Fast Particle Neighbor Searching for Unlimited Scene with Fluid Refraction Improvement

Sio-Kei Im and Ka-Hou Chan

***Abstract*—With the popular application of physics-based simulation in virtual reality, real-time simulation and rendering of dynamic fluids have always been the pursuit for virtual reality research. In this paper, we propose an efficient algorithm for Smoothed Particle Hydrodynamics (SPH) particle neighbor searching and rendering by taking advantage of Graphics Processing Unit (GPU) parallelization. In our method, all particles and its neighbors will be collected together and no need to search its neighboring grid actually. Thus, it does not require memory to store the start index of each grid and it can effectively solve the computational issue in inactive areas of a scene. Further, we reconstruct the fluid surfaces by the Screen Space Fluid Rendering (SSFR) method, and improve the refraction effect in real-time. The results are analyzed and discussed to show the efficiency of the scheme. Both simulation and rendering of 3D liquid computing are able to produce faster performance for realistic virtual environments.**

***Index Terms*—Smoothed particle hydrodynamics, particle neighbor searching, screen space fluid rendering, unlimited scene, fluid refraction.**

I. Introduction

Various physics-based simulation techniques have been introduced for reality simulation, but at the cost of more intensive computation in the simulation and rendering as the pursuit of realistic details and artistic quality become more demanding. With the increasing performance of hardware, the physics simulation can better approximate nature, for example representing water, waves, fire and gas. Physically plausible dynamic fluids simulation continues to be a hot research topic in Computational Fluid Dynamics (CFD). Most of the work is implemented by Smoothed Particle Hydrodynamics (SPH) [1] particle systems, a Lagrangian approach that is relatively new for simulating fluids. The fluid behavior in such an approach system will be represented by a set of independent particle motions. Compared with other physics-based simulation methods, SPH shows a greater capability for showing detail.

Unlike an Eulerian approach, SPH does not require the solving of the Poisson equations, but the methods of particle searching and visualization have to be considered. Since the SPH method needs a large number of particles for fine-detail fluids, the computational burden is quite high when we need high-resolution flows or large scenes in real-time. In order to accelerate the computation with a large particle set, researchers have proposed various methods to improve this performance. However, the limitation is the huge gap between the running particles and the computing units. The NVIDIA Compute Unified Device Architecture (CUDA) allows the programming to make use of multi-thread technology but we can only execute several thousands of particles at one turn, the other particles having to wait in the execution queue. The current parallel computing approach is logically parallel and we need thousands of computation runs to finish one simulation step. Traditionally, an SPH particle only has to interact with all of its neighbors, but animating n particles should take about $O(n^2)$ time for interaction forces, and then integrate the equations of motion for each particle to obtain its density, velocity and position. Thus, a faster particle neighbor searching and collection method can speed up the whole simulation process.

Furthermore, how to render these SPH particles as the liquid surface is another task in virtual reality. Since each particle position is discrete and the number of particles used for fluid motion description is finite, it is quite difficult to extract a surface for rendering. In early years, the Marching Cube [2] method is widely used for fluid iso-surface reconstruction. Following that, relaxation and optimization operations can be applied to the entire mesh to increase the smoothness of the surface, which is both computational and memory intensive. Actually, the Marching Cube method is very time-consuming, poor for presenting detail and the primitive size of surface extraction is dynamic and unpredictable. Currently, the screen space surface construction has been used for real-time rendering [3]. This method is a screen-view dependent GPU rendering method where the view is discretized using an adaptively sampled perspective condition, and there is no need to consider how many particles are visible to the current view. Therefore, it is suitable to apply our method in an unlimited scene (scene is constructed without grids).

In this paper, we focus on a Lagrangian approach, mesh-less particle-based simulation of viscous fluids using SPH [4] and SSFR [3] for the fluid rendering. Since the SPH and SSFR can strongly affect further calculations of fluid dynamics, so they become usually the bottleneck of the overall animation. The improved approaches for real-time fluid simulation and rendering that are proposed are based on the Graphics Processing Unit (GPU) to further speed up the performance of the fluid animation.

The main contributions of the work are summarized as follows:

- We have extended the particle neighbor searching method [5] for unlimited scene. This algorithm can effectively solve the computational issue in inactive areas of a scene.

