**The Visual Computer manuscript No.**
(will be inserted by the editor)

**Takahiro Harada · Seiichi Koshizuka · Yoichiro Kawaguchi**

# Smoothed Particle Hydrodynamics on GPUs

**Abstract** In this paper, we present a Smoothed Particle Hydrodynamics (SPH) implementation algorithm on GPUs. To compute a force on a particle, neighboring particles have to be searched. However, implementation of a neighboring particle search on GPUs is not straightforward. We developed a method that can search for neighboring particles on GPUs, which enabled us to implement the SPH simulation entirely on GPUs. Since all of the computation is done on GPUs and no CPU processing is needed, the proposed algorithm can exploit the massive computational power of GPUs. Consequently, the simulation speed is many times increased with the proposed method.

**Keywords** Fluid simulation · Particle method · Smoothed Particle Hydrodynamics · Graphics hardware · General Purpose Computation on GPUs

## 1 Introduction

At the present time, physically based simulation is widely used in computer graphics to produce animations. Even the complex motion of fluid can be generated by simulation. There is a need for faster simulation in real-time applications, such as computer games, virtual surgery and so on. However, the computational burden of fluid simulation is high, especially when we simulate free surface flow, and so it is difficult to apply fluid simulation to real-time applications. Thus real-time simulation of free surface flow is an open research area.

In this paper, we accelerated Smoothed Particle Hydrodynamics (SPH), which is a simulation method of free surface flow, by using of Graphics Processing Units (GPUs). No study has so far accomplished acceleration of particle-based fluid simulation by implementing the entire algorithm on GPUs. This is because a neighboring particle search cannot be easily implemented on GPUs. We developed a method that can search for neighboring particles on GPUs by introducing a three-dimensional grid. The proposed method can exploit the computational power of GPUs because all of the computation is done on the GPU. As a result, SPH simulation is accelerated drastically and tens of thousands of particles are simulated in real-time.

T.Harada
7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan
Tel.: +81-3-5841-5936
E-mail: takahiroharada@iii.u-tokyo.ac.jp

S. Koshizuka and Y.Kawaguchi
7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan

## 2 Related Works

Foster *et al.* introduced the three-dimensional Navie-Stokes equation to the computer graphics community[8] and Stam *et al.* also introduced the semi-Lagrangian method[34]. For free surface flow, Foster *et al.* used the level set method to track the interface[7]. Enright *et al.* developed the particle-level set method by coupling the level set method with lagrangian particles[6]. Bargteil *et al.* presented another surface tracking method that used explicit and implicit surface representation[2]. Coupling of fluid and rigid bodies[3], viscoelastic fluid[9], interaction with thin shells[10], surface tension, vortex particles, coupling of two and three-dimensional computation[14], octree grids[25] and tetrahedron meshes[17] have been studied.

These studies used grids, but there are other methods that can solve the motion of a fluid. These are called particle methods. Moving Particle Semi-implicit (MPS) method[20] and Smoothed Particle Hydrodynamics (SPH) [27] are particle methods that can compute fluid motion. Premoze *et al.* introduced the MPS method, which realized incompressibility by solving the Poisson equation on particles and has been well studied in areas such as computational mechanics, to the graphics community[32]. Müller *et al.* applied SPH, which had been developed in astronomy, for fluid simulation[28]. They showed that SPH could be applied for interactive applications and used a few thousand of particles. However, the number

was not enough to obtain sufficient results. Viscoelastic fluids[4], coupling with deformable bodies[29] and multi-phase flow[30] were also studied. Kipfer *et al.* accelerated SPH using a data structure suitable for sparse particle systems[16].

The growth of the computational power of GPUs, which are designed for three-dimensional graphics tasks, has been tremendous. Thus there are a lot of studies that use GPUs to accelerate non-graphic tasks, such as cellular automata simulation[13], particle simulation[15] [19], solving linear equations[21] and so on. We can find an overview of these studies in a review paper[31]. Also, there were studies on the acceleration of fluid simulation, i.e., simplefied fluid simulation and crowd simulation[5], two-dimensional fluid simulation[11], three-dimensional fluid simulation[23], cloud simulation[12] and Lattice Boltzmann Method simulation[22]. Amada *et al.* used the GPU for the acceleration of SPH. However, they could not exploit the power of the GPU because the neighboring particle search was done on CPUs and the data were transferred to GPUs at each time step[1]. Kolb *et al.* also implemented SPH on the GPU[18]. Although their method could implement SPH entirely on the GPU, they suffered from interpolation error because physical values on the grid were computed and those at particles were interpolated. A technique for neighboring particle search on GPUs is also found in [33]. They developed a method to generate a data structure for finding nearest neighbors called stencil routing. A complicated texture footprint have to be prepared in advance and it needs a large texture when it applied to a large computation domain because a spatial grid is represented by a few texels.

There have been few studies on the acceleration of free surface flow simulation using GPUs.

