FLUID SIMULATION BY PARTICLE LEVEL SET METHOD WITH AN EFFICIENT DYNAMIC ARRAY IMPLEMENTATION ON GPU

Yasuhiro Matsuda
The University of Tokyo

Yoshinori Dobashi
Hokkaido University

Tomoyuki Nishita
The University of Tokyo

ABSTRACT

We propose an efficient method to treat dynamic array data on Graphics Processing Unit (GPU), which is applicable to fluid simulations. Few numbers of dynamic structures have been realized on GPU since most of the previous methods store the data in texture to represent the structure and it is difficult to manage the dynamic structure on the texture. Our method uses vertex buffer object for representing the data structure and combines the transform feedback and the geometry program, which are the functionalities of GPU. Our method offers a simple but an efficient way of realizing dynamic array on GPU. Furthermore we apply our method to the implementation of a particle level set method. The particles data are represented by our dynamic array and they are updated, added, and deleted completely on GPU. By using the method, a fast and accurate fluid simulation can be realized.

1. INTRODUCTION

Fluid such as water, smoke, and fire are fairly common phenomena in daily life. This is the natural reason why creating realistic images of fluid is one of the most important research topics in the field of computer graphics. Synthetic animations of fluids are used in many applications such as commercial movies, games and virtual reality. Therefore it is very important to develop a method that can create realistic animations of fluids.

A lot of methods have been presented to create animations of fluid. Many methods employ physically-based approaches for producing a realistic fluid animation. Most of the previous methods calculate the motion of fluids by solving the Navier-Stokes equations, which are the dominant equation of fluid dynamics. The grid-based approach that we employed in this paper subdivides the simulation space into grids and calculate the physical quantities at each grid point. Fluid motion is represented by these physical quantities at each point.

The grid-based methods need a surface tracking method to extract the fluid surfaces, which are interfaces between different kind of fluids (e.g. water and air). The level set method [15] has been widely used as the surface tracking method for not only a fluid simulation but also many areas such as computer graphics, image processing and computational geometry. The level set method represents the interfaces as the zero value points of the implicit function called a level set function, and tracks the interfaces by updating the function based on the equation called advection equation. But naive implementations of the level set method for a fluid simulation suffer from a volume loss caused by numerical dissipation. To alleviate the problem, the method is extended to the particle level set method [6], which is known as one of the most accurate surface tracking methods. The particle level set method additionally places fictitious particles around the interfaces, and updates the particles independently of the level set function. The method can track the interface very accurately by correcting the level set function with the information of the particles. The accuracy is however achieved at the expense of high computational costs. To address this problem, we show a fast method that accelerates the particle level set method by exploiting the parallel computational power of GPU. The particle level set method needs addition and deletion of particles in the simulation of fluid to accurately track the surfaces. Previous GPU-based simulation methods store data as a form of textures. However, textures are not suitable for storing the particle's information since the number of particles dynamically changes throughout the simulation. We alternatively use vertex buffer object to store the particle data. By combining transform feedback and geometry program, that are the functions of GPU, we realize dynamic array on GPU. In other words, update, addition, and deletion of element are carried out fully on GPU. Our method offers a simple and efficient way of particle addition and deletion in the particle level set method. Finally we show a complete GPU implementation of the fluid simulation with the free surfaces by using our method.

Contributions of this paper are listed below.

- Fast and accurate fluid simulation with free surfaces completely on GPU

- Data transferring technique from the particle structure to the grid structure using the depth buffer
- Particle addition and deletion algorithm on GPU

The rest of the paper is organized as follows. Section 2 discusses the related works. In Section 3, we give an overview of the particle level set method. Section 4 describes the details of our algorithm that is suitable for GPU acceleration. Section 5 demonstrates the efficiency of our method by using several examples. Finally conclusions and future work are explained in Section 6.

2. RELATED WORK

Most of the methods for simulations of complex fluid motions are based on the methods developed in computational fluid dynamics. Stam proposed an unconditionally stable fluid model [19], which used the semi-Lagrangian method with an implicit solver. This model allowed a significantly large time step and realized very fast simulations without losing stability. Because of this advantage, the model is widely used in computer graphics.

To track free surfaces, Osher et al. proposed the level set method [15] using an implicit signed distance function called level set function. To improve the accuracy, Foster et al. proposed a method for tracking dynamic liquid surface, which was based on combining the level set method with fictitious particles [7]. Enright et al. extended the method and proposed the particle level set method [4, 5, 6], which improved the accuracy of tracking water surface. Moreover, Enright et al. [5] demonstrated that the particle level set method yielded sufficient accuracy even if first order semi-Lagrangian scheme was used to advect the level set function. Losasso et al. extended the method to enable the method to treat multiple liquids [13].

