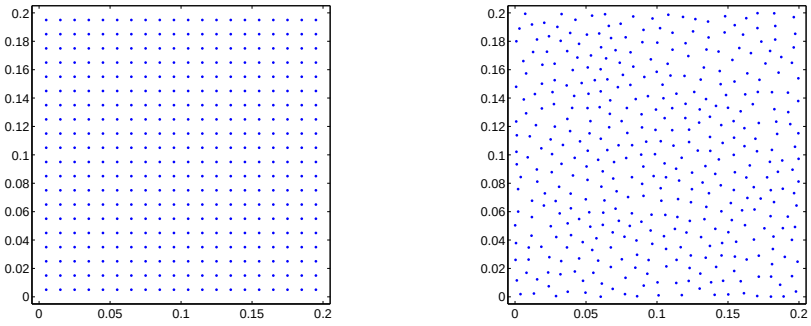
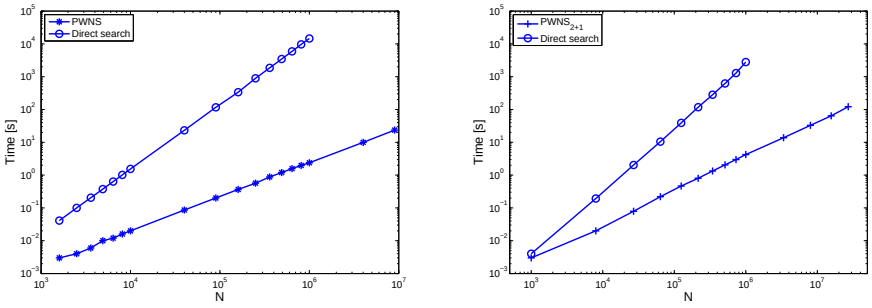



Figure 12: Two types of 2D particle distributions (400 particles). (a) Particles placed uniformly; (b) Particles placed randomly.

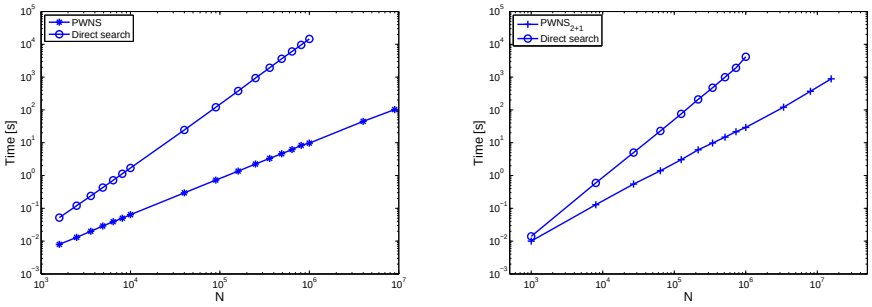
initial and run-time particle distributions are covered. Note that PWNS$_{2+1}$ is used to evaluate the 3D performance here. All the experiments are run on a single processor, which usually results in an unacceptable time consumption for the brutal $\mathcal{O}(N^2)$ neighbour search. Figs. 13 and 14 show the runtime for both direct search and the PWNS method as the number of particles increases. We can clearly observe there that a notable reduction in the runtime is achieved by the PWNS method. The time consumption of the PWNS method for the random case is slightly slower than the uniform cases. This is because the container tree is constructed from the particle indices. For example, for the 2D case, the uniform-distribution with $N$ particles yields only $\sqrt{N}$ different $y$-coordinates, i.e., the container tree only has $\sqrt{N}$ leaf nodes. However, for an equal-scale random-distribution, there would be $N$ $y$-coordinates, leading to an apparently larger container tree. Fortunately, the worst time complexity of the random case will only be a constant times that of the uniform case for the cost of query operations in the container tree is $\mathcal{O}(\log N+k)$.

As mentioned in Section 3.3, there are two ways to implement the 3D PWNS method, i.e., PWNS$_{2+1}$ and PWNS$_{1+2}$. Better performances of PWNS$_{2+1}$ over the direct search is clearly shown in Figs. 13(b) and 14(b). To compare the performance of these two ways, we repeated the same test cases aforementioned with PWNS$_{1+2}$. Two types of containers, i.e., the quadtree and the BUBtree, were both tested in PWNS$_{1+2}$. Fig. 15 shows the performance of PWNS$_{2+1}$, PWNS$_{1+2}$ with the quadtree and PWNS$_{1+2}$ with the BUBtree in *Sweep stage*. We can see there that PWNS$_{2+1}$ works apparently better than the other two methods, especially when the number of particles grows large. As aforementioned, for PWNS$_{1+2}$, the window query operations are needed on 2D containers (e.g., the quadtree and the BUBtree). But it is addressed in Section 1 that the 2D window query often encounters a worse complexity than the range query on 1D containers (e.g., the segment tree), which deteriorates as the data sets grow. Thus the $\mathcal{O}(N\log N)$ complexity cannot be guaranteed in PWNS$_{1+2}$. In view of this, we will choose PWNS$_{2+1}$ to implement the 3D PWNS SPH solver in practice.