## 3 Smoothed Particle Hydrodynacimcs

### 3.1 Governing Equations

The governing equations for incompressible flow are the mass conservation equation and the momentum conservation equation

$$\frac{D\rho}{Dt} = 0 \tag{1}$$

$$\frac{D\mathbf{U}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{U} + \mathbf{g} \tag{2}$$

where $\rho, \mathbf{U}, P, \nu, \mathbf{g}$ are density, velocity, pressure, dynamic viscosity coefficient of the fluid and gravitational acceleration, respectively.

### 3.2 Discretization

In SPH, a physical value at position $\mathbf{x}$ is calculated as a weighted sum of physical values $\phi_j$ of neighboring particles $j$

$$\phi(\mathbf{x}) = \sum_j m_j \frac{\phi_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j) \tag{3}$$

where $m_j, \rho_j, \mathbf{x}_j$ are the mass, density and position of particle $j$, respectively and $W$ is a weight function.

The density of fluid is calculated with eqn.3 as

$$\rho(\mathbf{x}) = \sum_j m_j W(\mathbf{x} - \mathbf{x}_j). \tag{4}$$

The pressure of fluid is calculated via the constitutive equation

$$p = p_0 + k(\rho - \rho_0) \tag{5}$$

where $p_0, \rho_0$ are the rest pressure and density, respectively.

To compute the momentum conservation equation, gradient and laplacian operators, which are used to solve the pressure and viscosity forces on particles, have to be modeled. The pressure force $\mathbf{F}^{press}$ and the viscosity force $\mathbf{F}^{vis}$ are computed as

$$\mathbf{F}_i^{press} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W_{press}(\mathbf{r}_{ij}) \tag{6}$$

$$\mathbf{F}_i^{vis} = \nu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla W_{vis}(\mathbf{r}_{ij}) \tag{7}$$

where $\mathbf{r}_{ij}$ is the relative position vector and is calculated as $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ where $\mathbf{r}_i, \mathbf{r}_j$ are the positions of particles $i$ and $j$, respectively.

The weight functions used by Müller *et al.* are also used in this study[28]. The weight functions for the pressure, viscosity and other terms are designed as follows.

$$\nabla W_{press}(\mathbf{r}) = \frac{45}{\pi r_e^6}(r_e - |\mathbf{r}|)^3 \frac{\mathbf{r}}{|\mathbf{r}|} \tag{8}$$

$$\nabla W_{vis}(\mathbf{r}) = \frac{45}{\pi r_e^6}(r_e - |\mathbf{r}|) \tag{9}$$

$$W(\mathbf{r}) = \frac{315}{64\pi r_e^9}(r_e^2 - |\mathbf{r}|^2)^3 \tag{10}$$

This value is 0 outside of the effective radius $r_e$ in these functions.

### 3.3 Boundary Condition

#### *3.3.1 Pressure term*

Because the pressure force leads to the constant density of fluid when we solve incompressible flow, it retains the distance $d$ between particles, which is the rest distance. We assume that particle $i$ is at the distance $|\mathbf{r}_{iw}|$ to the

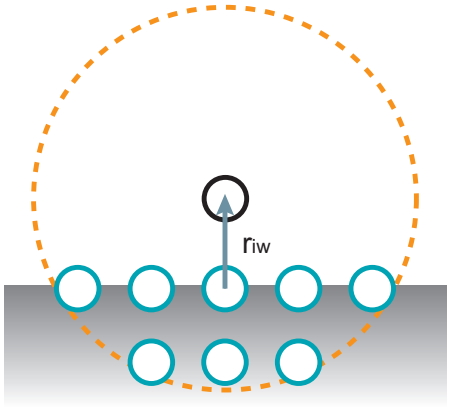


**Fig. 1** Distribution of wall particles.

wall boundary and $|\mathbf{r}_{iw}| < d$. The pressure force pushes particle $i$ back to the distance $d$ in the direction of $\mathbf{n}(\mathbf{r}_i)$ which is the normal vector of the wall boundary. Thus, the pressure force $\mathbf{F}_i^{press}$ is modeled as

$$\mathbf{F}_i^{press} = m_i \frac{\Delta \mathbf{x}_i}{dt^2} = m_i \frac{(d - |\mathbf{r}_{iw}|)\mathbf{n}(\mathbf{r}_i)}{dt^2}. \quad (11)$$

*3.3.2 Density*

When a particle is within the effective radius $r_e$ to the wall boundary, the contribution of the wall boundary to the density has to be estimated. If wall particles are generated within the wall boundary, their contribution as well as those of fluid particles can be calculated.

$$\rho_i(\mathbf{r}_i) = \sum_{j \in fluid} m_j W(\mathbf{r}_{ij}) + \sum_{j \in wall} m_j W(\mathbf{r}_{ij}) \quad (12)$$

The distribution of wall particles is determined uniquely by assuming that they are distributed perpendicular to the vertical line to the wall boundary and the mean curvature of the boundary is 0 as shown in fig.1. Therefore, the contribution of the wall boundary is a function of the distance $|\mathbf{r}_{iw}|$ to the wall boundary.

$$\rho_i(\mathbf{r}_i) = \sum_{j \in fluid} m_j W(\mathbf{r}_{ij}) + Z_{wall}^{rho}(|\mathbf{r}_{iw}|) \quad (13)$$

We call $Z_{wall}^{rho}$ as *wall weight function*. Since this wall weigh function depends on the distance $|\mathbf{r}_{iw}|$, it can be precomputed and referred in the fluid simulation. Precomputation of the wall weight function can be done by placing wall particles and adding their weighted values. This function is computed in advance at a few points within the effective radius $r_e$ and the function at an arbitrary position is calculated by linear interpolation of them.