On the other hand, several particle-based methods have been proposed as alternatives to the above grid-based methods. Stam and Fiume [20] introduced smoothed particle hydrodynamics (SPH) to depict fire and gaseous phenomena. In SPH, the fluid is modeled as a collection of particles with a smoothed potential field. Adams et al. reduced the computational costs of SPH by introducing an adaptive sampling algorithm [2]. Premôze et al. [16] introduced the use of the moving particle semi-implicit method (MPS) for simulating incompressible multiphase fluids. One drawback of particle-based methods is that, if insufficient particles are used, they tend to track free surfaces inaccurately. To prevent this, a sufficiently large number of particles must be used, which increases the computational cost. However, particle-based methods have a merit that it can easily treat the topology change of free surfaces.

Many methods have been implemented on GPU for exploiting computational power of GPU. Harris et al. presented a cloud simulation and rendering method [8] that is completely implemented on GPU. Wu et al. presented a three dimensional fluid simulation method [21] on GPU but the simulation was limited to fixed boundary conditions. Rumpf et al. implemented the level set method [17] on GPU. This method is extended to a sparse method [12] by Lefone et al. Particle-based fluid simulation is also implemented on GPU. Amada et al. accelerated a fluid simulation by SPH using GPU [3]. But processes for searching neighboring particles is done in CPU so the transferring data between CPU and GPU can be a bottleneck. Kolb presented another method for SPH on GPU [11], which eliminates the necessity of the neighbor search by summing up the kernel function of particles on texture by using alpha blending of GPU. Independently from us, Nicolas et al. recently proposed a GPU implementation of the particle level set method [14]. But they did not treat the fluid simulation and the addition and deletion of particles on GPU.

3. OVERVIEW OF PREVIOUS METHODS

This section describes an overview of the particle level set method which our method is based on.

3.1. Fluid Dynamics

The motion of fluids is calculated by solving the following Navier-Stokes equations.

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \nu \nabla^2 \mathbf{u} + \mathbf{g}, \tag{2}$$

where $\mathbf{u}$ is the velocity of the fluid, p is the pressure, ν is the kinematic viscosity, ρ is the density and $\mathbf{g}$ is a gravitational force.
To calculate the equations, we used the stable fluids method [19]. This method updates the velocity field by calculating Eq. (2). This method obtains the updated velocity by calculating the terms on the right side of Eq. (2) one by one. The advection term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ is calculated by a semi-Lagrangian method. The divergence free condition is satisfied by solving the Poisson equation derived from Eqs. (1), (2).

3.2. Level Set Method

The particle level set method is an extended version of the level set method. The level set method represents fluid surfaces with an implicit function ϕ called level set function. This function satisfies the following conditions,

$$\phi(\mathbf{x}) > 0 \quad \text{for} \quad \mathbf{x} \notin \Omega,$$
$$\phi(\mathbf{x}) \leq 0 \quad \text{for} \quad \mathbf{x} \in \Omega,$$

where $\mathbf{x}$ is the coordinate of the simulation space and Ω represents the simulation space containing fluids. In other

words, $\phi < 0$ cooresponds the fluid region, $\phi > 0$ to the air region and $\phi = 0$ indicates the interfaces. The initial value is determined in accordance with the initial settings of fluid. A singed distance from the interfaces is often used for the level set function and it is used in this paper, too.

The level set function is updated with the advection equation given by

$$\phi_t + \mathbf{u} \cdot \nabla\phi = 0. \tag{3}$$

To maintain the signed distance property of the level set function, we solve the reinitialization equation given by,

$$\phi_\tau \quad = \quad -S(\phi_{\tau=0})(|\nabla\phi| - 1), \tag{4}$$

where τ is a fictitious time. $S(\phi)$ is a smoothed signed distance function defined by

$$S(\phi) \quad = \quad \frac{\phi}{\sqrt{\phi^2 + h^2}}, \tag{5}$$

where h is a grid spacing.

3.3. Particle Update

The particle level set method uses fictitious particles to improve the accuracy of the level set method described in the previous subsection. The particles are placed near the surfaces. Each particle has a sign (± 1) which is the same as the sign of the value of the level set function where the particle is initially placed. The position of the particles are updated using,

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}(\mathbf{x}_p), \tag{6}$$

where $\mathbf{x}_p$ is the position of each particle and $\mathbf{u}(\mathbf{x}_p)$ is its velocity.