Manuscript received December 5, 2015; revised March 1, 2016.

The authors are with the MPI-QMUL Information Systems Research Centre, Macao Polytechnic Institute, Macao, China (e-mail: marcusim@ipm.edu.mo, chankahou@ipm.edu.mo).

DOI: 10.7763/IJMO.2016.V6.506

- We have improved the refraction effect during SSFR [3]. The result is calculated from the current frame (color buffer) without provided depth buffer.
- A framework for SPH and fluid rendering for real-time simulation is developed to be fully parallel and can be implemented on the newest GPU.

The rest of the paper is organized as follows. Section II discusses the related work in SPH acceleration and rendering result. Section III gives the details of the proposed neighbor searching. Following this, the proposed algorithms for screen space fluid refraction are presented in Section IV. We then have the experimental results and discussions with other references in Section V, and finally Section VI concludes the paper.

II. Related Works

Physics-based fluid simulation is developed according to the Navier-Stokes equations. There are mainly two approaches that are Eulerian (grid-based) method and Lagrangian (particle-based) method in CFD. With the development of computing power, *Stam* had presented the flow in 3D [6], and the complex behavior was successfully simulated in real time. An adaptive sampling method was proposed based on Eulerain (grid-based) method to solve the advection effect and improve the computational performance. Recently, the liquid simulation has become widely known in computer animation. SPH was introduced into the Lagrangian (particle-based) approach by *Monaghan* [7] and in this approach fluids properties are stored in particles and the evaluation is solved through tracking the current particle motion. Thus, many researchers focus on solving the Navier-Stokes equations that govern the motion of fluids for animation.

A. Smoothed Particle Hydrodynamics Simulation

With the computational power available increasing, SPH could possibly be deployed in real-time liquid animation. SPH is already one of the most widely used methods for liquid simulation because of its flexibility to simulate the dynamic behaviors of the fluid such as splash in [8], however it is computationally intensive. Since *Müller*, *Charypar* and *Gross* [4] first introduced the SPH kernel function in liquid animation, there have been many methods proposed to improve the performance including adaptive sampling [9] [10] and predictive-corrective pressure computation [11]. In [12], the fluid simulation by SPH was implemented on a GPU, but the neighbor particle searching was still computed in a time consuming manner. Later, many GPU-based methods for the neighbor particle searching have been proposed. *Harada*, *Koshizuka* and *Kawaguchi* [13] proposed a method based on bucket sort to search the neighboring particles in a 3D grid on a GPU using the shader in graphics pipeline. Later, *Green* [5] proposed a faster particle neighbor searching method that works in the sorting-based algorithm and the complexity is $O(n \log n)$. It is suitable for fully GPU computation and in CUDA with optimization. But the searching for potential particle contacts must done within the grid cells and between immediate neighbors, thus we require more memory for cell handling and all SPH particles must be trapped in a defined container. In our paper, a neighbor search algorithm without grid constraints and full sorting is implemented on a GPU. Therefore, the particle can move to anywhere without containment and still be efficient for real-time simulation.

B. Fluid Surface Rendering

Traditionally, particle rendering via the Marching Cube algorithm [2] and the point splatting [14] method were both successfully tested in real-time rendering to present the SPH particle as a liquid like surface for rich visualation. However, the Marching Cube algorithm has to extract the iso-surface of the specified scalar field by intensive computations for each primitive. Splatting has a high requirement on the particle number for rendering. These two methods are still time-consuming. Later, many GPU-based methods [15] [16] are also introduced for fast rendering. For high-quality fluid rendering in [17], *Dyken*, *Ziegler*, *Theobalt* and *Seidel* proposed an ideal tool for high-resolution rendering. But with more and smaller cubes, more computing resource is needed and the particle is still trapped in a defined container. In recent years, *van der Laan*, *Green* and *Sainz* proposed a screen space for real-time fluid rendering [3], in which the fluid property will be projected into screen space and be analyzed as a 2D case. It makes the surface extraction more explicit, which greatly reduced the geometric considerations. But this method may lose the 3D info during the projective operation, and some issues can be exposed in an illuminated scene (such as light reflections and refraction effects). In our paper, the refraction color is captured form the screen space frame buffer (not cube mapping). The advantage of our algorithm is that the refraction result can consider the non-background objects without using dynamic cube mapping.