To obtain the distance to the wall boundary, we have to compute the distance from each particle to all the polygons belonging to the wall boundary and select the minimum distance. Since this operation is computationally expensive, a distance function is introduced. The distance to the wall boundary is computed in advance and stored in a texture.

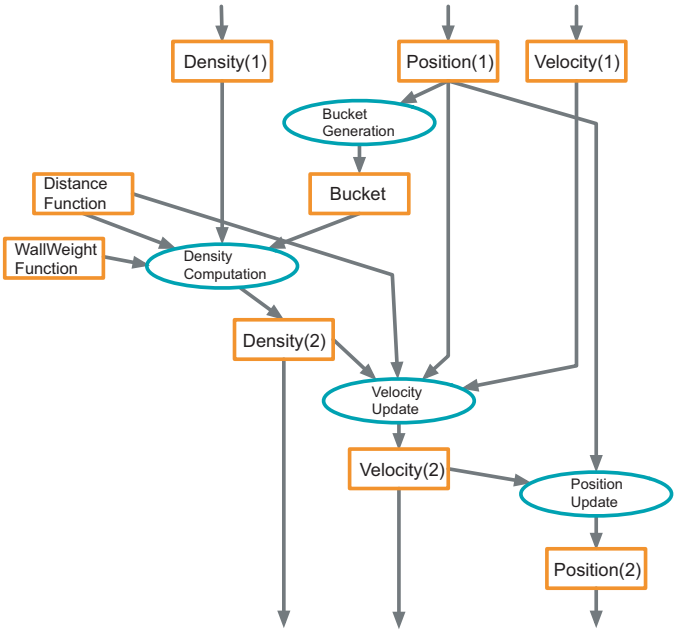


**Fig. 2** Flow char of one time step. Green ellipsoids represents operations (shader programs) and orange rectangles represents data (textures).

### 3.4 Neighbor Search

The computational cost of searching for neighboring particles is high when a large number of particles are used. To reduce the computational cost, a three-dimensional grid covering the computational region, called a bucket, is introduced as described by Mishira *et al.*[26]. Each voxel encoded as a pixel in the texture is assigned $d^3$ computational space. Then, for each particle, we compute a voxel to which the particle belongs and store the particle index in the voxel. With the bucket, we do not have to search for neighboring particles of particle $i$ from all particles because neighboring particles of particle $i$ are in the voxels which surrounding the voxel to which it belongs.

## 4 Smoothed Particle Hydrodynamics on GPUs

### 4.1 Data Structure

To compute SPH on GPUs, physical values are stored as textures in video memories. Two position and velocity textures are prepared and updated alternately.

A bucket texture and a density texture are also prepared. Although a bucket is a three-dimensional array, current GPUs cannot write to a three-dimensional buffer directly. Therefore, we employed a flat 3D texture in which a three-dimensional array is divided into a set of two-dimensional arrays and is then placed in a large texture. The detailed description can be found in [12]. As well as these values updated in every iteration, static values are stored in textures. The wall weight function is stored in a one-dimensional texture and a distance function which is a three-dimensional array is stored in a three-dimensional texture. The reason why a flat 3D texture is not employed is that the distance function is not updated during a simulation.

### 4.2 Algorithm Overview

One time step of SPH is performed in four steps.

1. Bucket Generation
2. Density Computation
3. Velocity Update
4. Position Update

The implementation detail is described in the following subsections. The flowchart is shown in Figure 2.

### 4.3 Bucket Generation

Since the number of particle indices stored in a voxel is not always one, a bucket cannot be generated correctly by parallel processing of all particle indices. This is because we have to count the number of particle indices stored in a voxel if there are multiple particle indices in the voxel.

Assume that the maximum particle number stored in a voxel is one. The bucket is correctly generated by rendering vertices which are made correspondence to each particles at the corresponding particle center position. The particle indices are set to the color of the vertices. This operation can be performed using vertex texture fetch. However, the operation cannot generate a bucket correctly in the case where there is more than one particle index in a voxel.

The proposed method can generate a bucket that can store less than four particle indices in a voxel. We assume that particle indices $i_0, i_1, i_2, i_3$ are stored in a voxel and they are arranged as $i_0 < i_1 < i_2 < i_3$. The vertices corresponding to these particles are drawn and processed in ascending order of the indices[15]. We are going to store the indices $i_0, i_1, i_2, i_3$ in the red, green, blue and alpha (RGBA) channels in a pixel, respectively by four rendering passes. In each pass, the particle indices are stored in ascending order. Color mask, depth buffer and stencil buffer are used to generate a bucket correctly.