(a) (b)

Figure 13: Comparison of the performance of uniform-distribution case between the direct-traverse neighbour search and the PWNS method. (a) 2D implementation; (b) 3D implementation.




(a) (b)

Figure 14: Comparison of the performance of random-distribution case between the direct-traverse neighbour search and the PWNS method. (a) 2D implementation; (b) 3D implementation.

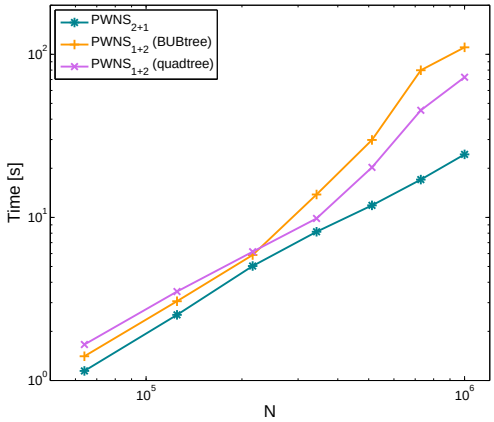
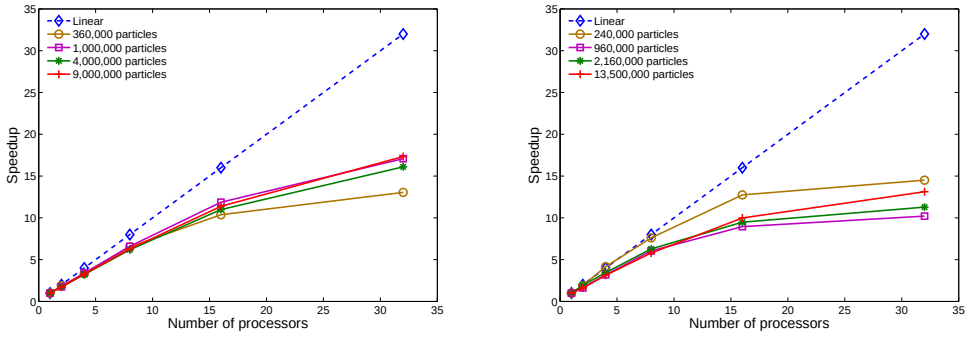



Figure 15: 3D PWNS performance of $\mathrm{PWNS}_{2+1}$, $\mathrm{PWNS}_{1+2}$ (quadtree) and $\mathrm{PWNS}_{1+2}$ (BUBtree) in *Sweep stage*.

## 5.2 Parallel performance

To evaluate the parallel performance, four test cases with different number of particles were computed. The speedup $S_p = C_1/C_p$ is used as the performance metric, where $C_1$ and $C_p$ are the time costs of the sequential code and the parallel code run on $p$ processors respectively. The placement of particles is in accordance with those cases in Section 5.1.

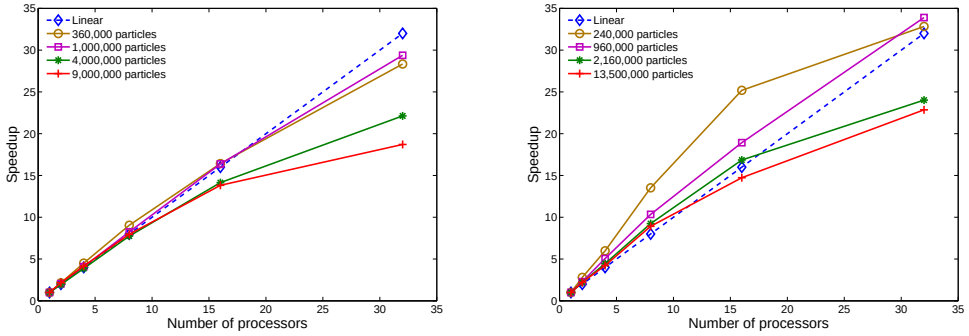
The parallel performance for the uniform distribution cases is shown in Fig. 16. Good performance with slight drop in speedup can be seen till 16 processors. The drop in speedup is a common phenomenon for MIMD (multiple instruction, multiple data) systems with a shared memory, where the efficiency of data accessing and synchronizing between CPU-caches and memory deteriorates with the increase of simultaneously-functioning processors [47]. It should be noted that, although unavoidable communications are introduced by a parallel sort as mentioned in Section 4, we still obtained an acceptable speedup.




(a) (b)

Figure 16: The parallel performance for the uniform-distribution cases. (a) 2D implementation; (b) 3D implementation.

In Fig. 17, one can see a better parallel performance for the random distribution cases. This can be attributed to the scale of the container tree. For the uniform distribution cases, the number of diverse integer indices in each subset is nearly the same as the entire system, while this number drops considerably for random cases. As a result, much smaller container trees are obtained for parallelized random distribution cases, leading to a decrease of the cost for both the insert and the query operations. In SPH simulations, the particles are in a disordered manner most of the time, thus the parallel performance of random distribution cases demonstrates that a significant speedup and load balancing for the SPH solver can be attained by our PWNS method. Also shown in Fig. 17, an obvious superlinear speedup is achieved by the PWNS method. This phenomenon is mainly due to the jump optimization technique and has been discussed in detail in Section 4. If taking the computer architecture into account, we will find that the superlinear speedup can be amplified by accumulated cache size [48]. That is, the accumulated capacity of




(a) (b)

Figure 17: The parallel performance for the random-distribution cases. (a) 2D implementation; (b) 3D implementation.

