Zhang et al. EURASIP Journal on Image and Video Processing (2017) 2017:79
DOI 10.1186/s13640-017-0227-9

EURASIP Journal on Image and Video Processing

RESEARCH Open Access

A fast framework construction and visualization method for particle-based fluid

Fengquan Zhang[1*], Zhaowei Wang[1], Jian Chang[2], Jianjun Zhang[2] and Feng Tian[3]

Abstract

Fast and vivid fluid simulation and visualization is a challenge topic of study in recent years. Particle-based simulation method has been widely used in the art animation modeling and multimedia field. However, the requirements of huge numerical calculation and high quality of visualization usually result in a poor computing efficiency. In this work, in order to improve those issues, we present a fast framework for 3D fluid fast constructing and visualization which parallelizes the fluid algorithm based on the GPU computing framework and designs a direct surface visualization method for particle-based fluid data such as WCSPH, IISPH, and PCISPH. Considering on conventional polygonization or adaptive mesh methods may incur high computing costs and detail losses, an improved particle-based method is provided for real-time fluid surface rendering with the screen-space technology and the utilities of the modern graphics hardware to achieve the high performance rendering; meanwhile, it effectively protects fluid details. Furthermore, to realize the fast construction of scenes, an optimized design of parallel framework and interface is also discussed in our paper. Our method is convenient to enforce, and the results demonstrate a significant improvement in the performance and efficiency by being compared with several examples.

Keywords: Art visualization, Multimedia, Fluid simulation, Bilateral filter, Particle-based

1 Introduction

Realistic and real-time simulation of physically based fluid is a hot research topic in virtual reality. Fluid simulation is used in many areas such as art visualization, video game, image effects, industry simulation, and entertainment media. Physically based method has been applied in fluid simulation for achieving vivid and high--quality visual image. In interactive fluid simulation, as one common physical simulation method, the particle-based method, such as Smooth Particle Hydrodynamics (SPH), has been extensively used for creating realistic image effects of fluids in image and video game due to superiority of Lagrangian representations [1, 2].

In these technologies, however, the rendering of fine details and lower cost for fluid surface are usually the complex problems because of the irregularly moving and expensive consumption, and some works have been presented to address this problem [3–6]. Among these methods, solving a scalar density field has been shown to be an effective method. Since it is general that the method needs to compute a polygonized isosurface of 3D field with marching cubes, the resolution of mesh will make a large influence about the quality of fluid surface and performance. The other issue comes from employing the computational power and parallelism of the accelerated GPUs. We find that there are few data dependencies in the standard particle-based method. This makes it possible to enforce the simulation and rendering on the GPUs platform, thus decreasing the complex data exchange between system and graphics memory.

In this work, we focus on the parallel framework design and visualization method for fast constructing and online rendering. The key characteristic of this work is that the entire physical simulation, including SPH computation and visualization of the particle sets, is

* Correspondence: nicky2008@163.com
[1]Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, North China University of Technology, Beijing, China
Full list of author information is available at the end of the article

carried out on our proposed parallel framework. As far as we know, this is the first address about the design of particle framework which can execute all the simulation works from solution deployment to fluid image effects. The main contributions of our work are as follows:

(1) Efficient visualization. A direct surface visualization pipeline for particle fluids such WCSPH, IISPH, and PBF is designed based on screen-space method, which utilizes GPU programmable pipeline to filter internal particles in geometric handling and can render fluid surface by visible particles in sub pixels. In order to get smooth surface, a new filtering model is presented, instead of Gaussian filter, the bilateral filter is used for depth image to create smooth fluid. Our method does not need conventional surface reconstruction steps and protects details.
(2) Fast constructing. Based on the features of less data dependencies in particle methods, we propose a fast constructing framework on the multi-GPU platform. A flexible interface is designed to deploy art effects conveniently based on the unified message layer. A concept of particle buffer is presented to execute the particle system management, and the algorithm is abstracted to a running instance. The framework can automatically establish the system mapping according to the requirement of user, which guarantees that the operation of application is fast and friendly.

2 Related works

The real-time fluid simulation is a challenge work, where the animation or art visual special is computed with a large scale of numerical modeling. Difficulties in the creation of an efficient method for fluid dynamics arise from the difference of appearances of animation which can be obtained bubbles, fluid-solid interaction, multi-phase interface, viscoelastic object et al. [7–11]. Existing methods such as predictive-corrective incompressible SPH (PCISPH), point-based fluid (PBF), and implicit incompressible SPH (IISPH) [12–16] create many vivid fluid animations with various characters. However, most of methods of simulation and visualization are unreal-time due to the complexity of physics computing. SPH is a famous particle-based method to get fluid behaviors, and it has the specialty in high degree of parallelism and has been tested on the variety of Graphic Processing Units (GPUs) platform [17–19]. SPH method can well accelerate by these devices and obtain a high speedup, which provide a solid foundation for online simulation and rendering. The earliest computer unified device architecture (CUDA) implementation of the SPH fluid was executed on the platform of INGV-CT [20]. Then, the full implementation on a single GPU was presented on some projects [21, 22]. With the increasing of requirement, multi-GPU versions of the SPH were introduced to simulate higher resolution of particle fluid [23, 24]. Based on their works, we will build a parallel framework to realize the objective in this paper.

In recent years, the visualization techniques of fluid are presented to reconstruct surfaces in many literatures. The main difficulty in surface reconstruction is not only to create smooth surface, but also to protect fine details, for example splashes, sheets, and droplets [25]. The traditional methods for visualization of fluid are polygon-based, voxel-based, and point sprite algorithm [3, 26]. A mesh-based level-set method was proposed in [27], which generated good fluid surface but with the high computing cost. Zhu and Bridson [4] proposed a method with an implicit function to get scalar field creating smooth surfaces. However, the technique leads some artifacts in concave regions. Guennebaud and Gross [28] presented an improved Moving Least Squares (MLS) method to improve the stability in low sampled and high curved sections. Unfortunately, because the computing of projection is cost expensive, it cannot be used for upsampling points in real time. In [29], an adaptive scalar field model was proposed for reconstructing fluid surface. Although it can generate high quality fluid, its accuracy depends on very large Marching Cube (MC) grids, resulting in the large computational time and memory requirements. Akinci [30] employed a multi-pass rendering method to reconstruct large-scale particle fluid surface with GPU pipeline, which had a good performance improvement compared to with ray casting method. However, it is only for rendering of form and foam fluid.

In this paper, our visualization method is inspired by the technique of [6], which renders surface using the screen-space algorithm. Compared with the traditional method, method [30] can render large particle fluids on the GPU and avoid grid drawbacks. However, there are some limitations when used for rendering surface. First, the surface of fluid is not smooth enough, especially in concave regions. Then, the effects of lighting are not natural and cannot effectively preserve the details of splashes and droplets. Moreover, it is easy to emerge boundary diffusion due to the method of filtering. Our interface of parallel framework is inspired by the method of [24], which implements the SPH simulation on multiple GPUs. However, method [24] is not fit for fluid rendering. With the widely demand in complex art effects, designing a high efficient framework for particle fluid is a very important requirement. Therefore, based on the feature of particle method, a friendly interface of parallel framework is designed in this paper, which can adaptively model and produce image effects and animations by fast construction.

