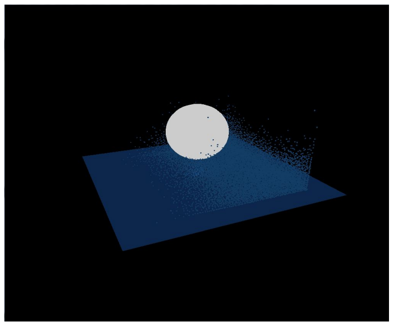
*Journal of Computer Graphics Techniques* Vol. 2, No. 1, 2013 http://jcgt.org

# CIS 565 Final Project : Position Based Fluids

Harmony Li
University of Pennsylvania

Joseph Tong
University of Pennsylvania



## Abstract

In light of the recent publics, we have implemented a GPU based position based fluid simulation stable enough to support large steps for real-time applications. Through the enforcement of the constant density positional constraints, the simulation allows for incompressibility and convergence similar to smooth particle hydro-dynamics solvers. Out project will be largely based on Muller and Macklin's recent paper "Position Based Fluids". Optimizing the simulator using a spatial hash grid, we were able to write a stable simulator that runs in real-time for 10,000+ particles.

## 1. Introduction

As the graphics industry continues to grow, physically based water simulation continues to be an important issue in the community. Previously, most of these solutions sought to simulate water through smooth particle hydro-dynamics (SPH). However, this method has known to be cumbersome, having intricate parameters that have no physical meaning at times. Similarly, SPH usually requires around 50 neighbors around any given particle at a time in order to ensure stability. With such a high member requirement, it makes it hard to move SPH as an offline method of simulation to an online or real-time simulator.

Beyond simulation, water simulations also have high upfront costs for rendering. Since SPH is a particle based simulation, mesh building is required to build a mesh that can be easily rendered in programs like Maya. Mesh reconstruction is typically accomplished through a method called level sets. While these methods have been used often, it is still complicated and difficult to accomplish.

## 2. Related Work

To address these issues, researchers have developed hybird methods like Fluid Implicit-Particle (FLIP) that use a grid to solve the pressue and transfer velocity changes back to the particle itself. Similarly, others have pursued predictive-corrective incompressible SPH (PCISPH), which attempts to make the time step less, uses a Jacobi-style. In 2012, some researchers have tried to acheive uniform density fluids by modeling incompressibility as a system of velocity constraints. Most recently, Muller and Macklin have applied the concept of position based dynamics to SPH, opting to use a parallel Jacobi iterator to achieve desired effects on the GPU. Most of our implementation is based on this paper.

## 3. Position Based Fluids

Position Based Dynamics provides a method of simulating dynamics based on Verlet integration. Because of its ability to support stable dynamic simualtions at large timesteps, it has been proposed for use in games and the like. While Muller et al used a Gauss-Seidel approach in their first paper in 2007, Muller and Macklin have chosen to use a parallel Jacobi iteration solver to take advantage of the GPU.

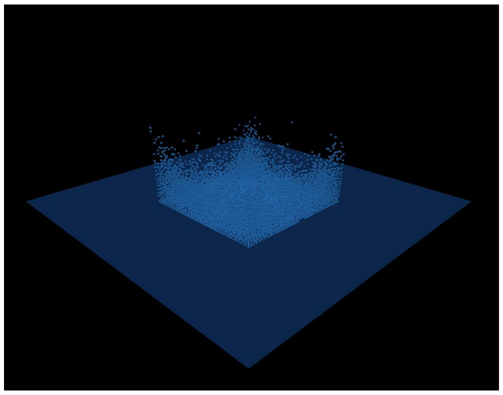


**Figure 1**. Preliminary results from alpha.

### 3.1. Theory

In most liquids, incompressibilty and constant density are crucial for a physically accurate recreation of a fluid simulation. The key idea behind Muller and Macklin's paper and our simulation is to enforce this incompressibiltiy with the use of position constraints. The following is the density constraint for every particle in the system.