Each particle has a radius, which is used to identify the error and to correct the level set function. The radius is determined based on the distance from the fluid surfaces. But it is bounded by a minimum r_{min} and a maximum r_{max}. We used $r_{min} = 0.1h$ and $r_{max} = 0.5h$. The radius is updated at every time step by the following equation,

$$r_p = \begin{cases} r_{max} & s_p\phi(\mathbf{x}_p) > r_{max} \\ s_p\phi(\mathbf{x}_p) & r_{min} \le s_p\phi(\mathbf{x}_p) \le r_{max} \\ r_{min} & s_p\phi(\mathbf{x}_p) \le r_{min} \end{cases}. \tag{7}$$

3.4. Error Correction by Particles

For each particle p, a spherical level set function ϕ_p is associated. Its value at $\mathbf{x}$ is determined by the particle radius as follows,

$$\phi_p(\mathbf{x}) = s_p(r_p - |\mathbf{x} - \mathbf{x}_p|),, \tag{8}$$

where s_p is the sign of the particle. The zero level set of ϕ_p corresponds to the boundary of the particle sphere. These level sets are only defined locally at the eight points (four points in two-dimensional case) of the grids neighbouring the particle. The values of ϕ_p are the estimated values of the level set function at the points. They are used to correct the level set function. To identify grids which needs to be corrected, the particle radius and the sign are used. As the simulation proceeds, some particles moves into a region where the sign of the level set value is different from the sign of the particles. Among them, the particles whose distances from the interfaces are longer than their radius are called *escaped particles*. The positive escaped particles are used to correct the level set function of $\phi > 0$ region and the negative escaped particles are used to correct the level set function of $\phi \le 0$ region. Let us consider the $\phi > 0$ region. Using Eq. 8, the ϕ_p values at the eight grid points neighbouring the particle are calculated. Each ϕ_p is compared to the local value of ϕ and the maximum of these two values is taken as ϕ^+. This is done for all escaped positive particles. That is, given a level set ϕ and a set of escaped positive particles E^+, ϕ^+ is initialized with ϕ and calculated as follows,

$$\phi^+ = \max_{\forall p \in E^+}(\phi_p, \phi^+), \tag{9}$$

where E^+ is a set of positive escaped particles. ϕ_p is calculated and compared with ϕ at only eight neighbor points of particles. Similarly, the corrected level set function of the $\phi \le 0$ region denoted by ϕ^- is initialized with ϕ and calculated as follows,

$$\phi^- = \min_{\forall p \in E^-}(\phi_p, \phi^-), \tag{10}$$

where E^- is a set of negative escaped particles. ϕ^+ and ϕ^- are finally merged to a single level set by

$$\phi = \begin{cases} \phi^+ & \text{if } |\phi^+| \le |\phi^-| \\ \phi^- & \text{if } |\phi^+| > |\phi^-| \end{cases}. \tag{11}$$

3.5. Addition and Deletion of Particles

As simulation advances, the interface stretches and tears. In such regions there may not be enough number of particles to obtain sufficient accuracy. In order to avoid this inaccuracy caused by sparse particles and maintain a sufficient density throughout the simulation, particles are reseeded periodically. Particles are not only added in grid cells near the interface, but also deleted when they have drifted too far from the interface to provide useful information, e.g. for a threshold b_{max}, positive particles with $\phi(\mathbf{x}_p) > b_{max}$ and negative particles with $\phi(\mathbf{x}_p) < -b_{max}$. This prevents particles from increasing infinitely.

3.6. Summary of Simulation Process

We use the stable fluids method to calculate the velocity field of the fluid and the particle level set method to track the surfaces. The velocity field, the level set function and the particles are initialized to the initial state. Then we update them by the above methods. The whole simulation is summilized as below. In the list, 2 to 7 correspond to the processes of one time step.