III. Particle Neighbor Searching in Unlimited Scene

The main advantage of SPH is that each particle only affects its neighboring particles. Thus, the SPH method can reduce the scaling problem of the partial differential equations, but with the disadvantage of the computational intensity of SPH, making it challenging to be used for real-time applications.

A. Fluid Simulation by SPH Method

In the SPH method, the fluid is represented by a finite number of sampled particles, and the properties will be expressed by these particles. The attributes of particle can be described by the kernel function as follows:

$$A_i = \sum_j A_j \frac{m_j}{\rho_j} \omega\left(\left\|\vec{r}_i - \vec{r}_j\right\|, h\right) \quad (1)$$

where A_i is the attribute of particle i (mass and density), $\vec{r}$ is the current position, j indicates the index of all the neighboring particles within the support domain h of particle i, and $\omega\left(\left\|\vec{r}_i - \vec{r}_j\right\|, h\right)$ is the smoothing kernel for the fluid simulator. The support domain h is a spherical space with an effective radius of the smoothing length. Commonly, these kernel functions have a "bell curve" shape and multiple

kernels offer better results in liquid simulation. Similarly, the Gradient and Laplacian of the smoothed properties functions A are computed by using the Gradient and Laplacian of ω, respectively. Generally, the density is used for computing ρ as expressed by (2):

$$\rho_i = \sum_j m_j \omega\left(\left\|\vec{r}_i - \vec{r}_j\right\|, h\right) \tag{2}$$

For the pressure term that is used in the internal force computations, we employ the ideal gas state equation [1]. Further, the internal pressure and viscosity forces computing can be obtained by the set of kernel functions that was proposed by [4] as (3) and (4).

$$\vec{f}_{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla\omega\left(\left\|\vec{r}_i - \vec{r}_j\right\|, h\right) \tag{3}$$

$$\vec{f}_{viscosity} = \mu \sum_j m_j \frac{\vec{v}_j - \vec{v}_i}{\rho_j} \nabla^2\omega\left(\left\|\vec{r}_i - \vec{r}_j\right\|, h\right) \tag{4}$$

B. Particle-Cell Relation Table Construction

Our simulation has extended the approach proposed in [5]. Similarly to the earlier work, the scene space is discretized into a virtual uniform indexing grid in the X, Y and Z-axes with the cell size always being equal to the SPH support domain h. Then all of the uniform grid will be employed for convenient particle storage and neighbor searching. This comes from the observation that one particle's neighbors must reside in its 27 surrounding grid cells in 3D. However, different from others, we do not map these (X,Y,Z) to one index (*e.g.* Hash result or Z-indexing technique [18]). The reason is that the minimum and maximum number of cells are no longer in our proposed approach, thus mapping could not provide the definitive value as a hash code for the hash operation.

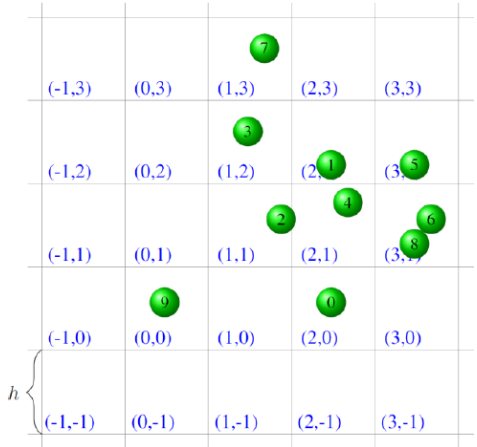

Fig. 1. A uniform grids subdivide the simulation space into a grid of uniformly sized (same as the smoothing kernel domain h) cells.