$$C_{\mathrm{i}}(\mathbf{p}_1,...,\mathbf{p}_{\mathrm{n}}) = \tfrac{\rho_{\mathrm{i}}}{\rho_0} - 1$$

The density is calculated using the standard SPH density estimator:

$$\rho_{\mathrm{i}} = \sum_j m_{\mathrm{j}} W(\mathbf{p}_{\mathrm{i}} - \mathbf{p}_{\mathrm{j}}, h).$$

For density estimation, Poly6 kernel is used. For gradient calculation, the Spiky kernel is used.

$$W_{\mathrm{poly6}}(r,h) = \tfrac{315}{64\pi h^9}(h^2 - |r|^2)^3$$

$$\nabla W_{\mathrm{spiky}}(r,h) = \tfrac{45}{\pi h^6}(h - |r|)^2 \tfrac{r}{|r|}$$

The resulting position update $\Delta\mathbf{p}_{\mathrm{i}}$, after including the corrections from neighborhood particles density constraints $(\lambda_j)$ is as follows:

$$\Delta\mathbf{p}_{\mathrm{i}} = \tfrac{1}{\rho_0}\sum_j (\rho_i + \rho j)\nabla W(\mathbf{p}_{\mathrm{i}} - \mathbf{p}_{\mathrm{j}}, h)$$

$$\text{where } \lambda_i = -\tfrac{C_{\mathrm{i}}(\mathbf{p}_1,...,\mathbf{p}_{\mathrm{n}})}{\sum_k |\Delta_{\mathbf{p_k}} C_i|^2 + \varepsilon}$$

$\varepsilon$ is user defined relaxation parameter. The inherent repulsive nature of $\lambda_i$ keeps density slightly lower than the rest density which causes particles to exhibit very realistic surface tenstion-like effects.
Like all postion based simulations, accumulation of damping occurs which is physically unrealistic. Thus, the introduction of vorticity confinement is necessary to introduce more energy into the simulation. The vorticity at each particle's location is calculated as follows:

$$\omega_i = \nabla \times \mathbf{v} = \sum_j \mathbf{v_{ij}} \times \nabla_{p_j} W(\mathbf{p}_{\mathrm{i}} - \mathbf{p}_{\mathrm{j}}, h) \text{ where } \mathbf{v_{ij}} = \mathbf{v_j} - \mathbf{v_i}.$$

The corrective force added into the system is

$$f_i^{vorticity} = \varepsilon(\mathbf{N} \times \omega_i).$$

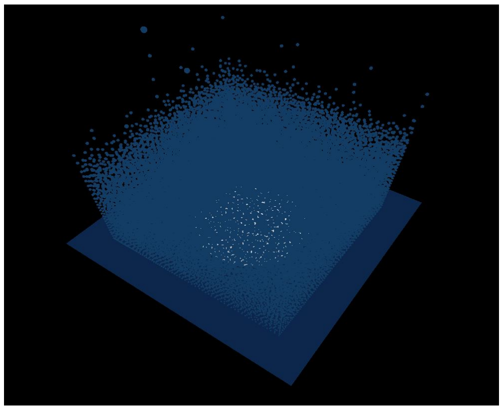


**Figure 2**. Our fluid simulator running with 50,000 particles.

## 3.2. Implementation

Similar to most simulations, collisions detection and response between particles and their enviornment is an integral part of the simulation. There are a variety of common methods to address particle and object collisions in a realistic manner. A common practice to maintain stability is to introduce ghost springs that effectively repel particles away from objects in the eviornment. However, this stability comes at a computational cost. Because our simulation is highly stable relative to other fluid simulations (like SPH), we were afforded the option to resolve collisions by physically displacing particles that collided with its enviornment and reflecting its velocity across the objects normal.

Throughout our implementation, we limit the number of neighbors within the smoothing kernel range. Originally, the paper looks for all the neighbors within the smoothing kernel. In very dense liquids, this could potentially mean many particles. Although this would be most accurate, an exceeding large number of neighbors would have diminishing returns. Since Muller and Macklin note that this simulator should still be stable taking only 20 or so neighbors into account, we chose to hard limit the number of neighbors [0].