3 Particle-based framework

Some particle-based methods have been presented in fluid simulations. Below, we focus our implementation using SPH method and briefly introduce the standard SPH model. In SPH, the fluid is expressed by irregular particle sets, and each particle has its mass and density. In each time step, physical quantities can be computed with summing up the weighted contributions of around particles. For particle i, mass m_i, position r_i, and density ρ_i are interpolated by a sum of the weighted of neighboring particle j as

$$\rho_i = \sum_j m_j W(\boldsymbol{r}_{ij}, h) \quad (1)$$

where $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$, W is the smoothing kernel and h is the smoothing length. In our simulation, m and h are constant. The pressure p_i is computed by a function of the density of the filed, and the Tait equation is used as

$$p_i = k\rho_0\left(\left(\frac{\rho_i}{\rho_0}\right)^{\gamma} - 1\right) \quad (2)$$

where γ and k are stiffness coefficient and ρ_0 is the rest density. In SPH framework, the pressure and viscous force of particles are derived from the Navier-Stokes equation as

$$\rho_i \frac{\phi \boldsymbol{v}_i}{\phi t} = -\langle \nabla p \rangle(\boldsymbol{r}_i) + \mu \langle \Delta \boldsymbol{v} \rangle(\boldsymbol{r}_i) + \boldsymbol{f}_i \quad (3)$$

where $\boldsymbol{v}_i$ is the velocity of particle i, $\boldsymbol{f}_i$ is an external force as gravity, μ is the viscosity coefficient, and $\langle \nabla p \rangle(\boldsymbol{r}_i)$ and $\langle \Delta \boldsymbol{v} \rangle(\boldsymbol{r}_i)$ denote the pressure gradient and the velocity Laplacian. We refer readers to see [2] for more details about different particle methods and applications in image and animation effects.

4 Method

Before visualization, we first show some data to illustrate the relation of the number of particles and computation time. Although the method of adaptive meshing can retain some details, it incurs much performance cost in sampling, and hardly achieve real-time rendering with GPU. As is shown in Table 1, when the number of particles increases from 122 to 234 K, the time of surface reconstruction increases by 2.121 times meanwhile rendering time increases by 1.134 times. It then can be indicated that the time of surface reconstruction has positive correlation with the scale of particle sets.

Table 1 The time relation of surface reconstruction, rendering in different number of particles

Time	122 K	234 K	Ratio (1.91)
Surface reconstruction (ms)	24.101	51.094	2.121
Rendering (ms)	29.728	45.054	1.134

To improve the problem shown above, we design and realize a direct rendering algorithm based on screen space for particle-based fluid, which utilizes GPU programmable pipeline to filter internal particles in geometric handling, and render fluid surface by visible particles in subpixels. The proposed algorithm does not need surface reconstruction steps in conventional rendering and retains all the details in original simulation. Major steps are summarized as follows:

(3) Do background mapping rendering;
(4) Generate fluid geometric buffer: draw particles as spheres, then use hardware depth measurement to acquire surface depth, compute, and save fluid texture information in the shader;
(5) Use depth information acquired in step 2 to smoothen the depth;
(6) Draw all 3D points as spheres and compute fluid depth by hardware Alpha;
(7) Do fluid shading: apply illumination algorithm on geometric buffer to compute refraction and reflection rays by fluid depth and then use the rays to sample from background mapping for presenting background color, at last, fuse these colors.

Background is used to define non-transparent models and environment whereas foreground means transparent solid object and fluid. The above algorithm decouples foreground and background rendering. For background rendering, any known method is allowed as long as the rendering can retain non-transparent objects' depth buffer value. In this way, the proposed algorithm can be easily integrated with existed systems [31, 32].

4.1 Background mapping rendering

The background mapping is adaptively rendered by dynamic selecting proper pipelines. The selection process is explained as follows. The number of non-transparent objects is expressed as N_s, the number of lights as N_l, and the ratio of these two values as $v = N_s/N_l$. We then define a threshold value as n. When $v < n$, rendering pipelines based on screen space is used. Otherwise, conventional rendering pipeline is used. Since depth of background mapping is required for fluid rendering, frame buffer is turned on after a certain rendering pipeline is chosen to draw scenes into textures as background mapping. If the forward pipeline rendering is chosen, the depth mapping is required when setting frame buffer. If the screen space rendering is chosen, the depth information in its geometric buffer can be used repeatedly.

When $v < n$, we use the technology of deferred shading for rendering of screen space, which includes the geometric processing stage and the lighting computation stage. In the stage of geometric stage, it only uses geometric data of object in the scene, and it is responsible for transforming the vertices into the viewing coordinate at the same time the filling of the position of vertex, the direction of the normal vector, and the color of the diffuse reflection into the G-Buffer. In the process of lighting computation, the data in G-buffer and the properties of the light source are used to calculate the illumination of each pixel by texture operation.

4.2 Generation of fluid geometric buffer

We can utilize hardware acceleration function of GPU to remove invisible primitives in geometric handling. Rendering depth is defined as z value from *near* to *far* range. Then after perspective transformation, a new z' value can be expressed by

$$z' = \frac{\text{far} + \text{near}}{\text{far} - \text{near}} + \frac{1}{z}\frac{-2^{*}\text{far}^{*}\text{near}}{\text{far} - \text{near}} \tag{4}$$

After z' s normalization in [− 1,1], the value of near range plane is in − 1 while the value of far range plane is in 1, then all primitives out of this range will be removed automatically by hardware and would not be involved in the next step of pipeline rendering. The value saved in depth buffer is the nonlinear depth value z' of the primitive in the eye coordinate system.

We apply point sprites to draw 3D particle point cloud to make sure that each applicable 3D point covers minimal pixels in rasterization. Point sprites, a regular Billboard method, refer to a movable image on the screen. Simple point sprites image bear one-to-one correspondence with pixels on the screen. In this way, GPU will generate texture coordinates $((x, y)\ \ x, y \in [0, 1])$ for each 3D point and discard pixels outside the range of $x^2 + y^2 < = 1$. Therefore, pixels in screen circles can be covered in rasterization.

Since point sprites contain only 2D information, its range has to be defined in rendering according to perspective principles. The value would contribute to the rendering quality; therefore, different weights are assigned to different scenes. In this paper, we use the scale by point of sight as

$$S_p = \frac{v_s}{a} d \tag{5}$$

where v_s denotes the projection length on Z axis, and d for the distance from the viewing plane. Weight factor a, is subject to amendment for artistic purposes.

In fragment, we amend the depth of point sprites to make sure that overlapping surface particles can pass hardware depth test and retain details. More specifically, we use the point position P_{eye}, normal vector N_p, and the value of point sprites to compute projection on the screen space P_{prj} as

$$P_{\text{prj}}(x, y, z, w) = M_p\left(P_{\text{eye}} + S_p N_p\right) \tag{6}$$

N_p denotes the normal vector of point sprites on the screen space. And from the texture coordinates (x', y') in the local coordinates system, we can infer that

$$N_p = \left(x', y', 1 - x^2 - y^2\right) \tag{7}$$

Then, the amended depth D_n can be expressed as the ratio of component z and w:

$$D_n = \frac{P_{\text{prj}}(z)}{P_{\text{prj}}(w)} \tag{8}$$

In screen space, rendering, shading, and illuminating computation are processed in fragment, when 3D primitives are transferred into 2D information corresponding to depth value. Some of the geometric information can be retrieved by the fragmented 2D information and depth value, such as the coordinates of a point in the eye coordinate system whereas irrecoverable information has to be written into frame buffer in primitive vertex processing stage as texture output. After the depth value is amended, the new depth is written into hardware depth buffer by the fragment shader. Then, the buffer is filled and outputted by the layout of fluid geometric buffer as shown in Table 2. The geometric buffer of particle consists of four channels. We use 32-bit 4-channel floating point textures to fill the buffer, where the RGBA four channels store the corresponding geometric properties. The remaining bits save the material properties of fluid using the pack mode. R channel is a 32-bit floating data that saves z value of current point in the eye coordinate. G channel is a 32-bit pack value, which saves the color of the transparent object. Each color component is stored in 8-bit memory space. B channel is used for the absorption coefficient. The attenuation factor of light is represented with three 8-bit binaries. A channel is 32-bit pack value that holds the transparent property, where 24-bit position is used to store the index of refraction of the transparent object and 8-bit holds the transparency.