As shown in Fig. 1 for a 2D representation, every cell size is the same as the smoothing kernel domain h. Each particle only needs to query 9 cells for potential neighbor searching. It must provide the *Start* and *Number* to specify the first index and particle number where the specific cell starts and how many particles there are in the sorted table. In our proposal, a *Particle-Cell* relation table is also involved in particle searching. We use (X,Y) instead of a single value, and store the particle information after each simulation step. Then we sort the array according to their (X,Y) in lexicographical order. Since CDUA does not provide the lexicographical sorting, we have to implement the pseudocode for sorting in CUDA (see Fig. 2).

```
Data: A(x, y, z): int3
      B(x, y, z): int3
Result: Whether A is less then B: bool
/* lexicographical sorting                    */
begin
    /* both vectors are the same size         */
    for i ← 0 to Vector Size do
        if A(i) < B(i) then
            return true;
        else if A(i) > B(i) then
            return false;
        end
        /* else mean A(i) equal to B(i)       */
    end
    /* A equal to B                           */
    return false;
end
```

Fig. 2. Compare two integer-vector *A* and *B*.

C. Particle Neighbor Searching and Collection

Although the uniform grid method can improve the neighbor searching performance, it needs to create an additional table to store the *Particle-Cell* related *Start* index and these indexes must be positive values. The reason is that each particle must prepare which 9 (in 2D, 27 in 3D) indices will start to be searched, and these searching regions are dynamic so we need to update them in each simulation loop. In order to reduce this additional work, we propose the relation table to store more information. Since each particle will become a potential neighbor for its 9 surrounding grid cells in 2D, we require 9 (2D case, 27 in 3D) more elements to present the *Particle-Cell* relation.

Fig. 3 shows the *Particle-Cell* relation table of particle 0 to 2 corresponding to Fig. 1. This related table has collected the potential particles together, and then each particle can exactly find its neighbor in a region. There is no need to create an additional table to store the *Particle-Cell* related *Start* index, and each particle can store the searching *Start* index itself. Further, the cell index (X,Y,Z) allows negative values that can be obtained by the current position as:

$$(X,Y,Z) = floor\left(\frac{Position}{h}\right) \tag{5}$$

where h is the smoothing kernel domain. After the *floor* operation in (5), (X,Y,Z) will be converted to integer values and stored in the *Particle-Cell* relation table with a particle index (see Fig. 3a). We then sort these elements based on the cell index using Fig. 2, a parallel radix sorting algorithm, and the sorting performance

is $O(n \log n)$ (see Fig. 3b). Then we should prepare the *Start* and *Number* for searching operation (see Fig. 3c). Finally, every particle and its potential neighbor will have been collected in the same region, thus there is only one loop searching operation and we know exactly how many particles should be queried in our method. This makes it suitable for parallel programming design and implementation. It should be noted that there is no concern on particle position, so that a particle can move to anywhere without container constraints in our method.

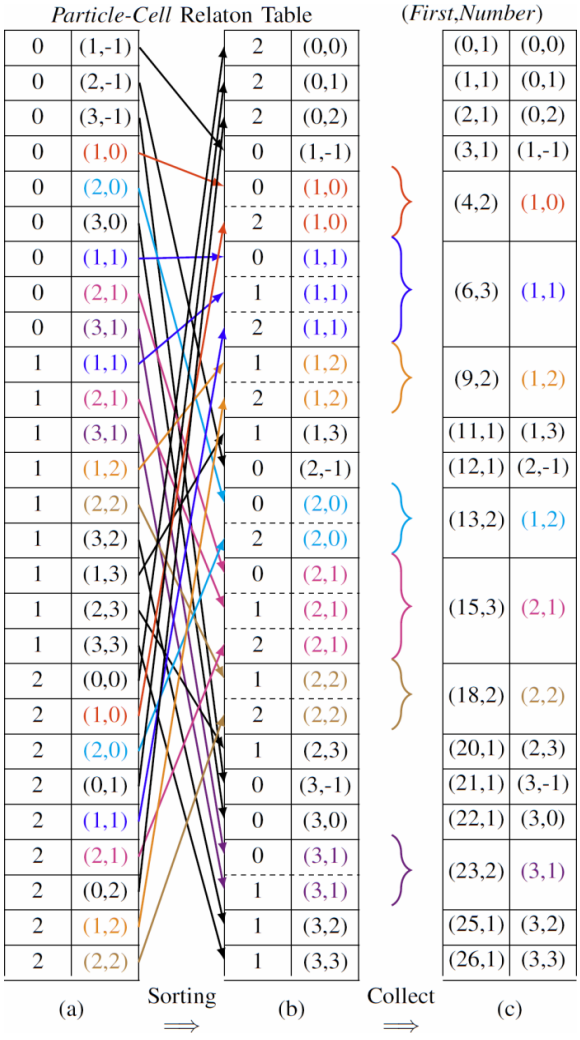

Fig. 3. Flow chart of sorting and collect the researching particles.