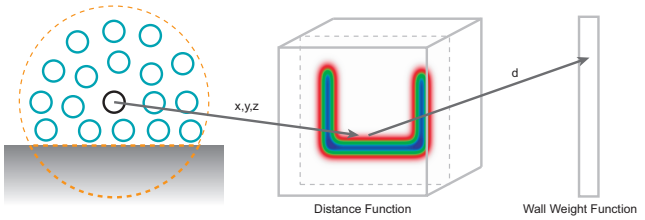


**Fig. 3** Estimation of the contribution of wall boundary. The distance to the wall $d$ is read from the distance function texture with particle position $x, y, z$. Then the density of wall boundary is read from the wall weight texture.

The vertex shader moves the position of a vertex to the position of the corresponding particle. Then the fragment shader outputs the particle index as the color and depth value. These shaders are used in all of the following rendering passes.

In the first pass, index $i_0$ is written in the R channel. By setting the depth test to pass the lower value and masking the GBA channels, $i_0$ can be rendered in the R channel at last. In the second pass, index $i_1$ is written in the G channel. The RBA channels are masked. The depth buffer used in the first pass is also used in this pass without clearing. Then the depth test is set to pass a greater value. However, the value rendered at last is $i_3$ which is the maximum depth value. To prevent writing $i_2$ and $i_3$ to this pixel, the stencil test is introduced. This stencil test is set to pass if the stencil value is greater than one. The stencil function is set to increment the value. Because the stencil value is 0 before rendering $i_1$, the vertex corresponding to $i_1$ can pass the stencil test and is rendered. However, the stencil value is set to 1 after $i_1$ is rendered and vertices corresponding to $i_2, i_3$ cannot pass the stencil test. Index $i_1$ is stored in the G channel at last. Indices $i_2$ and $i_3$ are written in the B and A channels in the third and fourth passes, respectively. The operations are the same as in the second pass other than the color mask. The RGA channels are masked in the third pass and the RGB channels are masked in the fourth pass. Since the depth value of $i_3$ is maximum among these vertices, the stencil test is not required in the last pass.

### 4.4 Density Computation

To compute the density of each particle, Equation (3) has to be calculated. The indices of neighboring particles of particle $i$ can be found with the generated bucket texture. Using the index of the particle, the position can be read from the position texture. Then the density of particle $i$ is calculated by the weighted sum of mass of the neighboring particles, which is then written in the density texture. If the particle is within the effective radius to the wall boundary, the contribution of the wall to the density have to be estimated in two steps. In the first

step, the distance from particle $i$ to the wall is looked up from the distance function stored in a three-dimensional texture. Then the wall weight function stored in a one-dimensional texture is read with a texture coordinate calculated by the distance to the wall. This procedure is illustrated in Figure 3. The contribution of wall boundary to the density is added to the density calculated among particles.

### 4.5 Velocity Update

To compute the pressure and viscosity forces, neighboring particles have to be searched for again. The procedure is the same as that for the density computation. These forces are computed using Equations (6) and (7). The pressure force from the wall boundary is computed using the distance function. Then, the updated velocity is written in another velocity texture.

### 4.6 Position Update

Using the updated velocity texture, the position is calculated with an explicit Euler integration.

$$\mathbf{x}_i' = \mathbf{x}_i + \mathbf{v}_i dt \tag{14}$$

where $\mathbf{x}_i$ and $\mathbf{v}_i$ are the previous position and velocity of particle $i$, respectively. The updated position $\mathbf{x}_i'$ is written to another position texture. Although there are higher order schemes, they were not introduced because we did not encounter any stability problems.

## 5 Results and Discussion

The proposed method was implemented on a PC with a Core 2 X6800 2.93GHz CPU, 2.0GB RAM and a GeForce 8800GTX GPU. The programs were written in C++ and OpenGL and the shader programs were written in C for Graphics.

In Figures 4 and 5 we show real-time simulation results. Approximately 60,000 particles were used in both simulations and they ran at about 17 frames per second. Previous studies used several thousand particles for real-time simulation, However, the proposed method enabled us to use 10 times as many particles as before in real-time simulation. The simulation results were rendered by using pointsprite and vertex texture fetch. The color of particles indicates the particle number density. Particles with high density are rendered in blue color and those with low density are rendered in white. The proposed method can accelerate offline simulation as well as real-time simulations. In Figures 6, 7 and 8, the simulated particle positions are read back to the CPU and rendered with a raytracer after the simulations. Surfaces of fluid is extracted from particles by using Marching

