# Particle-Based Fluid Simulation on GPU

*Takashi Amada, Masataka Imura, Yoshihiro Yasumuro, Yoshitsugu Manabe and Kunihiro Chihara*

**Nara Institute of Science and Technology**
**8916-5, Takayama, Ikoma, Nara, 630-0192, Japan.**
**takasi-a@is.naist.jp**

## Introduction

Rendering realistic moving water is one of the key techniques that immerse the viewers into interactive graphics world including computer games. Physical simulations based on computational fluid dynamics (CFD) is useful for rendering the realistic behaviour of water. However, real-time fluid rendering has been one of the challenging tasks because of high computational cost of CFD. According to the recent development of programmable graphics hardware, many graphics functions are replaced by hardware processors. In this research we propose real-time particle-based fluid simulation with Smoothed Particle Hydrodynamics (SPH) on Graphics Processing Unit (GPU).

## Smoothed Particle Hydrodynamics

SPH is one of the CFD methods which represent fluid as a collection of particles instead of grid-based field. Field quantities are expressed as summation of physical values each of which is weighted by smoothing kernel product in the vicinity of each particle. A physical quantity $A_i$, its gradients and Laplacian are computed by **Eq.1**, **Eq.2** and **Eq.3** where $m_j$ is the mass of particle $j$, $\rho_j$ the density, $\mathbf{r}_i$ and $\mathbf{r}_j$ is the location of particle $i$ and $j$ respectively. $W$ is the smoothing kernel which has decreasing profile along distance between particles. To solve these equations the particle density distribution around each particle must be known beforehand.

$$A_i = \sum_j m_j \frac{A_j}{\rho_j} W(\mathrm{r}_i - \mathrm{r}_j) \qquad (1)$$

$$\nabla A_i = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\mathrm{r}_i - \mathrm{r}_j) \qquad (2)$$

$$\nabla^2 A_i = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\mathrm{r}_i - \mathrm{r}_j) \qquad (3)$$

## Process Outline

Proposed method has two key features: a neighbour map creation on CPU and procedural computation of kernel products with the most use of GPU. In our design, bare-bones process for dynamic searching with condition evaluations is assigned to CPU, whose output, neighbour texture map structure allows GPU to carry out the further computation to treat variety of attributes data on GPU. First, all parameters are initialized and a neighbour map is constructed in CPU by searching and listing up neighbouring particles within a fixed range for each particle. The neighbour map is then transferred to GPU as a texture, which is referred to and used for determining the physical attributes of the particles. Attributes computations, including collision handling, are rapidly processed as texture modifications by fragment program on GPU. Finally, time integration is performed to update the state. **Fig.1** shows the process flow.

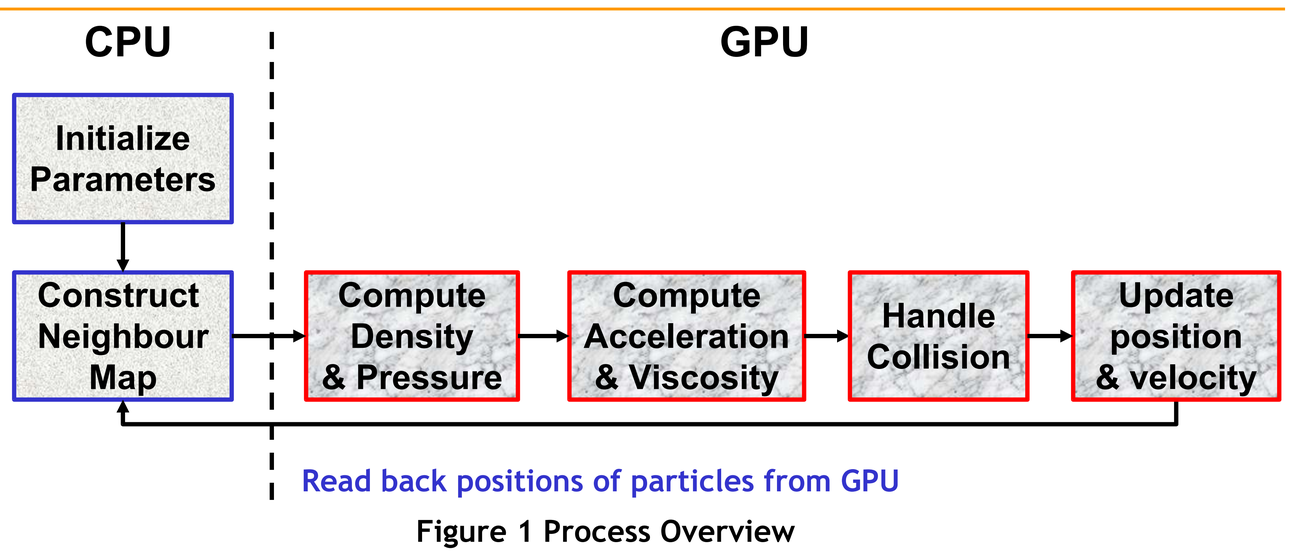



Figure 1 Process Overview

## Neighbour Map Texture

Before computing on GPU, a neighbor map in a form of 2D texture is constructed on CPU to store indices of the particles along *s*-axis in the vicinity of a particle indexed by *t* coordinate[**Fig.2**]. Neighbour map shows the lists of particles that may physically effect on every single particle.