IV. Fluid Surface Rendering

In screen space rendering, the scene effect can be achieved as an image effect by 2D algorithms (*e.g.* edge detection and ambient occlusion [19], [20]). This fact can reduce greatly the time needed for computation, and will also make it predictable and more suitable for real-time simulation. With fluid rendering, since the topology of the fluid surface is dynamic, the surface construction is a heavy load for traditional 3D methods, thus we prefer to use SSFR [3] for fluid rendering.

A. Screen Space Fluid Rendering

All SPH particles have been updated after each simulation loop. They will be rendered in any order and projected to 2D screen coordinates, in which particle size usually depends on current density. The basic steps of the SSFR method are listed as follows:

1) the method in [21].
2) The resulting depth buffer and thickness is written to two render targets.
3) Use two-pass bilateral filter to blur depth buffer and thickness.
4) Generate a dynamic noise map on the surface depth then also blur it.
5) According to these blurred depth buffer values, find the surface positions of each pixel, and then calculate a normal map for the scene in view coordinates.
6) Apply the noise and bump mapping for normal recalculating and lighting result.
7) Calculate thickness using the opacity of the additive blend result, and then calculate absorption for color, and thickness for opacity.
8) Use additive blending to render the whole scene so we can obtain some measurement of lighting results including reflection, refraction and Fresnel [22] effect.

In step 8, the whole scene means the non-fluid rendered result (such as ground floor and background objects). They should be rendered before the SSFR, and then can be used as a scene (2D) texture in the final step.

```
Data: Thickness: float
      Texcoord: float2
      N_view: float3
Result: Refracted result in SSFR: float3
/* sample code in GLSL                    */
begin
    /* refract is adjusted by fluid
       thickness and surface normal        */
    Thickness ← 1.0 − exp(−Thickness);
    Texcoord ← Texcoord + N_view.xy * Thickness;
    return texture(SceneMap, Texcoord);
end
```

Fig. 4. SSFR background image refracted.

B. Screen Space Fluid Refraction

Commonly, we can use the refracted vector in a fragment shader to access a dynamic cube mapping [23] to determine the surface color for a transparent object. A built-in function for refraction vector calculation is provided in shading language as:

$$R_{refracted} = refract\left(I, N, \frac{n_1}{n_2}\right) \tag{6}$$

where I is a given incident vector, fluid surface normal N and ratio of indices of refraction for two materials n_1/n_2, and applies Snell's Law to compute the refracted vector $R_{refracted}$. However, this method requires pre-rendering the scene (with different orientations of the camera) six more times. In the work in [3], the whole scene is pre-rendered once and the refraction result is obtained as Fig. 4.

As shown in Fig. 4, N_{view} is the surface normal in view coordinates, P_{view} is the vector from original to surface

position in view coordinates. The Fig. 4 does not consider the ratio of indices, thus causing an unnatural rendering effect when the thickness is large (see Fig. 6). In order to solve this problem, we combine the *refract* function in (6) with both refractive indices as an approximation (see Fig. 5).

```
Data: n1: float
      n2: float
      Pview: float3
Result: Refracted result in our proposed: float3
/* sample code in GLSL                          */
begin
    /* refract is adjusted by fluid
       thickness, surface normal, depth
       of view and refractive indices           */
    Thickness ← 1.0 − exp(−Thickness);
    n1 ← mix(n2, n1, Thickness);
    E ← normalize(Pview);
    Texcoord ← refract(E, Nview, n1/n2).xy/ − E.z;
    /* convert to texture coordinates           */
    return texture(SceneMap, Texcoord ∗ 0.5 + 0.5);
end
```

Fig. 5. Improve background image refracted.

(a) SSFR background image refracted, *t*=10s.

(b) SSFR background image refracted, *t*=15s.

Fig. 6. Screen space refraction effect calculated by SSFR.