CPU-caches for multiple processors is larger than that of a single processor. Thus in parallel mode, more data can be stored in caches and accessed rapidly, which significantly reduces the traffic between memory and CPUs.

### 5.3 Example application

As discussed before, our method can work efficiently with arbitrarily distributed particles and achieve parallel load balancing without considering the geometry of the system. Here, we choose the dam break problem with flood impacting on an obstacle [49] as our example application. The geometrical parameters are slightly changed in our work. As shown in Fig. 18, the water with length $L_f = 2.4$ m, height $H_f = 1$ m and width $W_f = 1$ m is initially at rest and bounded by the tank (3.22 m $\times 1$ m $\times 1$ m) and the dam. The water is set to have density $\rho = 1000$ kg/m$^3$ and dynamic viscosity $\eta = 1\times 10^{-3}$ Pa·s. The sound speed $c$ is set to be $10\sqrt{gH_f}$. The solid wall is mimicked by the dummy-particle boundary condition [12,50]. The initial particle spacing is $\Delta x = 0.02$ m, i.e., 181364 particles are used. The time step is controlled by $\Delta t \leq \min(0.25\frac{h}{c}, 0.25\frac{h^2}{\nu}, 0.25\sqrt{\frac{h}{|f|}})$, where $f$ is the body force per mass on SPH particles. We choose the time step $\Delta t = 1\times 10^{-4}$ s. The motion equations for this dam break problem is in accordance with those used in [12,50].

When the simulation starts, the dam bounding the water is immediately removed. Thereafter, the water will flood the tank due to gravity. When the water front reaches the cuboid obstacle ($0.16m \times 0.4m \times 0.16m$), it will hit the obstacle and form a stream moving upward. Meanwhile, part of the flood will bypass the obstacle and move forward. The flow behaviour at $t = 0.4$ s and $t = 0.56$ s is shown in Fig. 19 and is in good agreement with the experimental results in [49]. The thread-distribution of the particles is clearly shown in Fig. 19 as well. The region of each thread is dynamically adapted to achieve load balancing when the shape of fluids changes.

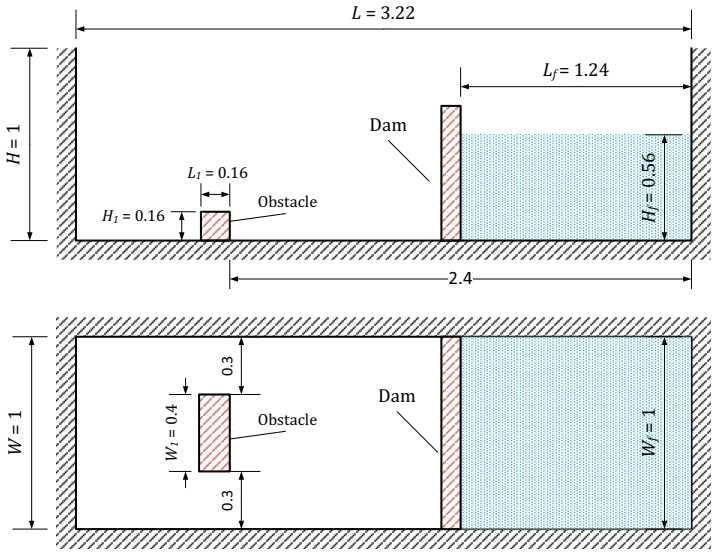
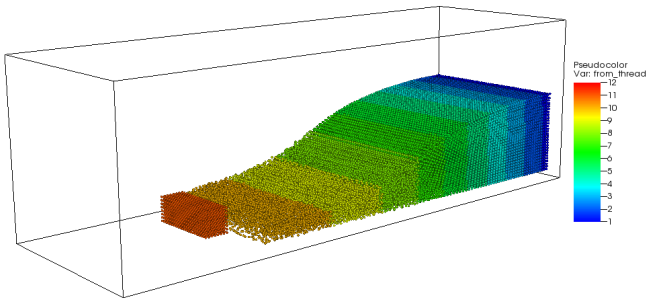
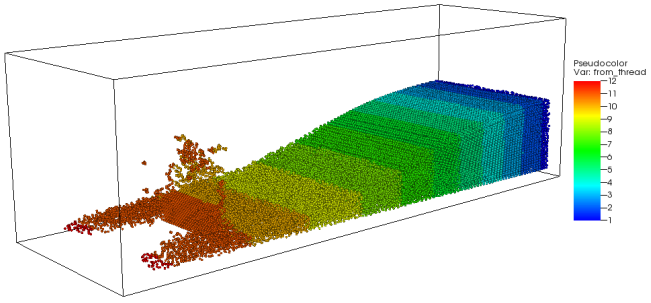



Figure 18: Schematic of the 3D dam break. Top: side view; Bottom: top view.