1. Initialization of data(level set funciton, velocity, particle).
2. Update of the velocity
3. Update of the level set function
4. Update of the paticles
5. Error correction of the level set function by paritlces
6. Reinitialization of the level set function
7. Addition and Deletion of particles (per several time steps)
8. Back to 2

4. PROPOSED METHOD

4.1. Data Structure

For GPU implementation, we used OpenGL graphics API and Cg (version 2.0 beta) shading language. To store the data of level set function ϕ, we used a 3D texture with a 32 bit floating point luminance color. For fluid velocity $\mathbf{u}$, we used a 3D texture with 32 bit floating point RGB colors, where each color is respectively corresponds to the x, y, or z component of velocity. For particles data, we used vertex buffer object, which offers a function to efficiently draw a large amount of primitives. To efficiently treat particles data, we packed particles coordinates and radii and signs into a four floating point values as follows. The first three values represent the particle coordinates. An absolute value of the fourth value represents the particle radius and its sign represents the particle sign. By this packing method, we represent a particle as an element of vertex buffer object with four components. Current GPU has a limitation that it cannot read from and write into a single texture or a single vertex buffer object at the same render pass. For this reason, we used a double buffering technique. That is, we prepared two textures for the level set function and the velocity, and two vertex buffer objects for the particles.

4.2. Velocity Computation

The fluid velocity is computed based on the stable fluids method [19]. The GPU implementation of the method is basically the same as Harris's one [9].

In this paper, air region is treated as empty. Therefore, we need a velocity extrapolation method to the air region for advancing particles at the region. We followed the model equation [6] for extrapolating the velocity given by

$$\frac{\partial u^a}{\partial \tau} = -\mathbf{N} \cdot \nabla u^a \ \ (a = 0, 1, 2), \tag{12}$$

where $\mathbf{u} = (u^0, u^1, u^2)$ is velocity vector, τ is a fictitious time and $\mathbf{N}$ is a unit vector perpendicular to the fluid surfaces. A fast method [6] can be used in case of CPU implementation. But this method is not suited for GPU implementation because it needs to proceed calculation in order of level set function value increasing by sorting. The method determines the velocity value by enforcing the condition $\nabla\phi \cdot \nabla\mathbf{u} = 0$ from the point where ϕ is the smallest.We used a similar method which was modified to suit for the GPU implementation. Our method enforces the condition in upwind manner without sorting. The computation order is show by numbers in Figure 1.

The concrete calculation processes are as follows. s_0 is x direction of point which has smaller level set function value and calculated as follows.

if $(\phi_{[i,j,k]} > \phi_{[i+1,j,k]}) \wedge (\phi_{[i,j,k]} > \phi_{[i-1,j,k]})$ then
  if $\phi_{[i-1,j,k]} > \phi_{[i+1,j,k]}$ then $s_0 = 1$
  else $s_0 = -1$
else if $\phi_{[i,j,k]} > \phi_{[i+1,j,k]}$ then $s_0 = 1$
else if $\phi_{[i,j,k]} > \phi_{[i-1,j,k]}$ then $s_0 = -1$
else $s_0 = 0$

s_1, s_2 are y and z directions respectively and calculated in a similar way with s_0. Updating velocity in the air region is computed by Eq. (13). The more details and derivation of the equation are found in [1]. By repeating this upwind update for several times, we obtain extrapolated values.

$$\begin{aligned}
\mathbf{I} &= (1,1,1), \\
\mathbf{\Phi}_{[i,j,k]} &= (\phi_{[i,j,k]}, \phi_{[i,j,k]}, \phi_{[i,j,k]}), \\
\mathbf{\Phi}'_{[i,j,k]} &= (\phi_{[i-s_0,j,k]}, \phi_{[i,j-s_1,k]}, \phi_{[i,j,k-s_2]}), \\
\mathbf{u}'^a_{[i,j,k]} &= (u^a_{[i-s_0,j,k]}, u^a_{[i,j-s_1,k]}, u^a_{[i,j,k-s_2]}), \\
u''^a_{[i,j,k]} &= \frac{\mathbf{u}'^a_{[i,j,k]} \cdot (\mathbf{\Phi}_{[i,j,k]} - \mathbf{\Phi}'_{[i,j,k]})}{\mathbf{I} \cdot (\mathbf{\Phi}_{[i,j,k]} - \mathbf{\Phi}'_{[i,j,k]})}.
\end{aligned} \tag{13}$$

4.3. Update of Level Set Function

By using GPU, the level set function is updated in pallalel based on Eq. (3). To compute this step, we used semi-Lagrangian advection method [5] in the fragment program.

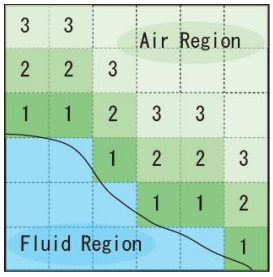


Figure 1. The calculation of velocity extrapolation proceeds from 1 to 3 .