(a) Improved background image refracted, *t*=15s.

(b) Improved background image refracted, *t*=15s.

Fig. 7. Improved screen space refraction effect (with better scene depth) calculated by our method.

V. Experimental Results

We have developed and tested both SPH simulation and SSFR rendering on a Windows 7 64-bit machine with the following hardware and software configuration: Intel(R) Xeon(R) CPU X5647@2.93GHz, 12.0GB memory and NVIDIA Quadro 6000, CUDA 7.0, OpenGL 4.5 and GLSL 4.5.

To test the visual quality of our method, we compare the refractive results of the same frame with and without our proposed algorithm. We can see that there is obvious visual difference between the SSFR [3] and our optimized method (see Fig. 6 and 7). When the fluid thickness becomes larger, the result will be strange with noise artifacts in the traditional method. With our method, we can see that the visual quality is natural and realistic.

TABLE I: Time-Consuming for Indices Sorting and SPH Computation for One Time-Step Including the Rendering (in Milliseconds)

Particle numbers	Sorting (ms)	SPH (ms)	Overall (ms)
32,768	0.23	17.29	18.02
40,960	2.53	22.81	25.84
49,152	5.69	26.17	32.36
57,344	8.72	29.84	39.06
65,536	10.67	33.28	44.45
73,728	12.79	36.74	50.03
81,920	14.38	41.94	56.82
90,112	15.41	46.99	62.90
98,304	16.34	51.19	68.03
106,496	16.74	54.19	71.43

To test the performance in SPH simulations, we implemented the method and measured the computation times. Table I shows the results and the time-consuming for sorting indices and SPH computation. We also compare our method with those in [18] and [5]. The testing results are shown in Fig. 8 with the same simulation material setting. Our method can lead to speed improvements from 10% to 20%. The reason for the improved performance is that there is no longer the need to use grid cells handling, making it suitable for parallel design and computing. This indicates that they are memory bound, which is expected given that their main feature is to reorder memory structures based on the cell index which is very fast to compute. For the force calculation procedure the speedup is considerably closer to the optimal speedup, indicating that the corresponding kernel functions are compute-bound as we expected.

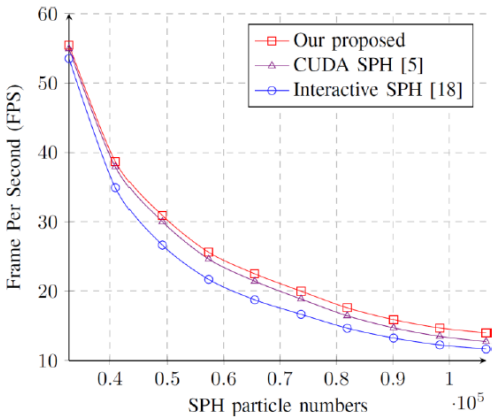

Fig. 8. Comparison of the simulation speed in FPS.

VI. Conclusion

In this paper, we focus on the parallel-optimization, simulation and rendering of SPH fluids for an unlimited scene. A new cell index storage and sorting method are presented to remove the constraints on the containers

condition. The performance of fluid computation is not obviously affected. With the fast screen render technique, we improved the refraction effect without increased workload. Hence, our proposed approach can easily be used for related development and our future work focuses on reducing the pre-render workload, which may involve looking at using an implicit formulation of the integration scheme, as this will be more stable and require fewer time-steps and thus improve performance.

REFERENCE

[1] M. Desbrun and M.-P. Gascuel, *Smoothed Particles: A New Paradigm for Animating Highly defoRmable Bodies*, Springer, 1996.

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM Siggraph Computer Graphics*, vol. 21, ACM, 1987, pp. 163-169.

[3] W. J. V. D. Laan, S. Green, and M. Sainz, "Screen space fluid rendering with curvature flow," in *Proc. the 2009 symposium on Interactive 3D graphics and games*, ACM, 2009, pp. 91-98.

[4] M. Müller, D. Charypar and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proc. the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, Eurographics Association, 2003, pp. 154-159.

[5] S. Green, "Particle simulation using cuda," *NVIDIA Whitepaper*, 2010.

[6] J. Stam, "Stable fluids," in *Proc. the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121-128.