4.3 Generation of fluid surface

Geometric buffer which generates surface fluid takes place in GPU primitive vertex processing and fragment stage. At this stage, geometric buffer only contains outmost information of SPH simulation particles, namely the geometric information of 3D points covering fluid surface. SPH simulation uses particles to express fluid. Although the introduction of point sprites ensure that all points contributing to the screen pixels are rendered, discrete points cannot create a smooth fluid surface.

Table 2 The mode of buffer allocation

The layout of geometry buffer										
R (32 bit)	G (32 bit)				B (32 bit)				A (32 bit)	
$P_{eye}(Z)$	Color				Absorption coefficient				Transparent property	
	R (8 bit)	G (8 bit)	B (8 bit)	A (8 bit)	A_r (8 bit)	A_g (8 bit)	A_b (8 bit)	Null	Refractive index (24 bit)	Transparency (8 bit)

To smooth particle's depth buffer, we combine with the characteristics of GPU rasterization rendering. Generally speaking, Gaussian functions are often used for smoothing purposes in image processing by applying high-pass filtering on the original image. Yet the priority of Gaussian filter is pixel's spatial distance, so little consideration is given to the resemblance of pixels. Therefore, the result is always fuzzy and obscure with much detail loss, as is shown in Fig. 1b, image smoothed by kernel function as Eq. 9, in which u and v are pixel's coordinates, and standard deviation in Gaussian distribution. Gaussian smoothing is the process of computing the weight average of surrounding pixels.

$$G(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}} \tag{9}$$

From Fig. 1b, we can see that smoothing on the same image and depth range can produce satisfying results, whereas obvious details loss occurs when there is a depth difference or the margin is outside Gaussian smoothing. We then use bilateral filter to smooth fluid depth buffer as is shown in Fig. 2c. Different from Gaussian filter, bilateral filter can retain margin since it weighs both spatial distance and the similarity of pixels. Figure 2d shows the original 3D color frequency images, and the smoothed images are shown by Gaussian and by bilateral filter respectively. It is clear that bilateral filter smooth high-frequency signals while retaining marginal features of the original image.

In Gaussian filter, surrounding pixels are assigned with different Gaussian weight and then are averaged. A weight factor is generated by the spatial distance between two pixels. From the Gaussian distribution curve, we know that the closer one pixel is to the target pixel, the bigger its contribution to the final result is, and also vice versa. The advance of bilateral filter is to add another weight factor to retain marginal details, as is expressed in

$$h(x) = k^{-1}(x) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi) c(\xi, x) s(f(\xi), f(x)) d\xi \tag{10}$$

$$k_r(x) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, x) s(f(\xi), f(x)) d\xi \tag{11}$$

where $h(x)$ is the image of the result. s denotes pixels' similarity, $c(\xi, x)$ for spatial similarity, and $k_r(x)$ for result

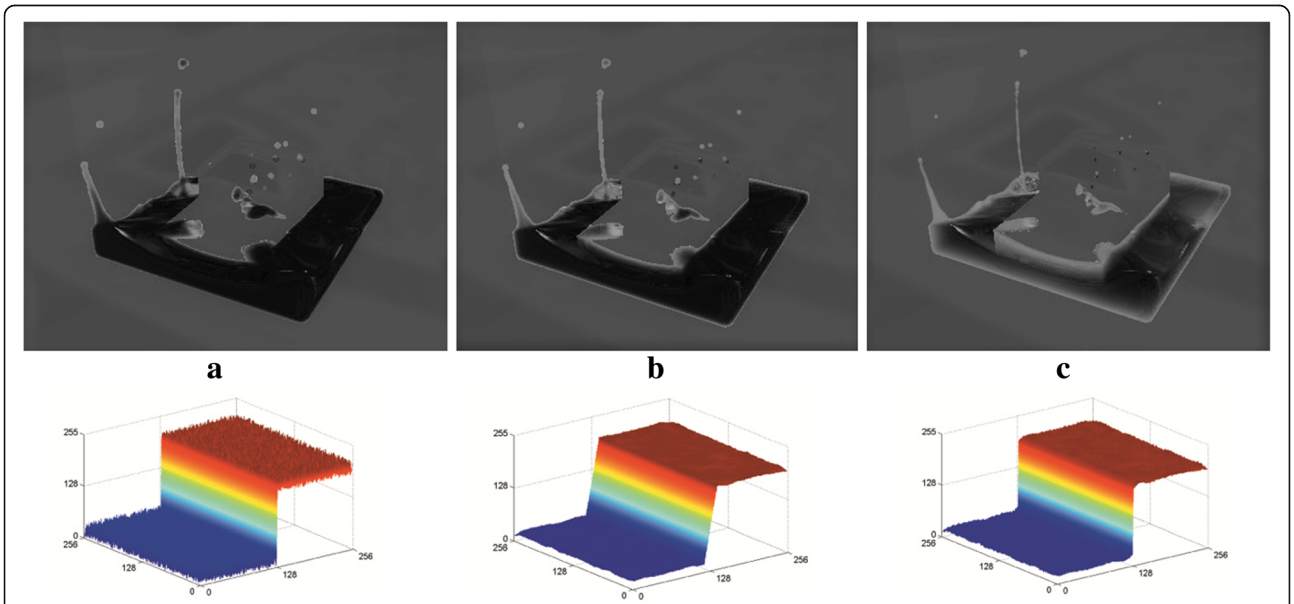

Fig. 1 Comparison of different smoothing methods. **a** Unsmoothed. **b** Smoothed by Gaussian filter. **c** Smoothed by bilateral filter. **d** From the left to right: original image, Gaussian filter, bilateral filter

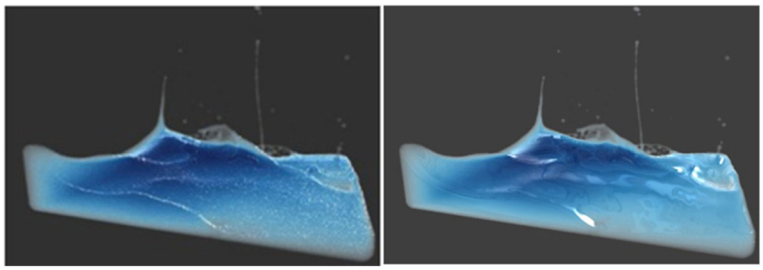

Fig. 2 The smoothed image by our filter. **a** Before smoothing. **b** After smoothing

normalization. It integrates two weight factors. $c(\xi, x)$ and $s(\xi, x)$ is defined respectively as

$$c(\xi, x) = e^{-\frac{1}{2}\left(\frac{d(\xi,x)}{\sigma_d}\right)^2} \tag{12}$$

$$s(\xi, x) = e^{\frac{1}{2}\left(\frac{\sigma(f(\xi),f(x))}{\sigma_d}\right)^2} \tag{13}$$

Equations (11) and (12) are infinite integration in the space. In this paper, we use the function of summation of its discrete condition, in a 4*4 pixel space as

$$h(x) = k^{-1}(x)\sum \Omega f(\xi)c(\xi, x)s(f(\xi), f(x)) \tag{14}$$

For making the comparison clearer, we render the smoothed image by our method. From the Fig. 2, we can find that image processed by our bilateral filter creates smoothed surface and protects droplets at the same time.