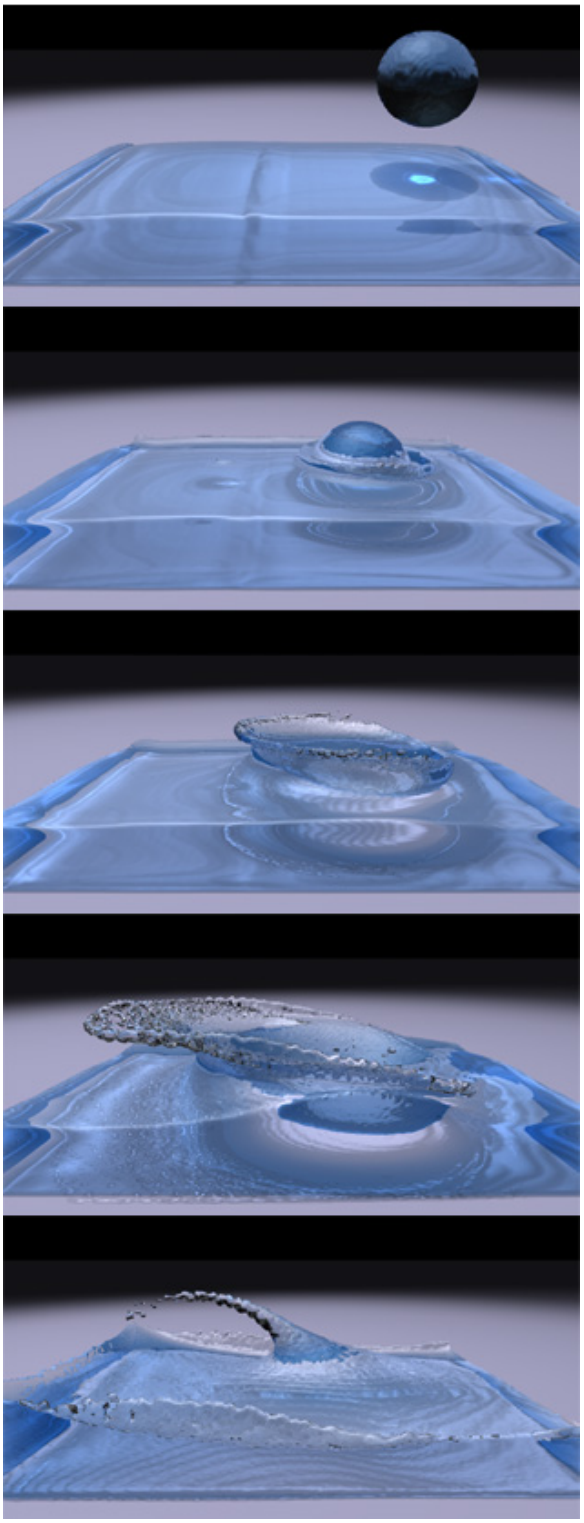

**Fig. 6** A ball of fluid is thrown into a tank.

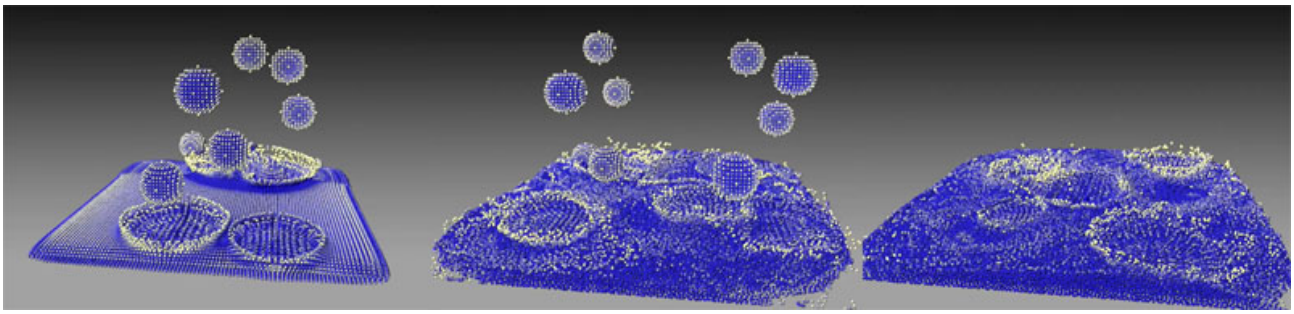

**Fig. 4** An example of real-time simulation. Balls of fluid is fallen into a tank.

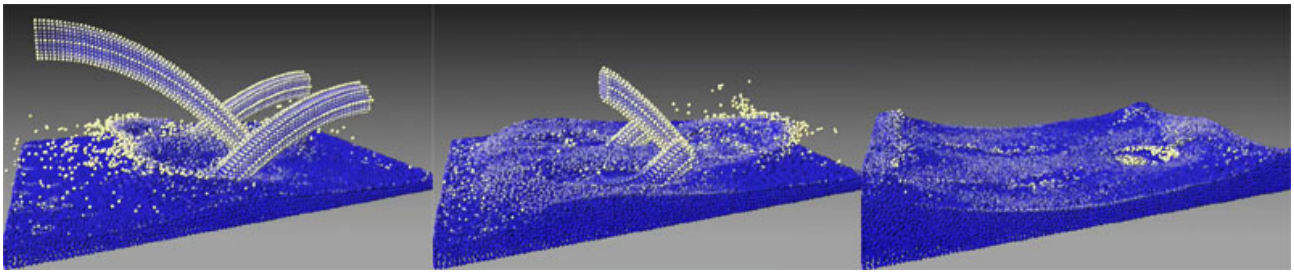

**Fig. 5** An example of real-time simulation. A fluid is poured into a tank.

Cubes[24]. A ball of liquid is thrown into a tank in Figure 6 and balls are fallen into a tank in Figure 7. In Figure 8, a liquid is poured into a tank. Approximately 1,000,000 particles were used in these simulations. They took about 1 second per one time step.