(a) 0.4s




(b) 0.56s

Figure 19: Snapshots of the 3D dam break at time (a) 0.4s; (b) 0.56s. Particles assigned to different threads are distinguished by their colours. A total of 12 threads are used. In order to show the behaviour of fluid clearly, only particles of the fluid and the obstacle are shown here.

## 6 Conclusions

In this work, a new $\mathcal{O}(N\log N)$ algorithm for neighbour search in SPH is proposed. This efficient method is based on a plane sweep algorithm. Compared with existing neighbour search methods, the proposed method is purely meshfree and compatible with processing variable smoothing length, arbitrarily distributed particles and irregular kernels. The plane sweep approach is based on a dimension-reduction methodology. By mapping certain axes into time indices (i.e., the direct mapping in 2D, the Z-order curve mapping in 3D), the neighbour search procedure for large amount of SPH particles can be performed in an efficient batch processing manner. Much to the knowledge of authors, this is the first plane sweep neighbour search (PWNS) method for SPH. Some optimization techniques for traversing the tree structure which reduces the cost significantly are also discussed. Furthermore, also with the help of the plane sweep idea, the system can be easily parallelized and the load balancing can also be obtained without considering the geometrical parameters. The numerical results of the serial code show that our method is able to efficiently process neighbour search in SPH. Meanwhile, the parallel performance of the PWNS method exhibits the strength for large scale problems. A successful application of the PWNS method has been presented in [12] for simulating the chemical mechanical polishing, a critical engineering problem in semiconductor manufacturing industry.

## Acknowledgments

This research was supported by grants from the National Natural Science Foundation of China (Nos. 11471025, 91330110, 11421101) and the Specialized Research Fund for the Doctoral Program of Higher Education (No. 20110001120112). W.D. acknowledges the support from Beijing International Center for Mathematical Research for his visit in the second half of 2012, during which the work on this paper is initiated.

## Appendix: BIGMIN decomposition depth

Recalling the $\text{PWNS}_{2+1}$ method in Section 3.3, we could find that the BIGMIN method is one of the key factors influencing the efficiency. That is, if the BIGMIN decomposition doesn't come to a suitable level, much of the effort will be wasted on filtering particles outside the support domain. If the BIGMIN decomposition is over performed, many Z-order curve subsegments may cover no particles, leading to an increase on the CPU time of the sweep stage. Thus, it is necessary to find a way to predict the effect of the BIGMIN decomposition, i.e., the ratio of those useful and redundant regions covered by Z-order curve segments. In this Appendix, the acceptable depth of BIGMIN decomposition used in Section 3.3 will be derived.

Before the derivation, an interesting observation on the Morton code mapping should be addressed. Usually, in 3D SPH, each particle occupies $\Delta x^3$ space. For a $L\times W\times H$ SPH system, we have the number of particle $N\approx\frac{LWH}{\Delta x^3}$. In the $\text{PWNS}_{2+1}$ method, these $N$

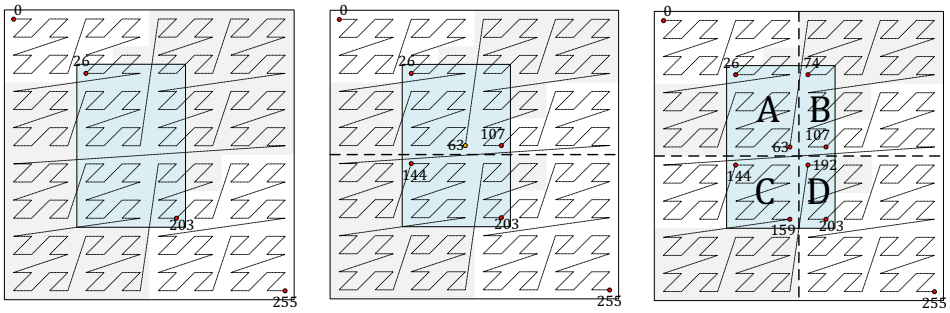
particles are mapped into an $N\times N$ lattice in $xy$-plane. The Morton code area for a typical support domain is about $Area_{\text{domain}}\approx 4\times\frac{3\Delta x}{L}N\times\frac{3\Delta x}{W}N$, where $3\Delta x$ is the common smoothing length. For ordinary cases, $L$, $W$ and $H$ are comparable with each other. Consequently, we have $Area_{\text{domain}}\propto N^{\frac{4}{3}}$ implying that the derivation will not be affected by the geometric information of the system.