To smooth the level set function, we conduct a process called reinitialization based on Eq. (4). To efficiently reinitialize the level set function, the fast marching method [18] is often used in CPU implementation. However the method is not suited for GPU implementation. Therefore we used a semi-Lagrangian style method used in [10] that is well-suited for GPU. The method divides Eq. (4) into an advection term $S(\phi_{\tau=0})|\nabla\phi|$ and a non-advection term $S(\phi_{\tau=0})$ and calculate each term one-by-one. Firstly, we calculate the advection by semi-Lagrangian method to produce an intermediate value. To this intermediate value, we add a value calculated by non-advection term. By repeting this process for several times, we obtain a reinitialized value.

4.4. Error Correction Algorithm with Depth Buffer

At this part, we identify and compute the *correction value* based on Eqs. (8) through (11) in order to fix the level set function. To compute above processes on GPU, we have to write the correction value into the texture of the level set function after reading the particle data from the vertex buffer object. Nicolas et al. proposed an implementation using alpha blending for this process [14]. We show another approach based on depth test.

Firstly we associate the correction value written to level set function texture with the depth value by defining transform function f. f is a monotonically increasing function and $f : \mathbf{L} \rightarrow [0, 1]$, where $\mathbf{L}$ is the range of the level set function. In practice, the range is usually bounded by some minimum value l_{min} and maximum value l_{max} so this function is simply defined for $l \in \mathbf{L}$ as follows,

$$f(l) = \frac{l - l_{min}}{l_{max} - l_{min}}. \tag{14}$$

By using this function, we prepare additional two textures for ϕ^+ and ϕ^-. We used flat 3D textures [8] as the data structure for these textures in order to allocate depth buffer for all level set function values. The initial values of these textures are set to the same values as the level set function ϕ obtained in the previous time step. To each texture we attach depth buffer by using framebuffer object, which is an OpenGL extension. The depth values of the buffers are set by transforming ϕ value using f. When we create the texture for ϕ^+, we set the depth test function to pass greater values. Similarly, when we create the texture for ϕ^- we set the depth test function to pass less values. Next we draw the data from vertex buffer object as points and calculate the destination position on texture in the vertex program. In the geometry program, we identify escaped particles. Around the position of escaped particles, we generate eight (four in two-dimentional case) points corresponding to neighboring grid points. Ordinary particles other than escaped particles are simply discarded by generating no point primitives. In the fragment program, we calculate a correction value and the depth value by transforming the correction value using transform function f. Finally, by merging ϕ^+ and ϕ^-, we obtain the level set function ϕ corrected by particles. Overall process is summarized as Figure 2

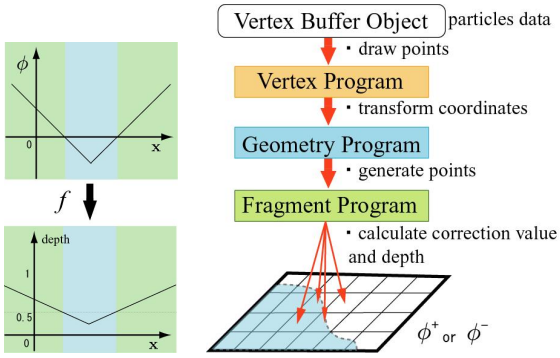


Figure 2. (Left) Mapping from level set function to depth value. (Right) Data flow in creating texture for ϕ^+, ϕ^-

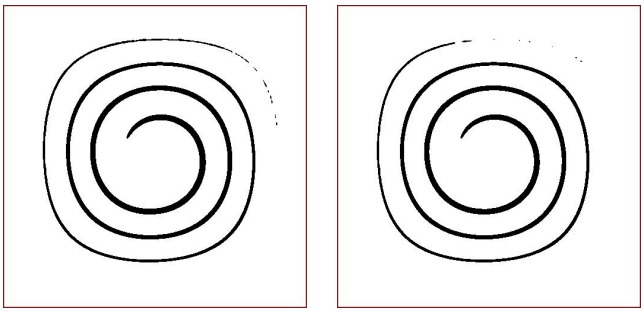
4.5. Particle Data Management on GPU

The position of the particles are updated using Eq. (6) by second order Runge-Kutta method in the vertex program. Among two vertex buffer objects for double buffering, one is used as input and the other as output. The results are written directly into the output vertex buffer object by using an OpenGL extension GL_NV_trnasform_feedback. It enables us to write the results calculated by the vertex program or the geometry program to the vertex buffer object without a rastarization. This function allows us to store the particle-based data as the vertex buffer object. The particle radius can be updated at the same vertex program pass because we packed the particle coordinate and radius into an element of vertex buffer object.