4.4 Integration of foreground and background mapping

After the geometric buffer is generated, we first use the bilateral buffer on its depth buffer to obtain a smooth surface, and then, we integrate foreground and background scenes by background mapping. At last, we add a step as post-production to enhance the fluid photo-realistic and output the result into the frame buffer. As is shown in Fig. 3, the arrows indicate sequences of output buffer in different stages.

Since the traveling of light in transparent medium is subject to Beer–Lambert law, monochromatic light, after passing through medium with thickness, it loses luminous strength since the medium absorbs part of its luminous energy. The denser the absorbing medium is, the medium is thicker, and the luminous loss is more obvious. The relation can be briefly expressed by

$$A = \lg\left(\frac{1}{T}\right) = K \cdot l \cdot c \tag{15}$$

where T stands for transmittance, K for absorption coefficient, l is the thickness, and c is the density of the light absorption object.

A vivid simulation of transparent fluid requires the thickness of the fluid. But the information generating fluid geometric buffer is in the screen surface; therefore, it cannot be recovered from the geometric buffer. We need to compute the fluid thickness on the viewing plane from 3D point cloud. In order to do so, we introduce point sprites again. Different from generation of

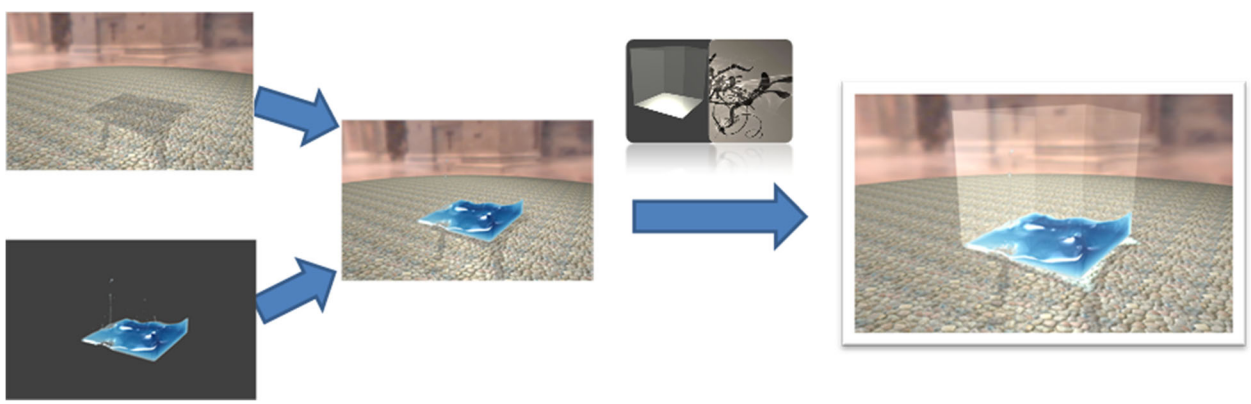

Fig. 3 Mapping process

geometric buffer, we turn off hardware depth detection and use GPU's alpha mixing function to acquire thickness. The weight is calculated by

$$T = N_z e^{-2r^2} \tag{16}$$

where N_z stands for normal vector's component on z axis,and r for radius of point sprites. In fragment, the acquired thickness is packed into 8-bit tri-channel frame buffer. Thickness value has little weight in shading; therefore, we use down sampling for thickness image generation. The frame buffer in 1/4 original resolution is used to output the thickness image. The original resolution is used for sampling in later integration. The linear interpolation of GPU can smooth depth buffer.

Foreground-background shading and integration are carried out in screen space. More specifically, background mapping and fluid geometric buffer structure are used in fragment buffer for foreground-background integration and fluid shading. First, for the pixel in question, we retrieve sample S from its corresponding geometric buffer. The sample will define property of pixel. If samples component on R, $S_R > 0$, then the pixel contains fluid information, and illumination on the pixel should be computed first before shading. On the contrary, the background color can be copied and output as the current pixel color. If the pixel in question has foreground information, we should also compute illumination before shading. Since bilateral filter has already done for fluid particle depth buffer, we should reconstruct $P_{\text{eye}}(Z)$ according to the formula as follows:

$$P_{\text{eye}} = \left(-uv(x,y)\left(f_x, f_y\right) z_{\text{eye}}, z_{\text{eye}}\right) \tag{17}$$

$z_{\text{eye}} = s_x$, uv (x,y) are pixel coordinates in the screen space. f_x and f_y are calculated by the frame buffer and field of view:

$$f_x = \tan(fov)^* \text{aspect}, f_y = \tan(fov) \tag{18}$$

At the same time, the reconstructed $P_{\text{eye}}(Z)$ is introduced to calculate the pixel's normal vector in the eye coordinates system. First, we set the calculated point as the center, as shown in Fig. 4a, and then we use difference operation of surrounding pixels depth information to acquire base of local coordinates, and finally, we apply multiplication cross to obtain the normal vector in the eye coordinates system. More specifically, we acquire surrounding pixel's sample in UDLR directions and reconstruct $P_{\text{eye}}(Z)$. The acquired four points can be then used to calculate the local bounding box, as shown in Fig. 4b:

$$L_{\text{bb}} = \min(x,y,z),\ \max(x,y,z) \tag{19}$$

Normal vector of the point is expressed in:

$$N_0 = \text{normalize}(L_{bb}(\min) \times L_{bb}(\max)) \tag{20}$$

The reconstructed P_{eye} and surface vector N_0 are introduced into Phong illumination model to calculate foreground color and output color value is as C_0. Thickness value T_0 is sampled from thickness mapping. The fluid property on the pixel in question is obtained in the fluid geometric buffer. The direction of refracted light, L_r is computed after two refractions by the method of paper [31]. The new color value C_b is sampled from background mapping according to L_r. At last, we use Eq. 21 to compute the final pixel color C and output the results to the frame buffer.

$$C = C_b e^{\frac{1}{T_0}} + aC_0 \tag{21}$$

5 Results

5.1 Framework of platform

In order to make full use of computing resources of CPU and GPU, we construct a mixing of CPU-GPU