As aforementioned, the Morton code implicitly reflects the visit sequence of quadtree leaves. Thus for any support domain bounded by $[\mathcal{M}_\alpha,\mathcal{M}_\beta]$, we can find a minimum envelop box $[\mathcal{M}_{\min},\mathcal{M}_{\max}]$ representing the lowest common ancestor of all Morton-code leaves located inside $[\mathcal{M}_\alpha,\mathcal{M}_\beta]$. For simplicity, we can subtract $\mathcal{M}_{\min}$ from $[\mathcal{M}_\alpha,\mathcal{M}_\beta]$ without losing the locality of those codes inside the subtree represented by $[\mathcal{M}_{\min},\mathcal{M}_{\max}]$. Then we have the normalized support domain $[\mathcal{M}_{\alpha'},\mathcal{M}_{\beta'}]$ and the normalized envelop box $[0,\mathcal{M}_{\text{upper}}]$, where $\mathcal{M}_{\alpha'}=\mathcal{M}_\alpha-\mathcal{M}_{\min}$, $\mathcal{M}_{\beta'}=\mathcal{M}_\beta-\mathcal{M}_{\min}$, $\mathcal{M}_{\text{upper}}=\mathcal{M}_{\max}-\mathcal{M}_{\min}$. For the Morton code representation of the quadtree, all the leaves can be arranged in a $2^n\times 2^n$ mesh [51], where $n$ is the depth of the quadtree. Hence, let $2^p$ and $2^q$ be the minimum power-of-two greater than $x_{\beta'}$ and $y_{\beta'}$ respectively, then it can be easily checked that $[0,\mathcal{M}_{\text{upper}}]$ with $\mathcal{M}_{\text{upper}}=2^{p+q}-1$ is the normalized minimum envelop of $[\mathcal{M}_{\alpha'},\mathcal{M}_{\beta'}]$ if and only if $x_{\alpha'}\leqslant 2^{p-1}-1\leqslant x_{\beta'}$ and $y_{\alpha'}\leqslant 2^{q-1}-1\leqslant y_{\beta'}$.

As discussed in [42], $\mathcal{M}_{\text{BIGMIN}}$ and $\mathcal{M}_{\text{LITMAX}}$ are obtained by replacing the actual bits of $\mathcal{M}_{\alpha'}$ and $\mathcal{M}_{\beta'}$ with certain bit patterns. The actual bit position is the left-most distinguishable bit (LDB) of $\mathcal{M}_{\alpha'}$ and $\mathcal{M}_{\beta'}$. An example of a depth-two BIGMIN is shown in Fig. 1. Since each bit in the Morton code exactly represents a level in a quadtree, it is implied that the lowest common ancestor of a group of Morton codes is determined by their LDB. Moreover, the bit pattern is determined by $2^{p-1}$ for $x$-direction splitting or $2^{q-1}$ for $y$-direction splitting depending on LDB. Therefore, it can be concluded that the procedure of the BIGMIN decomposition is the same as finding the normalized envelop and conducting the binary space partitioning on the normalized envelop.

Employing the properties of the normalized envelop and the BIGMIN decomposition aforementioned, we can predict the effect of the BIGMIN decomposition in a quantitative manner. For a depth-$(s+t)$ BIGMIN decomposition, the resulting normalized envelop of each sub support domain will become $[0,2^{(p-s)+(q-t)}]$ or even smaller, where $s$ (resp. $t$) denotes the depth of partitioning in the $x$-direction (resp. $y$-direction), where $s=t$ or $s=t+1$. Meanwhile, the area of the redundant regions covered by the Z-order curve segments can be calculated as well. As shown in Fig. 2, the envelops of the border part of the support domain after the BIGMIN decomposition will finally follow four patterns. Each of such border envelops consists of a useful region and a redundant region. For each pattern, the area of the redundant regions can be obtained by the equations listed in Tab. 1. If the entire support domain is evenly divided, the numbers of the occurrences of Patterns I, II, III, IV will be $2\times(2^s-2)$, $2\times(2^t-2)$, 2, 2 respectively. Eventually, we have the ratio of the redundant regions to the support domain as

$$\mathcal{R}=\frac{2\times(2^t-2)Area_I+2\times(2^s-2)Area_{II}+2Area_{III}+2Area_{IV}}{(2^s-2)(2^t-2)\mathcal{L}\mathcal{W}+2\times(2^t-2)\mathcal{L}y_0+2\times(2^s-2)\mathcal{W}x_0+4x_0y_0},$$




(a) Before BIGMIN (b) depth-1 BIGMIN (c) depth-2 BIGMIN

Figure 1: An example of the BIGMIN decomposition for a Z-order curve $[26,203]$. The centre rectangle (blue) is the useful region while the grey area represents the redundant region. The actual bit position for $26=(00011010)_2$ and $203=(11001011)_2$ is the left-most distinguishable bit (LDB), i.e., the first bit counting from the left in this case, and then the minimum envelop should be $[0,255]$ if noting $255=(11111111)_2=(100000000)_2-(000000001)_2=2^8-1$ (see (a)). Here, the LDB comes from the $y$ index through the bit interleaving, hence the $y$ part of the midpoint $63=(2^6-1)=(00111111)_2$, i.e., $7=(0111)_2$ provides the LITMAX pattern "0x1x1x1x" and the BIGMIN pattern "1x0x0x0x" (by $7+1=8=(1000)_2$ ), where "x" represents the bits unchanged. By replacing the maximum range value $203=(11001011)_2$ with the LITMAX pattern "0x1x1x1x", we have the first $\mathcal{M}_{\text{LITMAX}}=107=(01101011)_2$. In the same way, we have the first $\mathcal{M}_{\text{BIGMIN}}=144=(10010000)_2$ (see (b)). After that, we can perform the second round of BIGMIN decomposition for the Z-order curves $[26,107]$ and $[144,203]$, respectively, and obtain the subsegments A$[26,63]$, B$[74,107]$, C$[144,159]$, D$[192,203]$ (see (c)). Meanwhile, the redundant regions (gray) covered by the Z-order curve segments also become smaller.