The computation times are measured by varying the total number of particles. Times (a) and (b) in Table 1 are the computation times for bucket generation and for one simulation time step including rendering time. These computation times were measured with rendering algorithms as shown in Figures 4 and 5. We can see that one time step is completed in 58.6 milliseconds for a simulation with 65,536 particles. We can also see that the ratio of the bucket generation time is low and most of the computation time is spent in density and force computations. We need to search for neighboring particles in the computation of the density and forces. In these computations, particle indices in the buckets surrounding a bucket in which particle $i$ is stored are looked up and then particle positions are also read from the texture using these particle indices. Since these computations are accompanied by a lot of texture look up with texture coordinates which are calculated with a value read from a texture, the computational costs are higher than for other operations.

We also implemented the method on the CPU and measured the computation times. Table 2 shows the results and the speed increase of the proposed method in comparison with the computation on the CPU. The computational speed on the GPU is about 28 times faster in the largest problem. When we used a small number of particles, the speed increase of the method was not so great. However, as the total number of particles increased, the efficiency of the proposed method increased.

The proposed method for bucket generation can generate a bucket correctly when the maximum number of particle stored in a voxel is less than four. Particles are placed as the simple cube structure and this state is used the rest state. This particle distribution is less dense than body-centered structure in which two particles are belongs to a voxel whose length of a side is the particle diameter. This indicates that there are less than two particles if the fluid keeps the rest density. When incompressible flow is solved by a particle method, fluid particles does not get much particle number density than the rest state. Therefore, if we solve incompressible flow, there are less than two particles in a voxel. However, since SPH solves not incompressible flow but near incompressible flow, a fluid can be compressed in varying degrees. So there is a possibility of packing more particles than the rest state. Overflow of particles in a voxel must causes artifacts. We can deal with the problem by preparing another bucket texture and storing fifth or later particles in this bucket. Because we could obtain plausible results with one bucket texture, another bucket texture was not introduced.

To accelerate SPH simulation by using of the GPU, all variables have to be stored in video memories. Therefore there is a limitation of the proposed method with respect to the total number of particles. Figure 9 shows a dam break simulation with 4,194,304 particles. This simulation needs approximately 600 MB memories. If

**Table 1** Times (a) and (b) are the computation times for bucket generation and computation time for one time step including the rendering time (in milliseconds).

| Number of particles | Time (a) | Time (b) |
|---|---|---|
| 1,024 | 0.75 | 3.9 |
| 4,096 | 0.80 | 5.45 |
| 16,386 | 1.55 | 14.8 |
| 65,536 | 3.99 | 58.6 |
| 262,144 | 14.8 | 235.9 |
| 1,048,576 | 55.4 | 1096.8 |
| 4,194,304 | 192.9 | 3783.6 |

**Table 2** Computation time on CPUs (in milliseconds) and speed increase of the proposed method (times faster).

| Number of particles | Time | Speed increase |
|---|---|---|
| 1,024 | 15.6 | 4.0 |
| 4,096 | 43.6 | 8.0 |
| 16,386 | 206.2 | 13.9 |
| 65,536 | 1018.6 | 17.3 |
| 262,144 | 6725.6 | 28.5 |

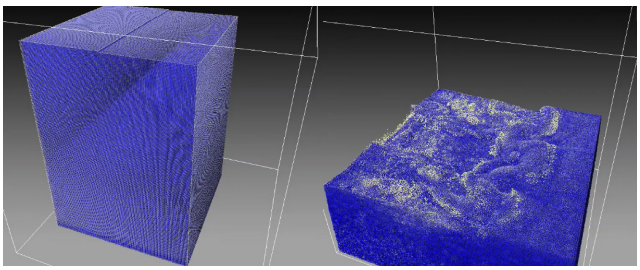

**Fig. 9** Dam break simulation with 4,194,304 particles.

10,000,000 particles are used, over 1.0 GB memories are needed. As the video memories of the graphics card used in this study is 768 MB, about 4,000,000 particles are the maximum number of particles that can be computed on the GPU.

There is room for improvement of the data structure of the bucket which is a uniform grid in this study. Most of the voxels at the upper part of the computation domain of simulations in Figures 6 and 7 are left unfilled. This uniform grid prevent us from applying to simulations in a larger computation domain. Generating sparse grid structure efficiently on the GPU is an open problem.

## 6 Conclusion

We presented a SPH implementation algorithm in which the entire computation is performed on the GPU. Approximately 60,000 of particles could be simulated in real-time and the proposed method also accelerated offline simulation. Then, the computation time was measured by varying the total number of particles and was compared with the computation time using the CPU. The comparison shows that the computational speed of the proposed method on the GPU is up to 28 times faster than that implemented on the CPU.

