

## SPH: Towards Flood Simulations

ROMAN ĎURIKOVIČ,<sup>†1</sup> MICHAL CHLÁDEK<sup>†2</sup>  
and TOMOYUKI NISHITA<sup>†1</sup>

Flood simulation is a complex problem involving large masses of fluid, soil watering, erosion and collision. We will demonstrate the flood simulation of cities where the fluid particles collide with a complex city model. For the fluid simulation we had used Smoothed Particle Hydrodynamics (SPH) method, by which we gained less expensive computation than by using other known methods of water simulation. Because of complex models, a particle nearest neighbor search and collision handling take an important role in the simulation and become a computational burden. We simulate a one-way solid fluid interaction (either solid influences the velocity of the fluid or fluid moves the solid) that requires having fine details in the colliding areas. We propose a new approach for nearest neighbor search and the fast approach of collision handling in particle based methods by using the distance from surface. We implemented SPH fluid simulator which can import a model represented by boundaries. We visualize the results, reconstruct the surface of the fluid and export it into a COLLADA file that can be rendered in Standard 3D rendering software.

### 1. Introduction

Physically based animations have been an active research area in the computer graphics. Computation fluid dynamics (CFD) makes a huge part of such animations. Everyday phenomena like rain, smoke, mud, steam, ocean waves or pouring water are important for computer graphics and one of the hardest to simulate. The mathematical model that handles the fluid motion is given by Navier-Stokes equations, one for conservation of momentum, and another for conservation of energy. The Navier-Stokes equations for incompressible fluids are

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \nabla(v \nabla u) - \frac{1}{\rho} \nabla p + \frac{f}{\rho}, \quad (1)$$

<sup>†1</sup> The University of Tokyo

<sup>†2</sup> Comenius University in Bratislava

$$\nabla \cdot u = 0, \quad (2)$$

where  $u = (u, v, w)$  is velocity vector,  $\rho$  is a fluid density,  $\nu$  is fluid viscosity,  $f = (f, g, h)$  is the external force, mostly it is only the gravitation force, i.e. ( $f = h = 0; g = 9.8ms^{-2}$ ),  $p$  is the pressure in the fluid and  $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$  is nabla operator. Equation 1 is the law of momentum and Eq. 2 represents the conservation of mass. Left side of Eq. 1 is the change of fluid velocity over time and the right side is the sum of the acting forces on fluids like advection, diffusion, pressure and external forces.

Two fundamental approaches for calculation of fluid movement exist. These are Eulerian and Lagrangian methods. They differ in fluid representation as well as in the time integration. Eulerian methods are based on Marker and Cell method<sup>1)</sup>, here the simulation space is divided into voxels, which form a grid. In each voxel of the grid, properties of the fluid are stored. In each time step advection of the fluid is computed. The use of Navier-Stokes equations in computer graphics was greatly popularized Foster<sup>2)</sup>, Stam<sup>3)</sup> and Fedkiw<sup>4)</sup>.

In Lagrangian methods, particles are used to represent a fluid. Particles are not connected and store physical values like mass, position, and velocity. Other properties can be used to further describe the fluid (e.g. viscosity, temperature). In the simplest simulation there are no forces acting between particles. For more complex simulation of fluid dynamics like pouring water forces acting between particles are needed. Various approaches how to model these forces exist. Further on, we will describe only solutions used to simulate the water flow.

Simulation of water is specific because of water incompressibility. Simulation methods used to simulate water should be able to manage this. There are used two solutions. One of them is moving-particle semi-implicit<sup>5)</sup>(MPS). A Poisson equation of pressure is solved here to achieve fully incompressible fluid flow.

Beside MPS, smoothed particle hydrodynamics (SPH) is widely used to simulate water. Originally developed for simulation of astrophysical phenomena<sup>6)</sup> it was introduced into computer graphics community for depicting fire and gaseous phenomena<sup>7)</sup>. Later it was used in simulation of highly deformable bodies<sup>8)</sup>, lava flows<sup>9)</sup>, real-time simulation of water<sup>10)</sup>, and it was adopted to simulate multiple fluid interaction<sup>11)12)13)</sup>. In contrast with

MPS, SPH can not handle fully incompressible fluids. Therefore, approaches that reduce the compressibility to a reasonable amount were proposed based on Tait equation<sup>14)</sup> or based on iterative error correction method.