4.6. Addition and Deletion Algorithm on GPU

Addition and deletion of particles are needed to keep an accurate simulation. Figure 3 shows the behavior of a sur-

face advected and stretched in the vortex flow field [4]. Particles are added periodically in the left figure and not added in the right. As the shape is streched, particles get sparse at the outer end of the shape in the right figure, which causes area losses. On the other hand, the particle density is kept sufficient by addition in the left figure so there are fewer area losses in the shape.

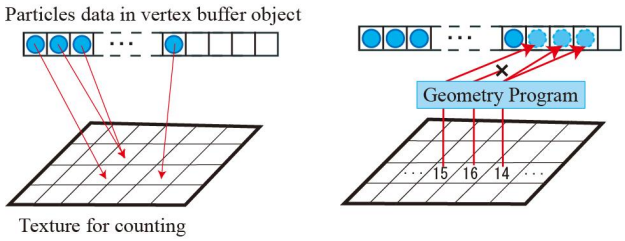


(a) Addition (b) No addition

Figure 3. Comparison of accuracy between particle addition and no addition in vortex flow.

In CPU implementations, we can easily do the addition and deletion of elements by using, for example, a list structure. However a similar structures efficiently working on GPU are not presented so far. We realize the addition and deletion in the following way. Firstly, before particle addition, we delete unnecessary particles. Unnecessary particles are the ones far from the surface or positive particles at the grid cell where the level set function has local minima, which is explained as a sink for air [13]. These particles are identified and deleted by not generating the point primitive in the geometry program.

Next we illustrate the addition algorithm. Although there are a variety of addition strategies according to situations, we used a simple reseeding strategy described in the following. To determine regions where particles are sparse, we simply count the number of particles at each grid cell and consider the grid cells as sparse when particles are fewer than the user specified number M. We add particles to the sparse grid cells at random position inside the grid cells until the number gets to M. To count particles at each grid cell, we utilize alpha blending and set the blending equation to GL_FUNC_ADD. In the same manner as correction of level set function by particle, we render points of size one into the texture for counting from vertex buffer object. The output colors of the points are set to 1.0 for calculating accumulated values of the number of particles. The number of added particles at each grid cell is computed in the geometry program. The number is zero if the grid point is too far from the surfaces, which is easily identified by checking the level set function value. Near the surfaces, the number is $M - N_{\mathbf{x}}$, where $N_{\mathbf{x}}$ is the number of particles in grid cells. We set the output offset in vertex buffer object to next to the last element of current particles. Finally we draw points of size one at each grid point and as described above, calculate $M - N_{\mathbf{x}}$ and generate the number of point primitives in the geometry program. Figure 4 illustrates this process (M=16 in the figure).



(a) Particle counting. (b) Particle generation.

Figure 4. (a)Firstly the particles are counted in each grid cell. (b)Then needed particles are calculated and generated in the geometry program.

5. RESULTS

To evaluate the accuracy of our method we conducted a deformation test under the incompressible flow field, which is also called Enright test [4]. This test advect the sphere of radius 0.15 placed at (0.35, 0.35, 0.35) in the velocity field given by

$$\mathbf{u}(x,y,z) = \begin{pmatrix} 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z) \\ -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z) \\ -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z) \end{pmatrix}.$$

Figure 5 is the images of this test. (a) is by our method, (b) by the CPU implementation, (c) by our method without correction of the level set function by particles. (a) and (b) revert to the almost the same shape as the original, which means they accurately track the surface. By comparing (a) with (b), we can see our method can track the surface with almost the same accuracy as the CPU implementation. On the other hand, a large volume is lost in the final shape during the test of (c) due to the numerical dissipations. The necessity of the correction of the level set function by particles is shown by this result.

Table 1 shows the time to calculate a single time step of the simulation of this test and compares the result between the GPU implementation (our method) and the CPU implementation. Table 2 lists the time for each process (96^3 grid cells). As shown in Table 1, we achieved about 10 to 13 times faster simulation than that of the CPU implementation.

Figures 6 and 7 show the results of three dimensional fluid simulations by our method. In Figure 6, a water droplet falls onto the water surface and makes waves. In

Figure 7, armadillo-shaped water collapses and splashes. The number of grids in each simulation is 128^3. All three dimensional graphics images are rendered by using Pov-Ray. In our experiments, we used a machine with Pentium Core2 Quad 2.66GHz, and GeForce 8800 Ultra as the graphics card.

Table 1. Comparison of the computational times (msec).