[7] J. J. Monaghan, "Smoothed particle hydrodynamics," *Reports on Progress in Physics*, vol. 68, no. 8, p. 1703, 2005.

[8] R. A. Dalrymple and O. Knio, "SPH modelling of water waves," in *Proc. Coastal Dynamics*, 2000, pp. 1082-1091.

[9] J. Swegle, D. Hicks and S. Attaway, "Smoothed particle hydrodynamics stability analysis," *Journal of computational physics*, vol. 116, no. 1, Elsevier, pp. 123-134, 1995.

[10] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas, "Adaptively sampled particle fluids," *ACM Transactions on Graphics (TOG)*, ACM, 2007, p. 48.

[11] B. Solenthaler and R. Pajarola, "Predictive-corrective incompressible SPH," *ACM Transactions on Graphics (TOG)*, ACM, 2009, p. 40.

[12] A. Hérault, G. Bilotta and R. A. Dalrymple, "SPH on GPU with CUDA," *Journal of Hydraulic Research*, vol. 48, pp. 74-79, 2010.

[13] T. Harada, S. Koshizuka and Y. Kawaguchi, "Smoothed particle hydrodynamics on GPUs," *Computer Graphics International*, SBC Petropolis, 2007, pp. 63-70.

[14] H. Pfister, M. Zwicker, J. V. Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proc. the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 335-342.

[15] Y. Zhang, B. Solenthaler and R. Pajarola, "Adaptive sampling and rendering of fluids on the GPU," in *Proc. the Fifth Eurographics/IEEE VGTC conference on Point-Based Graphics*, Eurographics Association, 2008, pp. 137-146.

[16] G. Akinci, M. Ihmsen, N. Akinci, and M. Teschner, "Parallel surface reconstruction for particle-based fluids," *Computer Graphics Forum*, Wiley Online Library, 2012, pp. 1797-1809.

[17] C. Dyken, G. Ziegler, C. Theobalt and H.-P. Seidel, "High-speed Marching Cubes using HistoPyramids," *Computer Graphics Forum*, Wiley Online Library, 2008, pp. 2028-2039.

[18] P. Goswami, P. Schlegel, B. Solenthaler and R. Pajarola, "Interactive SPH simulation and rendering on the GPU," in *Proc. the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 2010, pp. 55-64.

[19] L. Bavoil and M. Sainz, "Screen space ambient occlusion," *NVIDIA Developer Information*, vol. 6, Citeseer, 2008.

[20] M. Tarini, P. Cignoni, and C. Montani, "Ambient occlusion and edge cueing for enhancing real time molecular visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, no. IEEE, pp. 1237-1244, 2006.

[21] P. Zemcik, P. Tisnovsky, and A. Herout, "Particle rendering pipeline," in *Proc. the 19th Spring Conference on Computer graphics*, ACM, 2003, pp. 165-170.

[22] M. Wloka, "Fresnel reflection technical report," *nVidia Corporation Paper*, 2002.

[23] T. Sousa, "Generic refraction simulation," *GPU Gems*, vol. 2, pp. 295-305, 2005.

Sio-Kei Im received his degree in computer science in 1998, the master degree in enterprise information system, in 1999, both from King's College, University of London, United Kingdom, and the PhD degree in electronic engineering in 2007 from Queen Mary University of London (QMUL), United Kingdom.

He gained the position of lecturer within the computing programme at Macao Polytechnic Institute (MPI) in 2001. In 2005 he became the operations manager of the MPI-QMUL information systems research centre jointly operated by MPI and QMUL where he carried out signal processing work.

Prof. Im was promoted to professor at the Macao Polytechnic Institute in 2015. He is a visiting scholar at School of Engineering, University of California, Los Angeles (UCLA) and a member of The Institution of Engineering and Technology (IET).

Ka-Hou Chan received his master degree in Faculty of Science and Technology from the University of Macau (UMAC) in 2015, and the bachelor degree in computer science from the Macau University of Science and Technology (MUST) in 2009.

He is also a research assistant working at Macao Polytechnic Institute (MPI). He has several papers published in the fields of computer graphics and virtual reality. His research interests include computer animation, physically based modeling, parallel computing, and video compression