**Our contribution.** In this paper, we aim at proposing a solution for flood simulation of cities by using the SPH method. The drawback of SPH is that often the boundaries had to be represented by particles attached to obstacle objects. This gives a unified approach to both the collision handling and the fluid simulation. However, with the increasing size and the complexity of a model, the number of particles used to represent boundaries increases rapidly and becomes a problem. Therefore, we propose alternative ways how to solve the collisions and particle interactions.

The paper is organized as follows: In the next section we describe SPH method that we used in particle simulation with the proposal for searching the nearest neighbor particles in Section 3. In Section 4, we show collision handling based on intersections as well as our proposed method. In Section 5, we briefly describe our visualization method. In the end, we present our results.

## 2. SPH

Smoothed particle hydrodynamics is a Lagrangian fluid simulation method originally developed for simulating astrophysical phenomena. Later, it was adopted in other fields like highly deformable solids<sup>8)</sup> or computational fluid dynamics<sup>10)</sup>.

SPH is an Lagrangian i.e. interpolation method. It approximates values and derivatives of a continuous fluid by interpolated discrete samples represented by particles. Each particle carries properties like mass, position, or velocity. These values can be computed at any point  $\mathbf{r}$  in simulation space by interpolating the properties of neighboring particles. This is done by so called kernel functions<sup>10)</sup> or smoothing kernel  $A$ :

$$A(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} W(\mathbf{r} - \mathbf{r}_b, h), \quad (3)$$

where we sum over all particles,  $m_b$  is mass of particle with index  $b$ ,  $\rho_b$  is its density, and  $A_b$  is known property  $A$  of particle  $b$  at position  $\mathbf{r}_b$ .  $W$  is a kernel function with support radius  $h$ . The kernel function must be even i.e.  $W(\mathbf{r}, h) = W(-\mathbf{r}, h)$  and normalized

$\int W(\mathbf{r} - \mathbf{r}', h) d\mathbf{r}' = 1$ . For example, to compute density  $\rho$  of particle  $i$  at position  $\mathbf{r}_i$ , we substitute  $\rho(\mathbf{r}_i)$  for  $A(\mathbf{r})$  in Eq. 3:

$$\rho(\mathbf{r}_i)\rho_i = \sum_b m_b \frac{\rho_b}{\rho_b} W(\mathbf{r}_i - \mathbf{r}_b, h) = \sum_b m_b W(\mathbf{r}_i - \mathbf{r}_b, h). \quad (4)$$

We can see that it corresponds with the physical definition of density because it represents mass around a particle. Unlike in Eulerian approaches with particles in a grid, the derivatives in SPH can be computed at an arbitrary point in the fluid. The advantage of Eq. 3 is that the derivation affects only the kernel functions. The gradient of  $A$  is

$$\nabla A(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla W(\mathbf{r} - \mathbf{r}_b, h) \quad (5)$$

and the Laplacian of  $A$  is

$$\nabla^2 A(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla^2 W(\mathbf{r} - \mathbf{r}_b, h). \quad (6)$$

### 2.1 SPH and Navier-Stokes Equations

Because we are working with isothermal fluids, the Navier-Stokes equations can be used in a simplified form. Fluid is expressed by a velocity field  $\mathbf{v}$ , a density field  $\rho$  and a pressure field  $p$ . Thus by expressing the evolution of the fluid in the time, we get one equation for conservation of mass

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0, \quad (7)$$

and another one for conservation of momentum

$$\rho \left( \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v}. \quad (8)$$

This equation can be further simplified. Because we use a particle method, where the particle mass and the particles number is constant, conservation of mass is guaranteed. Thus Eq. 9 can be omitted. The expression  $\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}$  in Eq. 10 is the convective derivative, which is a derivative taken with respect to Lagrangian coordinate system. Lagrangian coordinate system is a coordinate system, which moves together with the fluid flow. Because particles move with the flow we can use time derivative of the velocity instead of convective derivative as follows

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = \frac{D\mathbf{v}}{Dt}, \quad (9)$$

where  $t$  states for time. After the simplifications of Navier-Stokes equations and notating the particle acceleration as  $\mathbf{a}_i$  we get:

$$\mathbf{a}_i \equiv \frac{\partial \mathbf{v}_i}{\partial t} = \frac{-\nabla p_i + \mu \nabla^2 \mathbf{v}_i + \mathbf{f}_i}{\rho_i}. \quad (10)$$