Grids	CPU	GPU	Ratio
64^3	364	36.7	9.92
96^3	896	70.8	12.7
128^3	1715	132	13.1

Table 2. Computational times of each process (msec).

process	CPU	GPU
update of level set function	37.5	3.12
reinitialization of level set function	104	12.5
update of particles	562	6.23
error correction of level set function	109	25.5
particle addition (per 20 time steps)	1667	188

6. CONCLUSION AND FUTURE WORK

We have proposed a novel method for dynamic array on GPU to treat particle data which dynamically increase and decrease. By using our method, a fast particle level set method on GPU is realized. Our method achieved about 10 to 13 times faster simulations than the particle level set method on CPU. We demonstrated that a fast and accurate fluid simulation with free surfaces was realized by our method.

In future work, we plan to develop a method for treating interaction between solid and fluid completely on GPU.

7. REFERENCES

[1] D. Adalsteinsson and J. A. Sethian. The fast construction of extension velocities in level set methods. *Journal of Computational Physics*, 148(1):2–22, 1999.

[2] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. Adaptively sampled particle fluids. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 48, New York, NY, USA, 2007. ACM Press.

[3] T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, and K. Chihara. Particle-Based Fluid Simulation on GPU. *ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH*, 2004.

[4] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[5] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Computers and Structures*, 83:479–490, 2005.

[6] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744, New York, NY, USA, 2002. ACM Press.

[7] N. Foster and R. Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[8] M. Harris, W. Baxter, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 92–101, 2003.

[9] M. J. Harris. Fast fluid dynamics simulation on the gpu. In *GPU Gems*, pages 637–665. Addison Wesley Pub., 2004.

[10] B. Kim, Y. Liu, I. Llamas, and J. Rossignac. Advections with significantly reduced dissipation and diffusion. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):135–144, 2007.

[11] A. Kolb. Dynamic particle coupling for GPU-based fluid simulation. *18th Symposium on Simulation Technique.*, 2005.

[12] A. Lefohn, J. Kniss, C. Hansen, and R. Whitaker. A streaming narrow-band algorithm: interactive computation and visualization of level sets. *International Conference on Computer Graphics and Interactive Techniques*, 2005.

[13] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw. Multiple interacting liquids. *International Conference on Computer Graphics and Interactive Techniques*, pages 812–819, 2006.

[14] C. Nicolas, S. Robert, and K. Andreas. Real-Time Particle Level Sets with Application to Flow Visualization. Technical report, Siegen University, 2007.

[15] S. Osher and J. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[16] S. Premože, T. Tasdizen, J. Bigler, A. Lefohn, and R. Whitaker. Particle-based simulation of fluids. *Computer Graphics Forum*, 22(3):401–410, 2003.

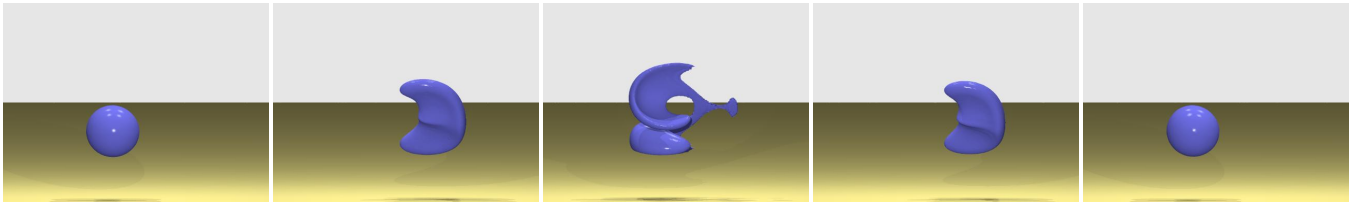
[17] M. Rumpf and R. Strzodka. Level set segmentation in graphics hardware. *Image Processing, 2001. Proceedings. 2001 International Conference on*, 3, 2001.

[18] J. A. Sethian. Evolution, implementation, and application of level set and fast marching methods for advancing fronts. *Journal of Computational Physics*, 169(2):503–555, 2001.

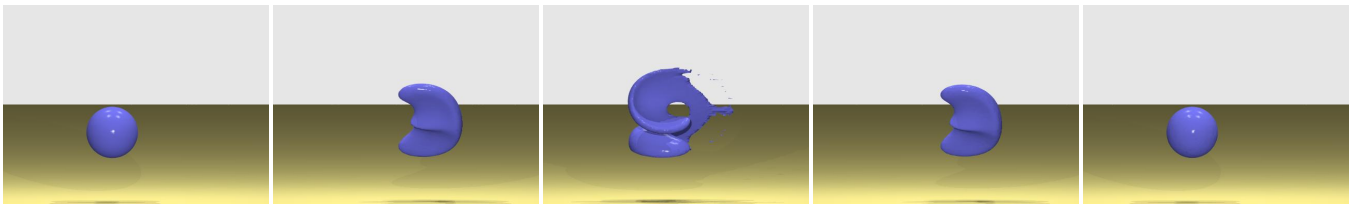
[19] J. Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[20] J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 129–136, New York, NY, USA, 1995. ACM Press.