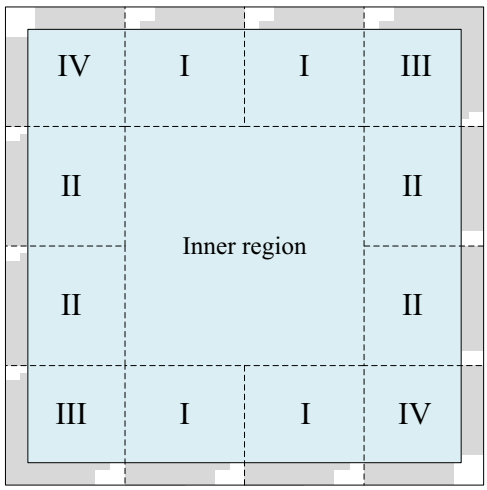
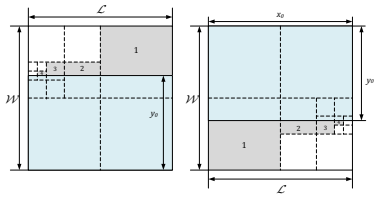
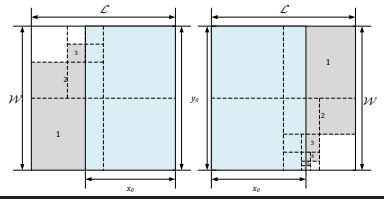
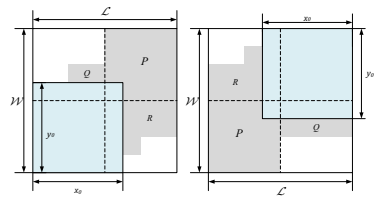
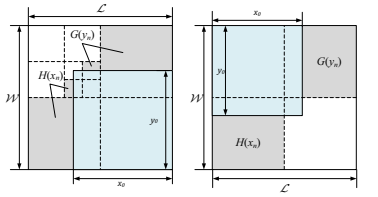



Figure 2: The envelops of the border part of the support domain (blue) after the BIGMIN decomposition will finally follow four patterns (I, II, III, IV). Each of such border envelops consists of a useful region (blue) and a redundant region (region).

where $\mathcal{L}$ and $\mathcal{W}$ are the length and the width of the normalized envelop respectively, $x_0$ and $y_0$ denote the length and the width of the useful region. If the BIGMIN decomposition is not applied, $\mathcal{R}$ could be 50% or even worse when the level of the envelop is

Table 1: The patterns of border envelops after the BIGMIN decomposition. In Pattern I, II, $F$ denotes the area of the decreasing redundant blocks (grey) labeled from 1 to $n$. In Pattern III, the redundant region can be divided into several parts labeled by $P$, $Q$, $R$. In Pattern IV, where the size of each redundant block ($G$ and $H$) shrinks sharply, we use $n'$ to label the next block smaller than the $n$th block. Note that $\mathcal{L}$ (resp. $\mathcal{W}$) is the minimum power-of-two larger than $x_0$ (resp. $y_0$).

| Pattern | | Area of the redundant region |
|---|---|---|
| I |  | $Area_I(x_0,y_0)=\sum_{0<y_n\leqslant y_1} F_{In}$<br>$F_{In}=\frac{\mathcal{L}}{2^n}(\frac{\mathcal{W}}{2^n}-y_n)$<br>$y_n=\begin{cases} y_{n-1}, & y_{n-1}\leqslant\frac{\mathcal{W}}{2^n} \\ y_{n-1}-\frac{\mathcal{W}}{2^n}, & y_{n-1}>\frac{\mathcal{W}}{2^n}\end{cases}$ |
| II |  | $Area_{II}(x_0,y_0)=\sum_{0<x_n\leqslant x_1} F_{IIn}$<br>$F_{IIn}=\frac{\mathcal{W}}{2^n}(\frac{\mathcal{L}}{2^{n-1}}-x_{n-1})$<br>$x_n=\begin{cases} x_{n-1}, & x_{n-1}<\frac{\mathcal{L}}{2^n} \\ x_{n-1}-\frac{\mathcal{L}}{2^n}, & x_{n-1}\geqslant\frac{\mathcal{L}}{2^n}\end{cases}$ |
| / III |  | $Area_{III}(x_0,y_0)=P+Q+R$<br>$P=\frac{\mathcal{L}\mathcal{W}}{4}-(x_0-\frac{\mathcal{L}}{2})(y_0-\frac{\mathcal{W}}{2})$<br>$Q=Area_I(\frac{\mathcal{L}}{2},y_0-\frac{\mathcal{W}}{2})$<br>$R=Area_{II}(x_0,\frac{\mathcal{W}}{2})$ |
| IV |  | $Area_{IV}=G(y_1)+H(x_1)$<br>$G(y_n)=\begin{cases} \frac{\mathcal{L}}{2^n}(\frac{\mathcal{W}}{2^{n-1}}-y_{n-1})+G(y_{n'}), & y_{n'}>0 \\ 0, & y_{n'}=0\end{cases}$<br>$H(x_n)=\begin{cases} \frac{\mathcal{W}}{2^n}(\frac{\mathcal{L}}{2^{n-1}}-x_{n-1})+H(x_{n'}), & x_{n'}>0 \\ 0, & x_{n'}=0\end{cases}$ |