Next, we calculate forces resulting from pressure ( $-\nabla p_i$ ) and viscosity ( $\mu \nabla^2 \mathbf{v}_i$ ) by substituting  $p(\mathbf{r}_i)$  for  $A(\mathbf{r})$  in Eq. 3

$$f_i^{pressure} \equiv -\nabla p(\mathbf{r}_i) = -\sum_b m_b \frac{p_b}{\rho_b} \nabla W(\mathbf{r}_i - \mathbf{r}_b, h). \quad (11)$$

and analogically, by substituting  $\mu \nabla^2 \mathbf{v}(\mathbf{r}_i)$  for  $A(\mathbf{r})$  in Eq. 3

$$f_i^{viscosity} \equiv \mu \nabla^2 \mathbf{v}(\mathbf{r}_i) = \mu \sum_b m_b \frac{v_b}{\rho_b} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_b, h). \quad (12)$$

Instead of ideal fluid law ( $p = k\rho$ ) we calculate the pressure by

$$p = k(\rho - \rho_0), \quad (13)$$

where  $k$  is the fluid stiffness and  $\rho_0$  is the rest density. Thus, the fluid tends to hold density near the rest density.

The problem with Eqs. 13 and 14 is that the forces resulting from them, are not necessarily symmetric. When we compute this forces for two particles which don't have the same pressure or velocity, the forces differ, and the third Newton's law is violated. Many approaches exist to make this equations symmetric. We use the approach from<sup>10)</sup>:

$$f_i^{pressure} = -\sum_b m_b \frac{p_b + p_i}{2\rho_b} \nabla W(\mathbf{r}_i - \mathbf{r}_b, h) \quad (14)$$

and

$$f_i^{viscosity} = \mu \sum_b m_b \frac{v_b - v_i}{\rho_b} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_b, h). \quad (15)$$

In addition we need gravitation and surface tension forces. Surface tension<sup>10)</sup> can be derived from the estimated fluid surface normal  $\mathbf{n}$  as follows:

$$c_s(\mathbf{r}) = \sum_j m_j \frac{1}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h), \quad \mathbf{n} = \nabla c_s, \quad (16)$$

$$\mathbf{f}^{surface} = -\sigma \nabla^2 c_s \frac{\mathbf{n}}{|\mathbf{n}|}, \quad (17)$$

where  $\sigma$  is the surface tension coefficient. Leap-frog numerical integration is used for advancing position and velocity every time step.

### 3. Neighborhood Query

The influence of each particle is only local, within a support  $h$ , it is essential to find the set of neighbor particles. Since this highly affects further calculations of fluid dynamics it becomes usually the bottleneck of the overall animation. In the standard uniform grid approach, we construct 3D grid of cells with size  $h$ . Each particle  $n$  with position  $x = (x; y; z)$  is inserted into one spatial cell with coordinates  $(k; l; m)$ . In order to determine the neighborhood of  $n$ , only particles that are in the same spatial cell or in one of the neighboring spatial cells within distance  $h$  need to be queried. For each particle  $n$ , all particles in the 27 neighboring cells are tested for interaction. Due to the sorting, particles that are in the same spatial cell are also close in memory. This improves the memory coherence (cache-hit rate) of the query. However, it depends on the indexing scheme if particles in neighboring cells are also close in memory.

Here we extend the usual *Spatial Hashing* technique for faster SPH simulation and propose a novel approach *Cell Indexing* based on indexing non-empty cells in a virtual subdivision grid, see Onderik<sup>15)</sup> for more details. Keys are usually 64-bit numbers, 16 bits for each  $(n, i, j, k)$  component. For each particle we define it's key  $(n, i, j, k)$  as

$$\text{key}(n, i, j, k) = n + 2^I i + 2^{I+J} j + 2^{I+J+K} k, \quad (18)$$

where  $2^I > N$ ,  $2^J > \lfloor B_x/h \rfloor$ ,  $2^K > \lfloor B_y/h \rfloor$ , and  $n$  is index of particle,  $N$  is the number of particles,  $B_x$ ,  $B_y$  and  $B_z$  are dimensions of bounding box,  $(i, j, k) = \text{cell}(x, y, z)$  are cell coordinates and  $I, J, K$  and are arbitrary constants, usually 16 or 32 depending on the size of the cell size  $h$ . Notice, that function key  $(n, i, j, k)$  encodes its parameters to

a unique key. Thus, given a key  $q$  we can compute unique  $(n, i, j, k)$ . Particle index is  $n = q \bmod 2^I$ , and cell coordinates are  $i = (q/2^I) \bmod 2^J, \dots$