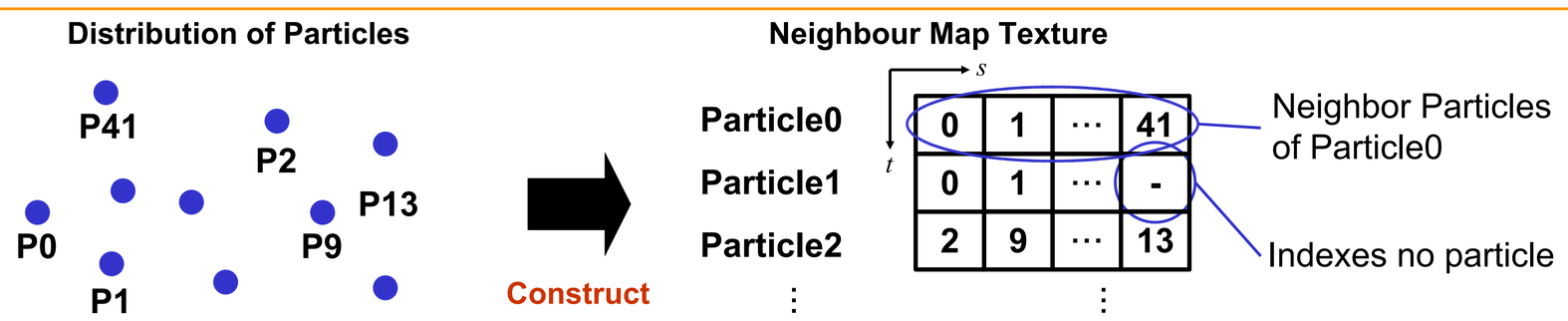



Figure 2 Neighbour Map Construction

## Computing Attributes

Equations of SPH(**Eq.1**, **Eq.2**, **Eq.3**) have similar forms of weighted summation to give physical quantities and can be computed by gathering the product of neighbour particles by referring neighbour map and drawing a line along *s*-axis in the product texture, whose length is equivalent of the number of *t*-th particles' neighbours.[**Fig.3**] Summing up the products for each particle determines all types of the attributes, including density, pressure and force due to pressure and viscosity for each particle consequently.

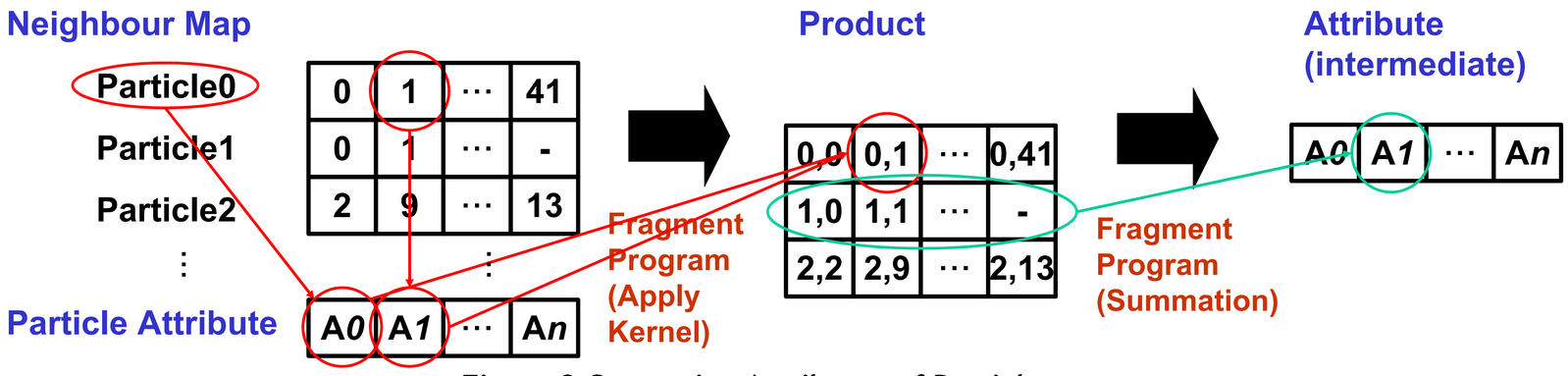



Figure 3 Computing Attributes of Particles

## Collision Handling

Process for detecting collisions and deriving their responses uses the attributes texture and boundary texture which contains triangle mesh surface information, then acceleration attribute is generated. Finally, time integration updates the position attributes based on Leap-Frog scheme which can be also executed by simple texture modification.