close to the root of the quadtree. However, after a depth-4 BIGMIN decomposition, for average cases with $x_0\approx\frac{3}{4}\mathcal{L}$ and $y_0\approx\frac{3}{4}\mathcal{W}$, $\mathcal{R}$ can be reduced to about 16%. In a typical SPH system with a $3\Delta x$ smoothing length, there can be over 200 neighbours for one particle. Therefore, in the PWNS$_{2+1}$ method, a depth-4 BIGMIN will only result in $30\sim40$ redundant neighbours. Distributing these redundant particles to the bordering normalized envelops (12 for depth-4 decomposition), we have only 3 of them found per query operation, which can be eliminate with marginal $\mathcal{O}(1)$ time cost. If the depth of BIGMIN decomposition goes larger, the amount of redundant particles only decreases slightly while the number of Z-order curve subsegments will grow exponentially, which on the contrary results in a serious increase of the time cost. Accordingly, the desired depth of the BIGMIN decomposition should be 4 or 5.

**References**

[1] L. B. Lucy, A numerical approach to the testing of the fission hypothesis, Astron. J. 82 (1977) 1013–1024.
[2] R. A. Gingold, J. J. Monaghan, Smoothed particle hydrodynamics: theory and application to non-spherical stars, Mon. Not. Roy. Astron. Soc. 181 (1977) 375–389.
[3] J. J. Monaghan, Simulating free suface flows with SPH, J. Comput. Phys. 110 (1994) 399–406.
[4] J. J. Monaghan, SPH without a tensile instability, J. Comput. Phys. 159 (2000) 290–311.
[5] A. Colagrossi, M. Landrini, Numerical simulation of interfacial flows by smoothed particle hydrodynamics, J. Comput. Phys. 191 (2003) 448–475.
[6] V. Springel, L. Hernquist, Cosmological smoothed particle hydrodynamics simulations: a hybrid multiphase model for star formation, Mon. Not. Roy. Astron. Soc. 339 (2003) 289–311.
[7] X. Y. Hu, N. A. Adams, A multi-phase SPH method for macroscopic and mesoscopic flows, J. Comput. Phys. 213 (2006) 844–861.
[8] J. J. Monaghan, A turbulence model for smoothed particle hydrodynamics, Eur. J. Mech. B-Fluids 30 (2011) 360–370.
[9] D. J. Price, Smoothed particle hydrodynamics and magnetohydrodynamics, J. Comput. Phys. 231 (2012) 759–794.
[10] C. Ulrich, M. Leonardi, T. Rung, Multi-physics SPH simulation of complex marine-engineering hydrodynamic problems, Ocean Eng. 64 (2013) 109–121.
[11] P. Cleary, J. Ha, V. Alguine, T. Nguyen, Flow modelling in casting processes, Appl. Math. Model. 26 (2002) 171–190.
[12] D. Wang, S. Shao, C. Yan, W. Cai, X. Zeng, Feature-scale simulations of particulate slurry flows in chemical mechanical polishing by smoothed particle hydrodynamics, Commun. Comput. Phys. 16 (2014) 1389–1418.
[13] J. L. Bentley, A survey of techniques for fixed radius near neighbor searching, Tech. Rep. SLAC-186, Stanford University, Stanford, CA (1975).
[14] J. J. Monaghan, Particle methods for hydrodynamics, Comput. Phys. Rep. 3 (1985) 71–124.
[15] M. Gomez-Gesteira, B. D. Rogers, A. J. C. Crespo, R. A. Dalrymple, M. Narayanaswamy, J. M. Domínguez, SPHysics-development of a free-surface fluid solver-Part 2: Efficiency and test cases, Comput. Geosci. 48 (2012) 300–307.
[16] L. Hernquist, N. Katz, TreeSPH: A unification of SPH with the hierarchical tree method, Astrophys. J. Suppl. Ser. 70 (1989) 419–446.
[17] W. Benz, R. L. Bowers, A. G. W. Cameron, W. H. Press, Dynamics mass exchange in doubly degenerate binaries. I. 0.9 and 1.2 $M_{\odot}$ stars, Astron. J. 348 (1990) 647–667.
[18] V. Springel, The cosmological simulation code GADGET-2, Mon. Not. Roy. Astron. Soc. 364 (2005) 1105–1134.
[19] X. Guo, S. Lind, B. D. Rogers, P. K. Stansby, M. Ashworth, Efficient massive parallelisation for incompressible smoothed particle hydrodynamics with $10^8$ particles, in: 8th Int. SPHERIC Workshop, Trondheim, Norway, 2013.
[20] P. Goswami, P. Schlegel, B. Solenthaler, R. Pajarola, Interactive SPH simulation and rendering on the GPU, in: Proc. 2010 ACM SIGGRAPH/Eurograph. Symp. Comput. Anim., Madrid, Spain, 2010, pp. 55–64.
[21] J. Onderik, Ďurikovič, Efficient neighbor search for particle-based fluids, J. Appl. Math. Stat. Inform. 4.
[22] B. Adams, M. Wicke, Meshless approximation methods and applications in physics based