Our first implementation takes a naive approach to finding the neighbors: search through the list of randomly generated particles looking for the first *k* particles within the smoothing kernel, resulting in a polynomial time search. In our second implementation, we use a spatial hash grid to sort and find the *k* closest particles within the smoothing kernel. To prevent an eggregious number of invalid reads, we make our hash grid two cells larger than the maximum box size in which the water is simulated. The hash grid, itself implements a very simple linear search for the max neighbor to replace, making finding *k* neighbors theoretically sub-optimal. However, since the whole list of neighbors is fairly small, this linear search is not as bad.

## 4. Results

| | |
|---|---|
| Naive (using vec4 arrays) | 29.80 |
| Naive (using particle struct) | 22.40 |
| Spatial Hash Grid | 42.13 |

**Table 1**. FPS run times for 10,000 particles.

| | |
|---|---|
| Spatial Hash Grid, 10000 particles | 42.13 |
| Spatial Hash Grid, 50000 particles | 14.13 |

**Table 2**. FPS run times on spatial hash grid.

## 5. Discussion

### 5.1. Solver Iterations

Like the paper had mentioned, stable configurations for the water simulation was achievable in 3-4 solver iterations. This is fairly significant as multiple kernels are spawned within each kernel run, decreasing the run time of the simulation altogether. It is also good to note that the time step was generally above $dt = .01$, which some good results when $dt = 0.1$.

### 5.2. Finding $k$ Neighbors

As previously mentioned, our first implementation of the solver uses a naive polynomial search for the first $k$ neighbors within the smoothing kernel. Surprisingly, this method yielded fairly good results, as seen above. Between implementations, we also switched from using multiple vec4 arrays to per particle structs. The former had slightly better results than the latter. This is understandable as the latter array is larger and loses cache coherency as the structs are continguous in memory (causing each corresponding field to be some $n$ bytes apart rather than contiguous themselves).

The latter move to structs was motivated by the spatial hash grid. We implement the spatial hash grid by hashing each cell with its constituent particles. In order to do so, we must sort in a key-pair fashion between the cell index of each particle and the particles themselves. Since all arrays with corresponding data must also be sorted, we opted to use a struct per particle instead. The resulting spatial hash grid worked wonderfully, giving us up to 50+ FPS on particular configurations. It is good to note that the frame rate drops substantially when the particles are close together, which exposes the weakness of a uniform spatial hash grid.

### 5.3. Parameter Tuning

Unlike the paper, which claimed that there could potentially be less finicky parameter tuning using this method, we have found that there is still a sizable amount of parameter tuning involved. While it is not involved as parameter tuning for SPH, it still requires a lot of time.

## 6. Future Work

### 6.1. Meshes

For the time being, we have been using simplistic meshes (ie. cubes, spheres) for collisions. However, we have the framework for obj mesh loading. In the near future, we would like to implement static mesh collisions, and, afterwards, move to rigid body physics with these static meshes interacting the simulated fluids. The former would probably be benefit from a spatial division tree like a stackless kd-tree.

### 6.2. Rendering

Unfortunately, we have been prevented by OpenGL woes from creating extra frame buffers to allow for screen space rendering with curvature flow. In the future, we hope to not only implement this, but also to add screen space caustics to achieve a holistic feel of water [0].

### 6.3. Adaptive Spatial Hash Grid

As noted above, uniform spatial hash grids do not properly optimize, as the structure becomes less helpful as the particles get closer together. We hope to experiment with some kind of adaptive spatial grid to allow for a distributed load on each thread as it search for neighbors.

## References

Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32(4):104:1–104:12, July 2013. 4

Wladimir J. van der Laan, Simon Green, and Miguel Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 91–98, New York, NY, USA, 2009. ACM. 6

http://jcgt.org/published/0002/02/01/

Received:
Recommended: Corresponding Editor: Editor Name
Published: Editor-in-Chief: Morgan McGuire