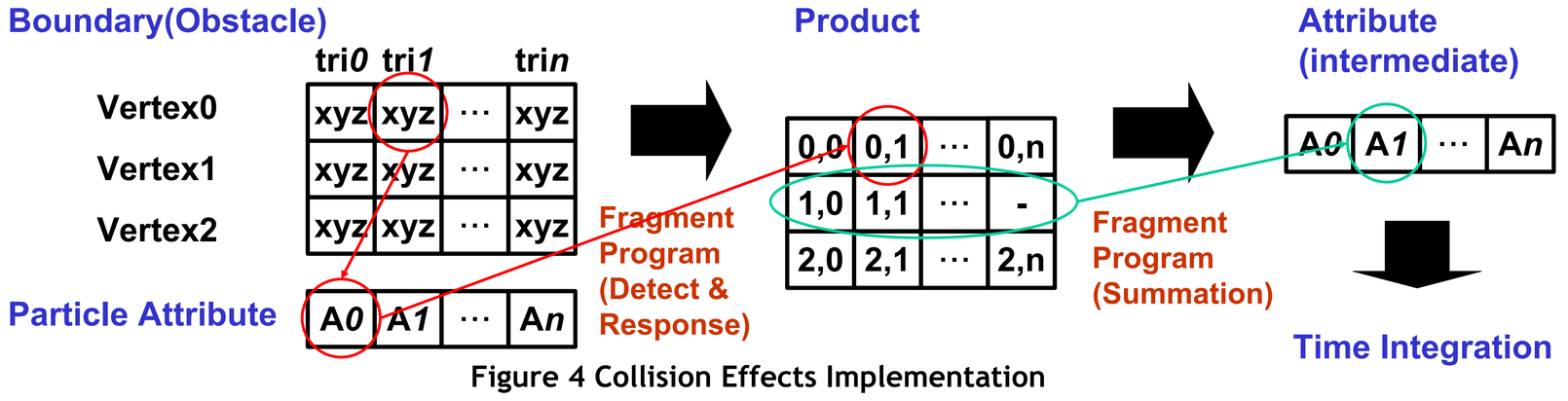
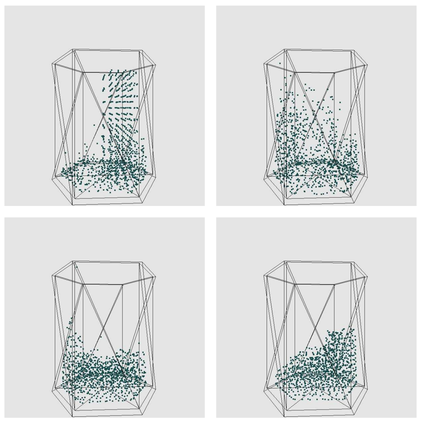



Figure 4 Collision Effects Implementation

## Experimental Results

We conducted simulations with 1000, 1500 and 2000 particles by the proposed GPU-assisted method and only on CPU. [Table1]

**Experiment Environment**
- **CPU: Intel Pentium4 2.8 GHz**
- **GPU: NVidia GeForce FX 5950 Ultra**
- **NVidia Cg Compiler 1.2.1**



| Num. of Particles | GPU: Construct Neighbour Map and Transfer | GPU: Solve Equations | GPU: Overall | CPU: Construct Neighbour Map | CPU: Solve Equations | CPU: Overall |
|---|---|---|---|---|---|---|
| 1000 | 4.4ms | 3.6ms | 8.0ms | 3.4ms | 11.1ms | 14.5ms |
| 1500 | 5.7ms | 4.0ms | 9.7ms | 4.3ms | 12.3ms | 16.6ms |
| 2000 | 6.9ms | 4.6ms | 11.5ms | 5.2ms | 14.5ms | 19.7ms |

Table 1 Results(Simulation Time)

## Discussions

The proposed GPU-assisted method is capable of solving the SPH equations about 3 times faster and running 2 times faster in overall process. The bottlenecks of the process are neighbour map constriction and data transferring between CPU and GPU. Simulation time can be even shortened by skipping neighbour map reconstruction in some cases in which neighbouring particle groups differ slightly, depending to the scene situations.