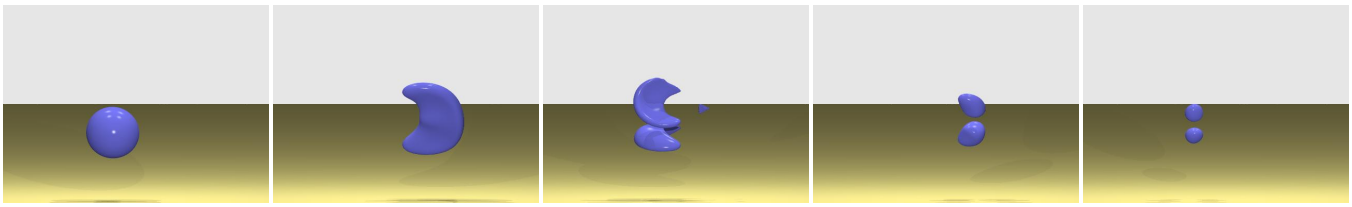
[21] E. Wu, Y. Liu, and X. Liu. An improved study of real-time fluid simulation on GPU. *Computer Animation and Virtual Worlds*, 15(34):139–146, 2004.



(a) GPU implementation



(b) CPU implementation



(c)Rotation test without the correction of the levelset function by particles.

Figure 5. Comparison of accuracy between CPU and GPU implementation by Enright test.

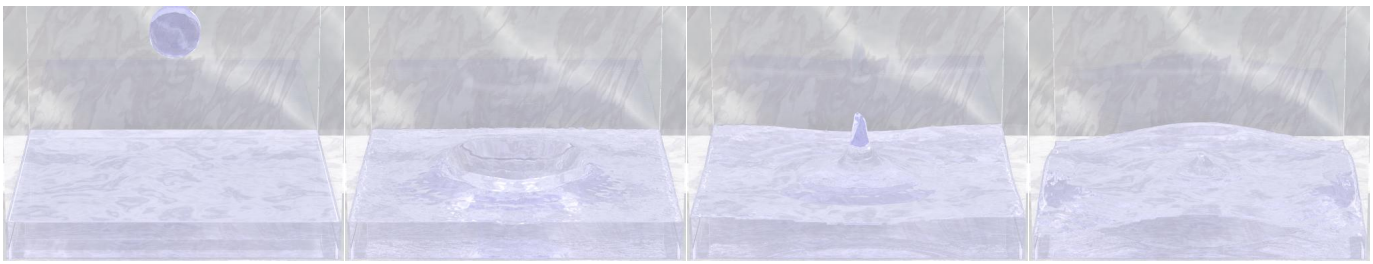


Figure 6. Water droplet falls on the surface.

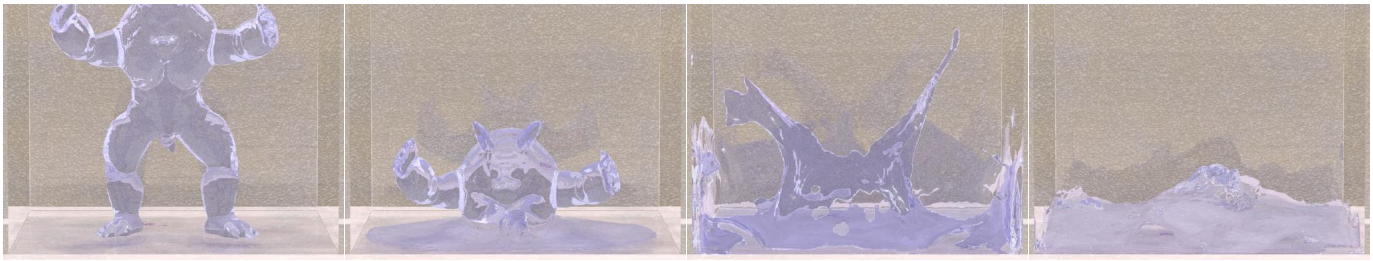


Figure 7. The fluid simulation of armadillo model collapsing.