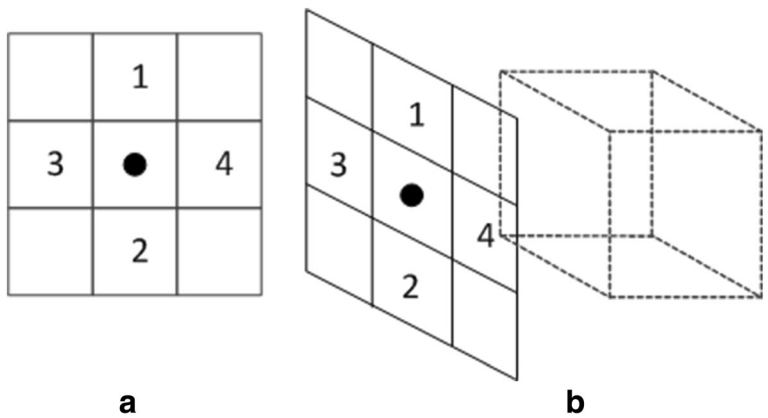

Fig. 4 The reconstruction of normal. **a** Centre point. **b** Local bounding box

heterogeneous physical accelerating architecture based on service clusters, which provide a convenient and efficient simulation platform for particle-based fluid. The parallel acceleration platform includes a multi-GPU cluster system. Each computing node includes four GPU devices. The details of platform are as follows:

(1) Multi-GPU clusters. The node machine uses Intel-based commercial server, and different nodes are linked with high-speed communication network. Each node contains multiple high-performance GPUs.
(2) Unified message layer. It encapsulates CUDA and OpenMP as a unified message passing layer. Message Passing Interface (MPI) is a popular parallel programming environment, which is used to transfer and control various data and information. In the CUDA programming model, the GPU as a coprocessor generates a large number of threads to solve the data that can be expressed as parallel data with a very high computational density. This layer is the basis for accelerating platform and is responsible for resource scheduling and computing parallelization.
(3) Engine of simulation and visualization. It includes two engines of physics computing and surface rendering. The engine uses the component architecture, and the specific algorithm is implemented by plug-ins, which ensures the robustness of the system and flexible scalability.
(4) Performance analysis. The platform developers can realize different physical simulation and rendering algorithms, while providing performance analysis function is to facilitate tracking and analysis of system performance.

Users in the parallel acceleration platform have two different roles: general users and system administrators, as shown in Fig. 5. The system administrator is the responsible for the general maintenance of system, network security, data confidentiality, software hardware upgrade, system deploy, and configuration. The platform can provide unified portal and resource of high performance for the general users to deploy their computing and simulations. From the perspective of general users, the parallel platform is consistent, and users can query the current system operation, such as the number of GPUs and available storage size. General users also can utilize the programming interface provided by the platform to build, submit, and enforce more effective and convenient computational resource for various applications. Furthermore, the platform can support various computational tasks by opening the underlying interface for special users.

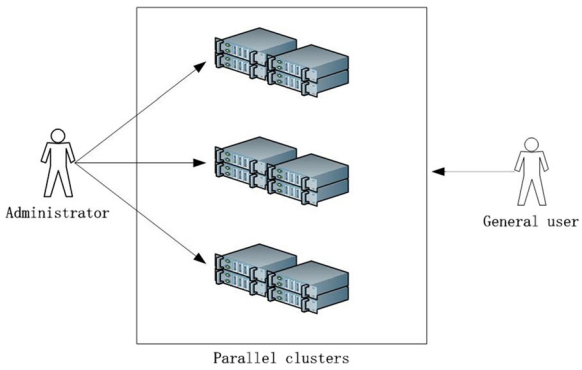

Fig. 5 The use case of platform

5.2 Interface design

SPH is a meshless Lagrangian method, which uses particle interpolation to calculate the fluid dynamics. The basic equations of fluid mechanics are transformed into the SPH equation, which can be evaluated by the kernel function. Particularly, the kernel function and interpolation method are the core concerns of SPH solution. Therefore, we integrate a variety of parallel implementation of physics kernel functions to meet the requirement of different users. With the increasing number of particles, it can be accelerated by computing resource of the GPU clusters. Although multi-GPU clusters can effectively solve the computational task of large-scale particles, the design and development of parallel program for the ordinary developers and researchers are difficult and still need to take time to learn. Therefore, in order to effectively utilize the platform with simplifying the development and deployment of the algorithm, we define the advanced development interface based on the unified message layer and we define the advanced development interface based on the unified message layer, which optimizes the particle

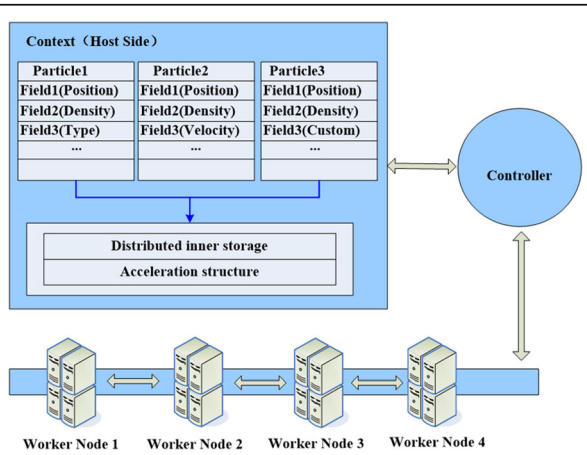

Fig. 6 The design of platform interface

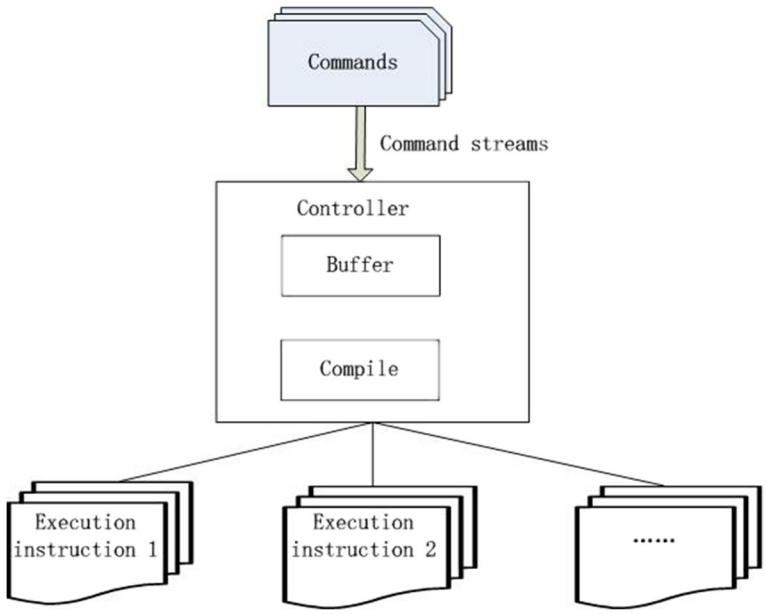

Fig. 7 Parallelize the serial instruction

simulation. Through the abstraction of the simulation process, the hardware details of the multi-GPU cluster are hidden so that the developer can focus on the development of the algorithm itself.