To make the overall neighbor search algorithm linear, we must sort keys in linear time. Since we have encoded indices of particles into their keys, we can use radix sort and have correct indices after sorting.

#### 4. Collision Handling

One of widely used methods for collision handling is sticking the surface with particles or representing the solid objects with particles. We have decided to not handle the collisions in this way because the number of particles increases rapidly with the size of the model. By simulating a flood of a city, this would be a great computational burden. We use two ways of collision handling. One method is based on intersections and the other one on distances.

##### 4.1 Intersection Method

In general, collision handling can be divided into two steps namely collision detection and collision response. In collision detection step, we consider that a collision occurred in time between  $t$  and  $t + \Delta t$ . Define a segment by position of collided particle  $i$  at time  $t$  and  $t + \Delta t$ . There has to be an intersection of this segment with the mesh. Thus, the collision detection is just finding an intersection of a segment with a mesh. In response step, we move the particle back to the intersection point and reflect the velocity along the surface normal. The velocity of the particle is updated to

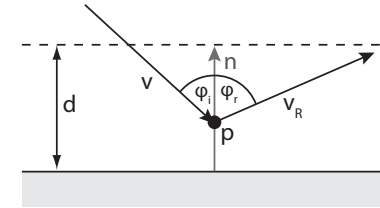
$$\mathbf{v}_i = \mathbf{v}_i - \left(1 + c_R \frac{d}{\Delta t |\mathbf{v}_i|}\right) (\mathbf{v}_i \cdot \mathbf{n}) \mathbf{n}, \quad (19)$$

where  $0 \leq c_R \leq 1$  is a constant controlling preservation of kinetic energy,  $v_i$  is the velocity, and  $d$  is the penetration depth. Value  $c_R = 0.45$  seems to work well. The main drawback of this method is a relatively slow calculation compared to the approach based on distances.

##### 4.2 Distance Method

If the distance of a particle from the surface is smaller than  $d$  then we presume that a

collision with the surface can occur. We used  $d = h$ , where  $h$  is support radius of the kernel functions<sup>16)</sup>, in our simulation. The collision will occur only when the particle  $i$  is moving toward the surface. So we have to test if  $\mathbf{v}_i \cdot \mathbf{n}(\mathbf{r}_i) < 0$ . If this test is positive we handle the collision by reflecting the velocity of the particle along the surface normal with greater angle of reflection than the angle of incidence, see Fig. 1. The response is applied before the collision actually happens, exactly at the time when the distance is smaller than  $d$ .



**Fig. 1** Collision handling in the distance method. Velocity  $v$  of particle  $p$  is reflected along the surface normal, where  $\phi_i \neq \phi_r$ , because it is moving towards surface.

Model used in the simulation does not change in time and is represented by a mesh, thus we can use this fact to accelerate the computation of the distance function. The distance of a particle from the surface corresponds to the distance of the particle to the closest face of the mesh. We can pre-compute a regular grid of pointers to nearest faces and at places, where the distance is above a certain threshold the pointer is NULL. When computing the distance of a particle from surface we find the closest pointer in the grid. If the pointer is NULL, we do not have to handle the collision. If pointer is not NULL, we compute the distance of the particle to the pointed face.

If gravitation or pressure force is pushing a particle towards the surface, in some cases, reflecting of the velocity is not enough to get the particle farther from the surface than  $d$ . Such particle will gradually come closer to the surface and eventually go through it. To avoid this problem, we use a force that pushes particles away from the surface, if they come closer to the surface than distance  $d$ . We apply the force by increasing velocity in the direction of surface normal

$$\mathbf{v} = \mathbf{v} + c(2d - \text{dist})\mathbf{n}, \quad (20)$$

where  $\mathbf{v}$  is the velocity of a particle,  $c$  is a user defined constant and  $dist$  is the distance of a particle from the surface.