## References


1. Amada, T., Imura, M., Yasumoto, Y., Yamabe, Y., Chihara, K.: Particle-based fluid simulation on gpu. In: ACM Workshop on General-Purpose Computing on Graphics Processors (2004)
2. Bargteil, A.W., Goktekin, T.G., O'Brien, J.F., Strain, J.A.: A semi-lagrangian contouring method for fluid simulation. ACM Transactions on Graphics **25**(1), 19–38 (2006)
3. Carlson, M., Mucha, P., Turk, G.: Rigid fluid: Animating the interplay between rigid bodies and fluid. ACM Transactions on Graphics **23**(3), 377–384 (2004)
4. Clavet, S., Beaudoin, P., Poulin, P.: Particle-based viscoelastic fluid simulation. In: Symposium on Computer Animation 2005, pp. 219–228 (2005)
5. Courty, N., Musse, S.: Simulation of large crowds including gaseous phenomena. In: Proc. of IEEE Computer Graphics International, pp. 206–212 (2005)
6. Enright, D., Marschner, S., Fedkiw, R.: Animation and rendering of complex water surfaces. ACM Transactions on Graphics **21**, 721–728 (2002)
7. Foster, N., Fedkiw, R.: Realistic animation of liquids. In: Proc. of ACM SIGGRAPH, pp. 471–478 (2001)
8. Foster, N., Metaxas, D.: Controlling fluid animation. In: Proc. of the 1997 Conference on Computer Graphics International, pp. 178–188 (1997)
9. Goktekin, T., Bargteil, A., O'Brien, J.: A method for animating viscoelastic fluids. ACM Transactions on Graphics **23**, 464–467 (2004)
10. Guendelman, E., Selle, A., Losasso, F., Fedkiw, R.: Coupling water and smoke to thin deformable and rigid shells. ACM Transactions on Graphics **24**, 910–914 (2005)
11. Harris, M.: Fast fluid dynamics simulation on the gpu. GPU Gems pp. 637–665 (2004)
12. Harris, M., Baxter, W., Scheuermann, T., Lastra, A.: Simulation of cloud dynamics on graphics hardware. In: Proc. of the the SIGGRAPH/Eurographics Workshop on Graphics Hardware, pp. 92–101 (2003)
13. Harris, M., Coombe, G., Scheuermann, T., Lastra, A.: Physically-based visual simulation on graphics hardware. In: Proc. of the SIGGRAPH/Eurographics Workshop on Graphics Hardware, pp. 109–118 (2002)
14. Irving, G., Guendelman, E., Losasso, F., Fedkiw, R.: Efficient simulation of large bodies of water by coupling two and three dimensional techniques. ACM Transactions on Graphics **25**, 812–819 (2006)
15. Kipfer, P., Segal, M., Westermann, R.: Uberflow: A gpu-based particle engine. In: Proc. of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 115–122 (2004)
16. Kipfer, P., W.Rüdiger: Realistic and interactive simulation of rivers. In: Proc. of the 2006 conference on Graphics Interface, pp. 41–48 (2006)
17. Klinger, B., Feldman, B., Chentanez, N., O'Brien, J.: Fluid animation with dynamic meshes. ACM Transactions on Graphics **25**, 820–825 (2006)
18. Kolb, A., Cuntz, N.: Dynamic particle coupling for gpu-based fluid simulation. In: Proc. of 18th Symposium on Simulation Technique, pp. 722–727 (2005)
19. Kolb, A., Latta, L., Rezk-Salama, C.: hardware based simulation and collision detection for large particle systems. In: Proc. of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 123–131 (2004)