For the interface of platform, each program executes the SPH algorithm becoming a running instance as shown in Fig. 6, and each instance holds the current context (host context) of the current runtime. The client context maintains the delivery of the current application with the cluster message and the distribution of the data. In the design of the development interface, the client does not have to communicate with the nodes in the cluster, just obtain through the front and end of the controller, and the controller's role is equivalent to the proxy and interpreter, by interpreting the client's message asynchronous to complete the task of the establishment and distribution. Controller back-end automatically maintains work unit (Worker) with heterogeneous unified communication platform based on

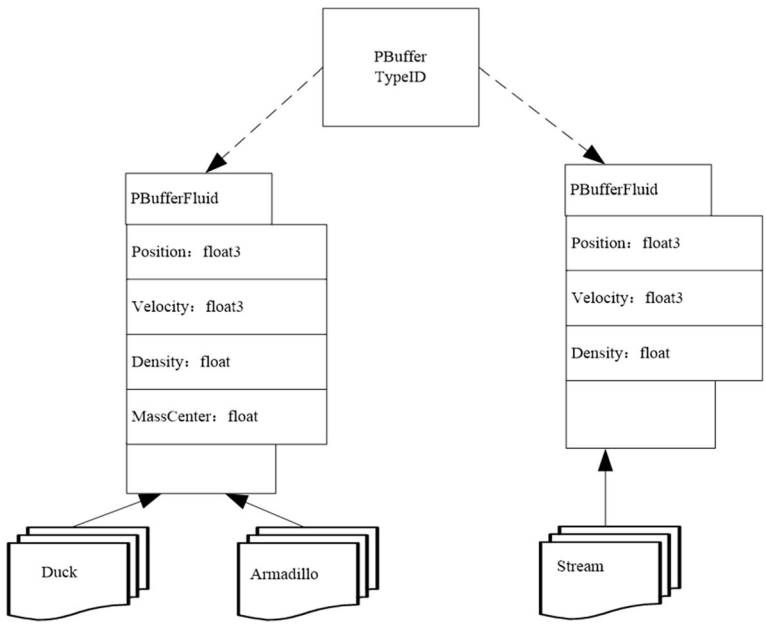

Fig. 8 The instance of fluid-solid PBuffer

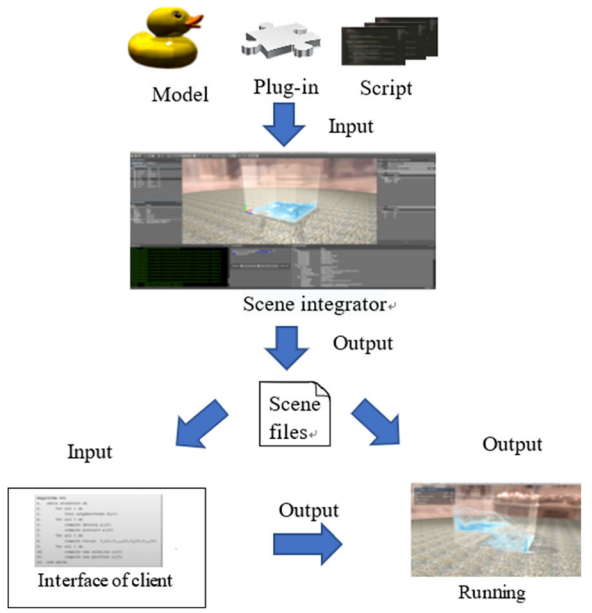

Fig. 9 The overview of platform tool

tasks' loads. Each worker can use its storage resource for the specific computing tasks. The communication and data exchange between the work units are also done by the unified communication platform.

The client API abstracts the SPH calculation process into large granularity commands, such as creating a particle (CREATE), setting a particle attribute (PROP), and creating an accelerated structure (CREATE_ACC). The client program uses the commands provided by the API to design the serial SPH algorithm. When the program is running, the client context will synchronize these commands and submit them to the controller. Then, the controller will interpret the instructions in the execution command stream buffer. For mapping the present serial instruction to parallel instruction of multi-GPU cluster, the computing units are dispatched by the unified communication platform, and the cell and controller can have two-way communications, as shown in Fig. 7.

5.3 Buffer design

In order to meet the wide requirements of fluid image effects, our framework concerns the particles of different properties. For example, the fluid-solid interaction coupling method includes two types of fluid and solid particles. In this case, an efficient mechanism for describing different particles is required. In addition, to hide the development details of the parallel program by users, the platform needs to provide a unified interface to manage the storage strategy. In view of the above problems, this paper presents the concept of particle buffer (PBuffer) to achieve the particle system management. Particle buffer is a storage model of logically different instances of a particle type, which are uniformly distributed by the system transferring to different computing nodes at runtime, this is, from the perspective of accelerating the platform, it is still the transmission of binary message. In this way, users can design different types of particles according to the idea of serial programs and allocate storage for their instances without the storage of parallel programs consideration. The acceleration platform also does not need to care about the type of particles, which maintains the computing resources by the logical relationship between the controller and the client.

From the perspective of the client, particle buffer is a container of particle properties. An undefined particle buffer only contains the property of TypeID, and user can define different particle types by adding various Slots during the initialization. The different type of particle buffer instances can be generated when the algorithm is executed. Taking the fluid-solid interaction scene as an example, the method defines two types of particles: PBufferSolid and PBufferFluid. Each particle is Slot such as Position, Velocity, and Density. The solid particle additionally includes the MassCenter and Temperature. Different type of particles generates different number of instances based on the simulated scenes, as shown in Fig. 8. The scene composes of instances of

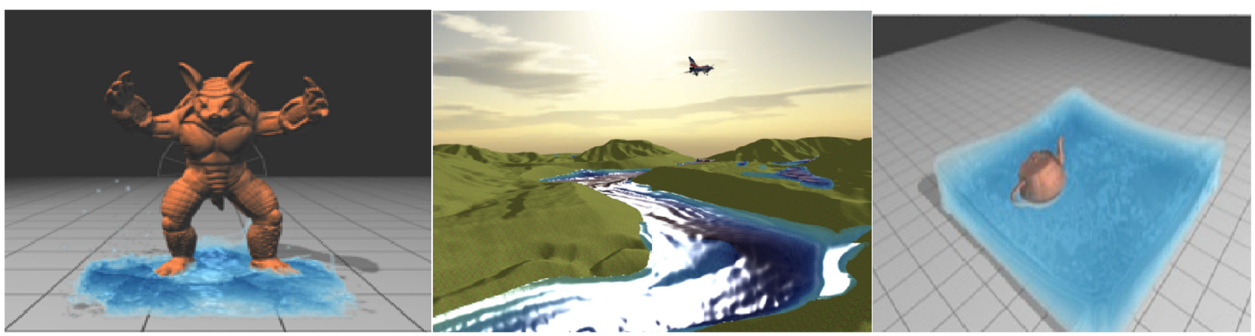

Fig. 10 The interaction of fluid-solid

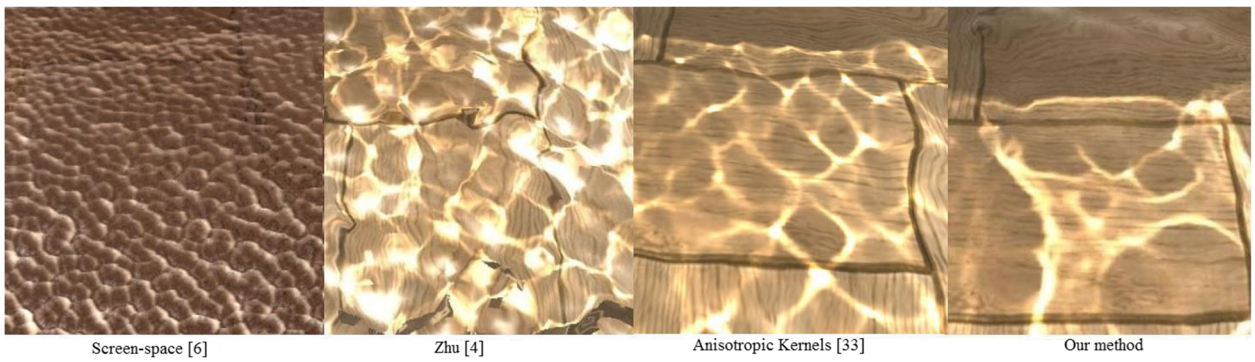

Fig. 11 Comparison of different surface reconstruction approaches

multiple solid particles such as Duck, Armadillo, and fluid instance of Stream. Finally, the CUDA script is submitted to compute the physical function, interaction model, and visualization method. The client context contains the logical structure of different types of particles defined by the developer through the interface. After the context of client is established, the system automatically establishes the mapping of internal distributed storage according to the type information and deploys the acceleration structure based on user's configuration. The storage is allocated by communication of the system controller and acceleration platform after the simulation starts.