## 5. Visualisation

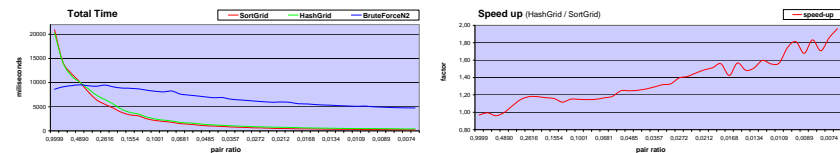
Fluid surface reconstruction is done using marching cube algorithm, where density is used as isofunction and obtained mesh is then fitted to the model. The marching cube algorithm works in such a way that it produces a small gap between the fluid surface and the obstacle mesh, next we wish to correct this gap. When a vertex of the fluid mesh is closer to the obstacle surface than user specified distance  $d_s$ , we fit the mesh vertex to the surface. It can be done as

$$\bar{\mathbf{x}} = \mathbf{x} - \mathcal{D}(\mathbf{x})\mathbf{n}(\mathbf{x}), \quad (21)$$

where  $\bar{\mathbf{x}}$  is a fitted vertex,  $\mathbf{x}$  is the vertex before fitting,  $\mathcal{D}(\mathbf{x})$  is the distance of the vertex  $\mathbf{x}$  from the surface, and  $\mathbf{n}(\mathbf{x})$  is the normal in the closest point of the model surface from the vertex  $\mathbf{x}$ . We use  $d_s = h$ , where  $h$  is the support radius of the the kernel functions used in SPH. The acceleration method of distance finding from the collision handling is used here to accelerate the surface fitting. The results are than exported into a COLLADA file that can be imported into 3ds Max and rendered.

## 6. Results

To measure the efficiency of our algorithm we have performed following testing scenario. Given 15000 randomly sampled particles we get  $15000 \times 15000 = 225000000$  all possible pairs of particles. Depending on the sampling domain size, only a fraction of them belong to close particle pairs. We define the *pair ratio* as the fraction between *close pairs* and *all pairs*. Setting  $h = 1$  we started with a sampling domain  $2 \times 2 \times 2$  (pair ratio = 0.9999) and end with domain  $15 \times 15 \times 15$  (pair ratio = 0.0069). Graph on left in Figure 2 shows measured total time [ms] on axis Y needed to report all close particle pairs using three different algorithms namely *SortGrid* (Cell Sorting), *HashGrid* (Spatial Hashing) and *BruteForceN2* (All-Pair-Test) for different test case with warring pair ratio shown on axis X. Figure 2 on right demonstrates the speed up ratio between Cell Sorting and Spatial Hashing for decreasing pair ratio, for low pair ratio Cell Sorting is almost



**Fig. 2** Algorithm efficiency. Left) Relation between pair ratio and the respective total time required to find close particle pairs using Cell Sorting (red), Spatial Hashing (green) and All-Pair-Test (blue). Axis Y notes the time in *ms* and axis X the decreasing pair ratio. Right) The speed up ratio between Cell Sorting and Spatial Hashing for decreasing pair ratio. Axis Y notes the speedup ration and axis X pair ratio.

twice as fast as Spatial Hashing. We implemented described our flood simulator on a PC with Intel P9400 2.4 GHz processor with 4.0GB RAM, but we did not fully exploited the multicore potential. We tested the presented methods with two scenarios summarised in Table 1. Collision handling based on distances produces similar visual results as the method based on intersections, but with a higher speed. The tested scene animation of a fluid in box is shown in Fig. 3 and the city urban model with flowing water is shown in Fig. 4. OpenSceneGraph was used for importing of the 3D models used in collision

**Table 1** Comparison of frames per second calculated using the collision handling based on distances and the collision handling based on intersections.

scene	number of particles	distance	intersections	time of the precomputation
cuboid	5510	17-20 fps	5-6.5 fps	5s
city	8892	10-13 fps	2-3.5 fps	2m 18s

handling the best results were achieved using 3DS and OBJ file formats. GNU triangulated surfaces library was used in surface reconstruction and COLLADA DOM was used for exporting the water surface and geometry of the model into a COLLADA file.