modeling and animation, in: Eurograph. Tutor., 2009, pp. 213–239.
[23] O. Awile, F. Büyükkeçeci, S. Rebouxa, I. F. Sbalzarini, Fast neighbor lists for adaptive-resolution particle simulations, Comput. Phys. Commun. 183 (2012) 1073–1081.
[24] J. Feldman, J. Bonet, Dynamic refinement and boundary contact forces in SPH with applications in fluid flow problems, Int. J. Numer. Methods Eng. 72 (2007) 295324.
[25] D. A. Barcaroloa, D. Le Touzé, G. Oger, F. de Vuyst, Adaptive particle refinement and derefinement applied to the smoothed particle hydrodynamics method, J. Comput. Phys. 273 (2014) 640–657.
[26] J. Barnes, P. Hut, A hierarchical $O(N\log N)$ force-calculation algorithm, Nature 324 (1986) 446–449.
[27] D. T. Lee, C. K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, Acta Inform. 9 (1977) 23–29.
[28] R. Davé, J. Dubinski, L. Hernquist, Parallel TreeSPH, New Astron. 2 (1997) 277–297.
[29] J. Yu, G. Turk, Reconstructing surfaces of particle-based fluids using anisotropic kernels, ACM Trans. Gr. 32 (2013) 5:1–5:12.
[30] M. I. Shamos, D. Hoey, Geometric intersection problems, in: 17th Annu. Symp. Found. Comput. Sci., Houston, TX, USA, 1976, pp. 208–215.
[31] C. J. Alpert, D. P. Mehta, S. S. Sapatnekar (Eds.), Handbook of Algorithms for Physical Design Automation, CRC Press, 2008.
[32] P. Rigaux, M. O. Scholl, A. Voisard, Spatial Databases: With Application to GIS, Morgan Kaufmann, 2002.
[33] G. M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, IBM, Ottawa, Canada, 1966.
[34] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry - Algorithms and Applications, 3rd Edition, Springer-Verlag Berlin Heidelberg, Berlin, Germany, 2008.
[35] J. C. Wyllie, The Complexity of Parallel Computations, Ph.D. thesis, Cornell University (1979).
[36] R. J. Anderson, G. L. Miller, Deterministic parallel list ranking, Algorithmica 6 (1991) 859–868.
[37] R. A. Finkel, J. L. Bentley, Quad trees: A data structure for retrieval on composite keys, Acta Inform. 4 (1974) 1–9.
[38] J. L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (1975) 509–517.
[39] R. Fenk, The BUB-tree, in: Proc. 28th Int. Conf. VLDB, Hong Kong, 2002.
[40] R. Bayer, The universal B-tree for multidimensional indexing: General concepts, in: Proc. Int. Conf. WWCA '97, Tsukuba, Japan, 1997, pp. 198–209.
[41] R. Bayer, E. M. McCreight, Organization and maintenance of large ordered indexes, Acta Inform. 1 (1972) 173–189.
[42] H. Tropf, H. Herzog, Multidimentional range search in dynamically balanced trees, Angewandte Informatik 2 (1981) 71–77.
[43] C. A. R. Hoare, Quicksort, Comput. J. 5 (1962) 10–16.
[44] D. E. Knuth, The Art of Computer Programming: Sorting and Searching, Vol. 2, Addison-Wesley, Boston, USA, 1998.
[45] L. Verlet, Computer "experiments" on classical fluids. I. Thermodynamical properties of Lenard-Jones molecules, Phys. Rev. 159 (1967) 98–103.
[46] H. Shi, J. Schaeffer, Parallel sorting by regular sampling, J. Parallel Distrib. Comput. 14 (2006)

361–372.
[47] A. Agarwal, Performance tradeoffs in multithreaded processors, IEEE Trans. Parallel Distrib. Syst. 3 (1992) 525–539.
[48] D. P. Helmbold, C. E. McDowell, Modelling speedup ($n$) greater than n, IEEE Trans. Parallel Distrib. Syst. 1 (1990) 250–256.
[49] K. Kleefsman, G. Fekken, A. Veldman, B. Iwanowski, B. Buchner, A volume-of-fluid based simulation method for wave impact problems, J. Comput. Phys. 206 (2005) 363–393.
[50] S. Adami, X. Y. Hu, N. A. Adams, A generalized wall boundary condition for smoothed particle hydrodynamics, J. Comput. Phys. 231 (2012) 7057–7075.
[51] I. Gargantini, An effective way to represent quadtrees, Commun. ACM 25 (1982) 905–910.