The platform of simulation provides a complete tool chain to support the application development and experiment of art effects. It consists of scene editor, runtime player, and secondary development interface. The overview of platform tool is shown in Fig. 9. The user can import 3D model into the editor. We choose Collada as the intermediate format, an XML-based common 3D scene exchange file, which can be compiled into a binary format to speed up model loading and editing. Figure 10 shows that the effect of fluid-solid interaction is constructed by our system.

6 Discussions

In the section, we design some interactive fluid scenes to illustrate the visualization effect and performance. All results are generated on multi-GPU cluster platform with four worker nodes, each node has two GPUs, windows10 OS, NVIDIA Geforce GTX 480 GPU.

Figure 11 shows a comparison between screen-space method [4, 6], anisotropic kernel method [33] and our direct particle-based method. The scene used for comparison is a simulation of the armadillo break with 400K particles. As shown in the (Fig. 11), although the depth buffer is smoothed by screen-space techniques, the near perspective is rather rough. Zhu's approach gets smoother surfaces than above methods, but it still displays some bumpy surfaces and loses some small features. The method of anisotropic kernel [33] generates high quality surfaces, but its volume is reduced and small bumps become evident as soon as the surface is magnified. Our approach obtains much smoother surface than all the methods mentioned above.

Figure 12 shows a detail comparing of classic isotropic kernel surface reconstruction [34] with our method. As the figure shown, the traditional density field surface reconstruction method produces rough surfaces and loses many details of surface, while our method can well keep drops, ripples, and splashes obviously.

Apart from its detail-retaining advantage, particle-based SPH method exerts better performance than conventional surface reconstruction rendering. We compare our direct particle-based rendering (DPD), adapting surface reconstruction (ASR) [35], and conventional surface reconstruction (CSR) [34] on the same scene with different particle scales. The results are summarized in Table 3.

Figure 13 shows visual result executed on our platform. On the one hand, it is clear that the performance of our method is not sensitive to particle scales, whereas the execution time of adaptive method and conventional mesh method increases as the particle scale increases.

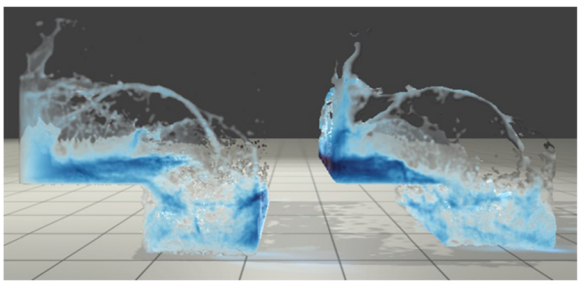

Fig. 12 Our direct particle-based rendering against isotropic kernel [34] (left, our direct rendering; right, MC)

Table 3 Performance comparison of the three methods

Time	52 K	71 K	102 K	152 K	231 K
DPD (ms)	8.3342	8.6535	9.8766	11.2793	12.5132
ASR (ms)	9.5802	13.5785	17.701	27.8085	34.3209
CSR (ms)	9.0147	12.4849	14.5257	25.0425	30.5764

The reason is adaptive method requires construction of scalar field from simulation particles before surface shading. On the other hand, this time-consuming computation is omitted in our method, which utilizes GPU hardware to screen large amount of invisible internal points. In this way, the increase of particle scale does not produce more visible pixels. Therefore, our method can remain sound and stable.

7 Conclusions

In this paper, we propose a fast framework construction and visualization method, which utilize programmable GPU to realize the rendering and constructing for wide application on art fluid effects. A direct surface visualization pipeline for particle sets is designed based on screen-space method, which render fluid surface by visible particles in subpixels. For obtaining smooth surface, a new filtering model is designed, instead of Gaussian filter; the bilateral filter is used for depth image to create smooth fluid. Our method does not need conventional surface reconstruction steps and can protect details of fluid. Furthermore, we propose a fast constructing framework on the multi-GPU platform. An efficient framework is implemented based on the features of particle method. The framework can automatically establish the system mapping according to the requirement of user, which guarantees that the operation of application is fast and friendly. However, some problems exist in our framework, and they may be addressed in the future work. Our following work is to enhance the platform function, making more complex and realistic art fluid effects such as multi-fluids interaction and multi-phase fluids in artistic effects.

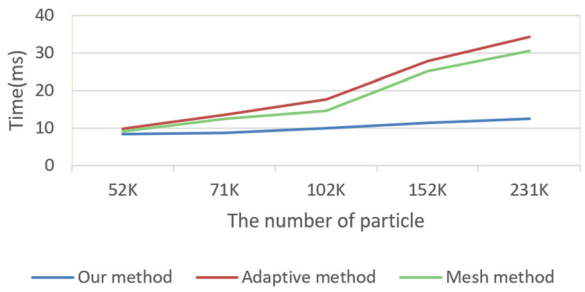

Fig. 13 Comparison of the three rendering methods

Funding

This paper is supported by the National Natural Science Foundation of China (no. 61402016, no. 61502094), Ministry of Education Humanities and Social Sciences Foundation (no. 14YJCZH200), Beijing Natural Science Foundation (no. 4154067), Research Plan of Beijing (no. KM201610009008), Youth Talent project of Beijing (no. 2016000026833ZK09), and NCUT Foundation of (no. XN018001).

Availability of data and materials

The data will be shared on Fengquan zhang personal website soon.

Authors' contributions

The work presented in this paper was carried out in collaboration between all authors. FZ and ZW carried out the main part of this manuscript. JC and JZ is a supervisor of this research. FT has assisted in the experimental part of the work. All authors read and approved the final manuscript.