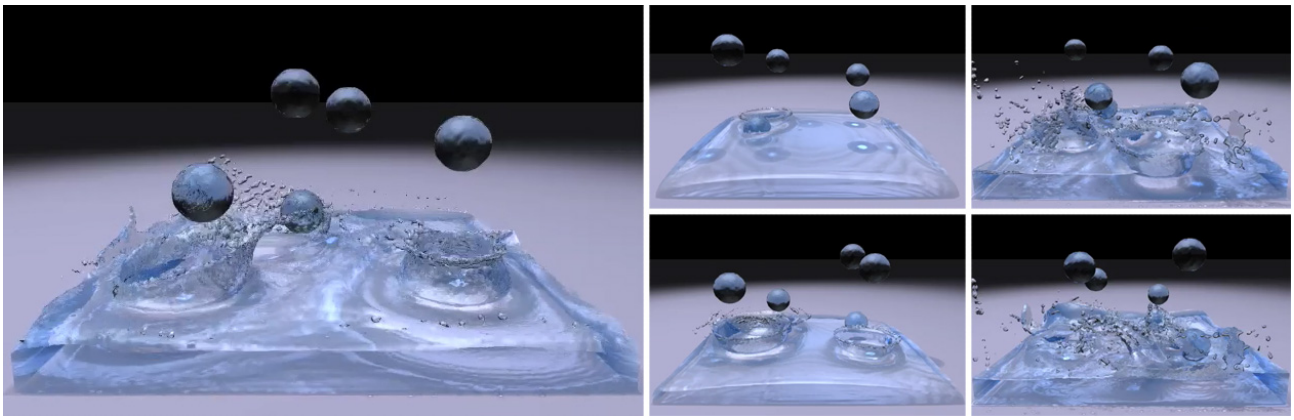

**Fig. 7** Balls of fluid is fallen into a tank.

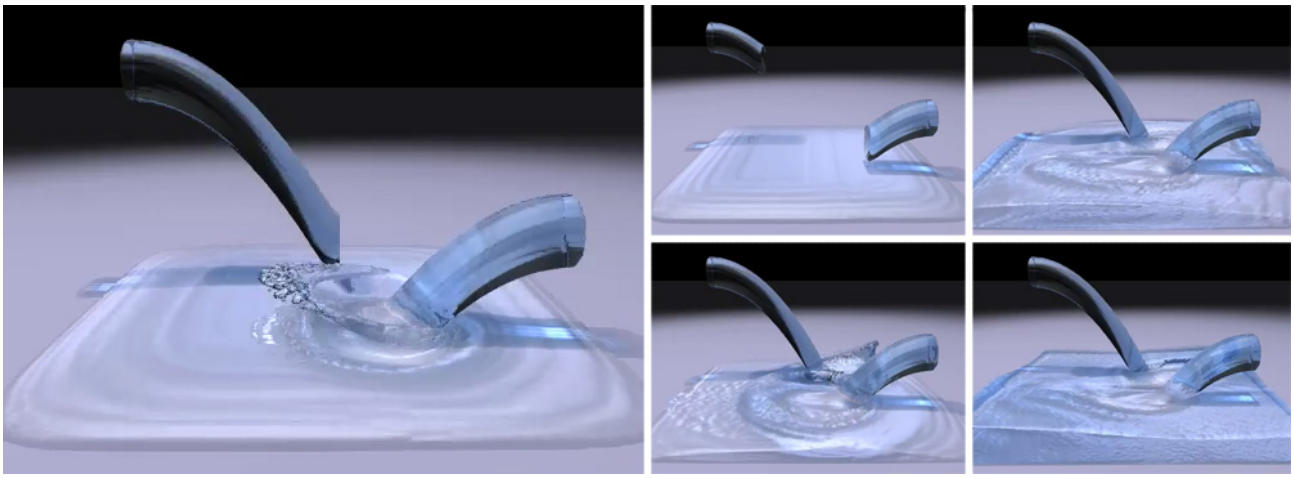

**Fig. 8** A fluid is poured into a tank.


20. Koshizuka, S., Oka, Y.: Moving-particle semi-implicit method for fragmentation of incompressible flow. Nucl. Sci. Eng. **123**, 421–434 (1996)
21. Krüger, J., Westermann, R.: Linear algebra operators for GPU implementation of numerical algorithms. ACM Transactions on Graphics **22**(3), 908–916 (2003)
22. Li, W., Fan, Z., Wei, X., Kaufman, A.: Gpu-based flow simulation with complex boundaries. Technical Report, 031105, Computer Science Department, SUNY at Stony Brook (2003)
23. Liu, Y., Liu, X., Wu, E.: Real-time 3d fluid simulation on gpu with complex obstacles. In: Proc. of the Computer Graphics and Applications, 12th Pacific Conference, pp. 247–256 (2004)
24. Lorensen, W., Cline, H.: Marching cubes: A high resolution 3d surface construction algorithm. In: Proc. of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 163–169 (1987)
25. Losasso, F., Gibou, F., Fedkiw, R.: Simulating water and smoke with and octree data structure. ACM Transactions on Graphics **23**, 457–462 (2004)
26. Mishira, B.: A review of computer simulation of tumbling mills by the discrete element method: Parti-contact mechanics. International Journal of Mineral Processing **71**(1), 73–93 (2003)
27. Monaghan, J.: Smoothed particle hydrodynamics. Annu.Rev.Astrophys. **30**, 543–574 (1992)
28. Müller, M., Charypar, D., Gross, M.: Particle-based fluid simulation for interactive applications. In: Proc. of Siggraph Symposium on Computer Animation, pp. 154–159 (2003)
29. Müller, M., Schirm, S., Teschner, M., Heidelberger, B., Gross, M.: Interaction of fluids with deformable solids. Journal of Computer Animation and Virtual Worlds **15**(3), 159–171 (2004)
30. Müller, M., Solenthaler, B., Keiser, R., Gross, M.: Particle-based fluid-fluid interaction. In: Proc. of SIGGRAPH Symposium on Computer Animation, pp. 237–244 (2005)
31. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Eurographics 2005, State of the Art Reports pp. 21–51 (2005)
32. Premoze, S., Tasdizen, T., Bigler, J., Lefohn, A., Whitaker, R.: Particle-based simulation of fluids. Computer Graphics Forum **22**(3), 401–410 (2003)
33. Purcell, T., Cammarano, M., Jensen, H., Hanrahan, P.: Photon mapping on programmable graphics hardware. In: Proc. of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 41–50 (2003)
34. Stam, J.: Stable fluids. In: Proc. of ACM SIGGRAPH, pp. 121–128 (1999)