## 7. Conclusions

In this paper, we presented a method of water simulation, which can be used in large simulations such as flood simulations. We use the SPH particle method with proposed nearest neighbor search algorithm using the sorted indexes with the hashing table. We implemented collision handling based on the distance from the surface.

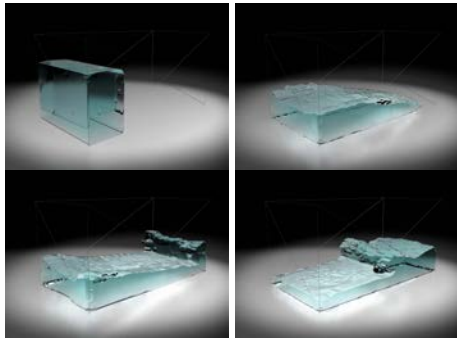


Fig. 3 Fluid surface rendered in 3ds Max.

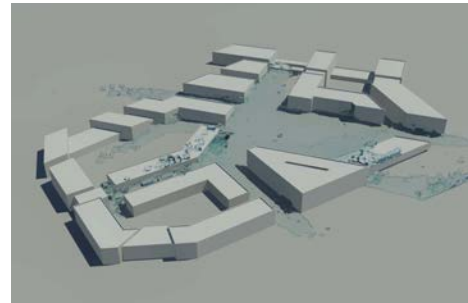


Fig. 4 Flood simulation of a city rendered in 3ds Max.

**Acknowledgments** This research was partially supported by a VEGA 1/0662/09 2009-2011 project a Scientific grant from Ministry of Education of Slovak Republic and Slovak Academy of Science; and The University of Tokyo research grant.

### References

- 1) Harlow, F.H. and Welch, E.J.: Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface, *Physics of Fluids*, Vol.8, No.12, pp.2182–2189 (1965).
- 2) Foster, N. and Metaxas, D.: Modeling the motion of a hot, turbulent gas, *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., pp.181–188 (1997).
- 3) Stam, J.: Stable fluids, *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., pp.121–128 (1999).
- 4) Fedkiw, R., Stam, J. and Jensen, H.W.: Visual simulation of smoke, *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, ACM, pp.15–22 (2001).
- 5) Koshizuka, S. and Oka, Y.: Moving-particle semi-implicit method for fragmentation of incompressible fluids, *Nucl. Sci. Eng.*, Vol.126, pp.421–434 (1996).
- 6) Monaghan, J.: Smoothed Particle Hydrodynamics, *Annual review of astronomy and astrophysics*, Vol.30, pp.543–574 (1992).
- 7) Stam, J. and Fiume, E.: Depicting fire and other gaseous phenomena using diffusion processes, *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, New York, NY, USA, ACM, pp.129–136 (1995).
- 8) Desbrun, M. and Gascuel, M.-P.: Smoothed particles: a new paradigm for animating highly deformable bodies, *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, New York, NY, USA, Springer-Verlag New York, Inc., pp.61–76 (1996).
- 9) Stora, D., Agliati, P.-O., Cani, M.-P., Neyret, F. and Gascuel, J.-D.: Animating lava flows, *Proceedings of the 1999 conference on Graphics interface '99*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., pp.203–210 (1999).
- 10) Müller, M., Charypar, D. and Gross, M.: Particle-based fluid simulation for interactive applications, *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 154–159 (2003).
- 11) Müller, M., Solenthaler, B., Keiser, R. and Gross, M.: Particle-based fluid-fluid interaction, *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, New York, NY, USA, ACM, pp.237–244 (2005).
- 12) Solenthaler, B. and Pajarola, R.: Density contrast SPH interfaces, *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp.211–218 (2008).
- 13) Lenaerts, T., Adams, B. and Dutré, P.: Porous flow in particle-based fluid simulations, *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, New York, NY, USA, ACM, pp.1–8 (2008).
- 14) Becker, M. and Teschner, M.: Weakly compressible SPH for free surface flows, *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp.209–217 (2007).
- 15) Onderik, J. and Durikovic, R.: Efficient neighbor search for particle-based fluids., *Journal of the Applied Mathematics, Statistics and Informatics*, Vol.4, pp.29–43 (2008).
- 16) Harada, T., Koshizuka, S. and Kawaguchi, Y.: Smoothed Particle Hydrodynamics in Complex Shapes, *Proceedings of the 23st spring conference on Computer graphics* (2007).