Authors' information

Zhaowei Wang, female, School of Mechanical Engineering, University of Science and Technology Beijing. Her main research interests are computer vision, pattern recognition, affective computing, and interdisciplinary research across computer science and art design. Jingyan Qin, female, Professor, PhD. degree, School of Mechanical Engineering, University of Science and Technology Beijing. Her main research interests are interaction design, information design, and information visualization on big data. Fengquan Zhang is currently an Associate Professor at the North China University of Technology. He received the Ph.D. degree in computer science from the State Key Laboratory of Virtual Reality Technology and Systems, Beihang University. His research interests concern on physical-based simulation, image processing, computer animation, rendering, and interactive game. Moreover, recent projects mainly comprise the interaction animation and game using OptiTrack, Kinect, Leap motion, and Web/Depth camera. Zhaowei Wang is currently a master's student at the North China University of Technology, China. His research interests include image processing, pattern recognition, computer graphic, and visualization.
Jian Chang is an associate professor at the National Centre for Computer Animation, Bournemouth University. He received his PhD degree in computer graphics in 2007 at the National Centre for Computer Animation, Bournemouth University. His research focuses on a number of topics relating to geometric modeling, algorithmic art, character rigging and skinning, motion synthesis, deformation, and physically based animation. He also has strong interest in applications in medical visualization and simulation.
Jian Zhang is currently a professor of computer graphics at the National Centre for Computer Animation, Bournemouth University, UK, and leads the Computer Animation Research Centre. He is also a cofounder of the UK's Centre for Digital Entertainment, funded by the Engineering and Physical Sciences Research Council. His research focuses on a number of topics relating to 3D virtual human modeling, animation, and simulation, including geometric modeling, rigging and skinning, motion synthesis, deformation, and physics-based simulation.
Feng Tian received the Ph.D. degree in computer science from the State Key Laboratory of Virtual Reality Technology and Systems, Beihang University. He is currently an Associate Professor at the Northeast Petroleum University. His research interests concern on pattern recognition, image processing, computer vision, and artificial intelligence.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details

[1]Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, North China University of Technology, Beijing, China. [2]National Centre for Computer Animation, Bournemouth University, Poole, UK. [3]School of Computer and Information Technology, Northeast Petroleum University, Daqing, China.

Received: 27 August 2017 Accepted: 12 November 2017
Published online: 01 December 2017

References

1. T Weaver, Z Xiao, Fluid simulation by the smooth particle hydrodynamics method: a survey. Int. Conf. Comput. Gaph. Theory. Appl., 1–11 (2016)
2. M Ihmsen, J Orthmann, B Solenthaler, et al., SPH fluid in computer graphics. Eurographics Starsbourg, 21–42 (2014)
3. WE Lorensen, HE Cline, Marching cubes: a high resolution 3D surface construction algorithm. ACM SIGGRAPH Comput. Graph., **21**(4):163-169 (1987)
4. Y Zhu, R Bridson, Animating sand as a fluid. Transac Graph ACM. **24**, 965–972 (2005)
5. B Solenthaler, YC Zhang, R Pajarola, Efficient refinement of dynamic point data, IEEE/Eurographics Symposium on Point-Based Graphics, 65-72 (2007)
6. J Wladimir, S Green, M Sainz, Screen space fluid rendering with curvature flow. Proceeding of Symposium on Interactive 3D Graphics and Games, 91–99 (2009)
7. J Bender, D Koschier, Divergence-free SPH for incompressible and viscous fluids. IEEE Trans. Vis. Comput. Graph. **23**(3), 1193–1206 (2017)
8. B Ren, X Yan, CF Li, T Yang, MC Lin, SM Hu, Fast SPH simulation for gaseous fluids. Vis. Comput. **32**(4), 523–534 (2016)
9. X Shao, Z Zhou, J Zhang, W Wu, Realistic and stable simulation of turbulent details behind objects in SPH. J. Comput. Animation. Virt. W. **26**(1), 79–94 (2015)
10. L Fernando, M Sandim, F Petronetto, et al., Particle-based fluids for viscous jet buckling. Comput. Graph. **52**(C), 106–115 (2015)
11. J Bender, D Koschier, Divergence-free smoothed particle hydrodynamics. Proceedings ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Los Angeles, 147–155 (2015)
12. T Takahashi, MC Lin, A multilevel SPH solver with unified solid boundary handling. Computer Graphics Forum, Online. **35**(7), 517–526 (2016)
13. M Sandim, D Cedrim, L Gustavo, et al., Boundary detection in particle-based fluids. Comput. Graph. Forum **35**(2), 1–10 (2016)
14. T Takahashi, Y Dobashi, T Nishita, MC. Lin. An efficient hybrid incomprehensible SPH solver with interface handling for boundary conditions. Version of record online: 6 September, (2017)
15. T Yang, MC Lin, RR Martin, J Chang, S-M Hu. Versatile interactions at interfaces for SPH-based simulations. Proceedings ACM SIGGRAPH/ Eurographics Symposium on Computer Animation, Zurich. 57–66, (2016)
16. D Goes, C Wallez, J Huang, et al., Power particles: an incompressible fluid solver based on power diagrams. ACM Trans. Graph. **34**(4), 50 (2015)
17. BD Rogers, D Valdez, New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. Comput. Phys. Commun. **184**, 1848–1860 (2013)
18. M Harris, Fast fluid dynamics simulation on the GPU. GPU Gems, 637–665 (2004)
19. T Harada, S Koshizuka, Y Kawaguchi. Smoothed particle hydrodynamics on GPUs. Proceedings of Computer Graphics International, 63–70, (2007)
20. A Herault, G Bilotta, RA Dalrymple, SPH on GPU with CUDA. J. Hydraul. Res. **48**, 74–79 (2010)
21. J Cornelis, M Ihmsen, A Peer, M Teschner, IISPH-FLIP for incompressible fluids. Comput. Graph. Forum **33**(2), 255–262 (2014)
22. R Bridson. Fluid simulation for computer graphics. (A KPeters/CRC Press, Boca Raton, 2015)
23. F Zhang, X Shen, X Long, et al., A particle model for fluid simulation on the multi-graphics processing unit. Int. J. Numer. Model. **26**, 397–414 (2013)
24. E Rustico, G Bilotta, A Herault, C Negro, Advances in multi-GPU smoothed particle hydrodynamics simulations. IEEE Transc. Parallel Distributed Syst. **25**(1), 43–53 (2014)
25. L Shi, G Chen, W Cao, et al, Analysis enhanced particle-based flow visualization, International Symposium on Electronic Image, Vis. Data. Anal. 12-21 (2017)
26. R Yasuda, T Harada, Y Kawaguchi, Fast rendering of particle-based fluid by utilizing simulation data, Proceedings of Eurographics short paper, 61–64 (2009)
27. S Premoze, T Tasdizen, J Bigler, et.al, Particle-based simulation of fluids, Proceedings of Eurographics, 401–410 (2003)
28. G Guennebaud, M Gross, Algebraic point set surfaces, Proceedings of ACM SIGGRAPH Papers. 23, (2007)
29. F Zhang, X Shen, X Long, An adaptvie model for particle fluid surface reconstruction. IEICE Trans. Inf. Syst. **5**, 1247–1249 (2013) vol. E96-D
30. N Akinci, A Dippel, G Akinci, et al, Screen space foam rendering, Proceedings of Computer Graphics, Visualization and Computer Vision, 1–10 (2012)
31. C Wyman, G Nichols, Adaptive caustic maps using deferred shading, journal of compute graphic. Forum **28**(2), 309–318 (2009)
32. T Scott, C Wyman, Interactive refractions with total internal reflection, Proceedings of Graphics Interface, Montreal, Canada, 185–9190 (2007)
33. J Yu, G Turk, Reconstructing surfaces of particle-based fluids using. Eurographics/ACM Siggraph Symposium on Computer Animation, 1–10 (2010)
34. M Muller, D Charypar, M Gross, Particle-based fluid simulation for interactive applications. In Symposium on Computer Animation, 154–159 (2003)
35. Y Zhang, B Solenthaler, R Pajarola, Adaptive sampling and rendering of fluids on the GPU. IEEE/EG Symposium on Volume and Point-Based Graphics, 1–10 (2008)