# SPH-Based Fluid Simulation for Special Effects

Peter Horvath*
David Illes†

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest / Hungary

## Abstract

Simulating natural phenomena like smoke, sand or fluid by physics-based algorithms is a very complex and important task in the visual effects industry. Animating fluids is time-consuming and the result is hard to control. There are only a few commercial fluid simulators on the market, which fit the high expectations of the effect artists. Our goal is to develop a physically correct fluid simulation system using a particle-based approach which is very scalable and flexible for special purposes.

**Keywords:** Special Effects, Particle Systems, Dynamics, Fluid Simulation, Houdini

## 1 Introduction

This paper describes the basic equations of fluid dynamics [1] [2] and covers implementation details of a method called Smoothed Particle Hydrodynamics (SPH) [3]. The particle-based approach reduces the complexity of the simulation because mass conservation equations can be disregarded. The particle system can be used to render the surface of the fluid using spheres, metaballs in the positions of the particles. A surface can also be created using marching-cube based surface generation. The point-type output of the simulation enables the user to add extra points using interpolation techniques or generate the surface even in render-time.

We will show how to implement an SPH-based fluid simulation engine and how to port the system to a high-end 3D animation tool called Houdini. For the integration and scalability we created a fluid simulation plugin using Houdini's SDK.

In Section 2, we describe the basics of fluid simulation. In Section 3, we discuss the Smoothed Particle Hydrodynamics approach. Finally, in Sections 5 and 5.1, we show how we have implemented our SPH-based algorithm.

*hp560hp@gmail.com
†asterix@index.hu

## 2 The Basics

### 2.1 Equations of Fluid Dynamics

The motion of a fluid system can be described with the Navier-Stokes equations. These partial differential equations are usually written in the following form:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u}$$

$$\nabla \cdot \vec{u} = 0$$

The symbol $\vec{u}$ is used for the *velocity* of the fluid, $\rho$ stands for the *density*, $p$ for *pressure*. The letter $g$ is the acceleration due to *gravity*. The $\nu$ is called *kinematic viscosity*. It measures how much the fluid resists deforming while it flows.

The first equation is called *momentum equation* which describes how fluid moves due to the forces. The second equation is the *incompressibility condition*.

For a detailed explanation of the equations see [5].

### 2.2 Lagrangian and Eulerian Viewpoints

For tracking the motion of the fluid we have two possibilities named *Lagrangian* viewpoint and *Eulerian* viewpoint.

The *Lagrangian* approach represents the motion as a finite interpolated system like a given number of particles. Each point is labeled as a particle with position ($\vec{x}$) and velocity ($\vec{v}$). Solids are usually simulated in a *Lagrangian* way.

The *Eulerian* approach looks at fixed points in the volume and measures how the fluid quantities *(density, velocity, pressure etc.)* change in time.

The *Lagrangian* viewpoint corresponds to a particle system [4], the *Eulerian* viewpoint uses a fixed grid that does not change in space.

## 3 The SPH Approach

### 3.1 Particle Systems

For smoke, water splashes or spray effects using particles is the most comfortable solution. A particle system contains a number of particles moving in the space based on

the effect of the surrounding forces. Usually they can collide with each other and with obstacles. Without a particle-particle interaction we call our system a *simple particle system*. Such a system can be implemented efficiently and with a low number of particles it runs real-time. These particles are created at the start of the simulation using a defined volume or generated by emitters. Particles are born at a given rate *(particles/sec)* and die after a certain time. When they get close to the end of their *lifetime*, they disappear.

Without a particle-particle collision model the dynamic properties can be described by a set of decoupled ordinary differential equations:

$$\dot{x_i} = v_i$$
$$\dot{v_i} = f_i/m_i$$

where $x_i$ is the position, $v_i$ is the velocity, $m_i$ the mass of particle $i$ and $f_i$ is the force affecting the particle.

## 3.2 Modelling Fluids Using Particles

SPH is an interpolation method for fluid motion simulation. SPH uses field quantities defined only at discrete particle locations and can be evaluated anywhere in space. SPH distributes quantities in a local neighborhood of the discrete locations using radial symmetrical smoothing kernels.

A scalar value $A$ is interpolated at location $r$ by a weighted sum of contributions from the particles:

$$A_S(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h)$$

where $j$ iterates over all particles in the scene, $m_j$ is the mass of particle $j$. $r_j$ the position, $\rho_j$ the density and $A_j$ the field quantity at $r_j$. The *W(r,h)* is called *smoothing kernel* with core radius $h$. The kernel is normalized if

$$\int W(r)dr = 1.$$

Because $m_i = m$ constant in our case, we can evaluate the density at every step using a modified equation based on [3]:

$$\rho_S(r) = \sum_j m_j \frac{\rho_j}{\rho_j} W(r - r_j, h) = \sum_j m_j W(r - r_j, h).$$

With the SPH approach the derivatives only affect the smoothing kernel. The problem with the method is that these equations are not guaranteed to satisfy some physical rules including symmetry of forces and conservation of momentum. [3] also solves these SPH-related problems.

Using particle-based simulation simplifies the solution of Navier-Stokes equations. Because the number and the mass of particles are constant, mass conservation is guaranteed. The *mass conservation* equation can be omitted. The expression $\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}$ on the left side can be replaced by the substantial derivative $D\vec{u}/D$t because the particles move with the fluid.

Based on simplified Navier-Stokes for the acceleration of particle $i$ we get:

$$a_i = \frac{d\vec{u}_i}{dt} = \frac{f_i}{\rho_i}$$

where $\vec{u_i}$ is the velocity of particle $i$, $f_i$ and $\rho_i$ are the force density field and the density field at the location of particle $i$. The next sections describe how to model the force fields.

## 3.3 Pressure, Viscosity

Instead of an equation described by the SPH rule a modified solution is used for pressure force because it guarantees the symmetry of forces:

$$f_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(r_i - r_j, h)$$

The pressure at particle locations has to be calculated first, which can be computed via the ideal gas equation:

$$p = k\rho$$

where $k$ is a gas constant that depends on the temperature. A modified version - which we used in our implementation - makes the simulation numerically more stable:

$$p = k(\rho - \rho_0).$$

where $\rho_0$ is the rest density. Applying the SPH rule to the viscosity term also yields to asymmetric forces because the velocity field varies. The idea of symmetrizing the expression is using velocity differences:

$$f_i^{viscosity} = \mu \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W(r_i - r_j, h).$$

## 3.4 External Forces

Additional forces can be applied to the particles without using the SPH-method. Gravity or other external forces change the acceleration component of the particles. Particle-object collisions are solved by reflecting the velocity component that is perpendicular to the surface.

## 3.5 Smoothing Kernels

Accuracy of the algorithm highly depends on the smoothing kernels. For our implementation we used the following kernel:

$$W_{poly6}(r,h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2)^3 & 0 \leq \mathrm{r} \leq \mathrm{h} \\ 0 & \text{otherwise} \end{cases}$$

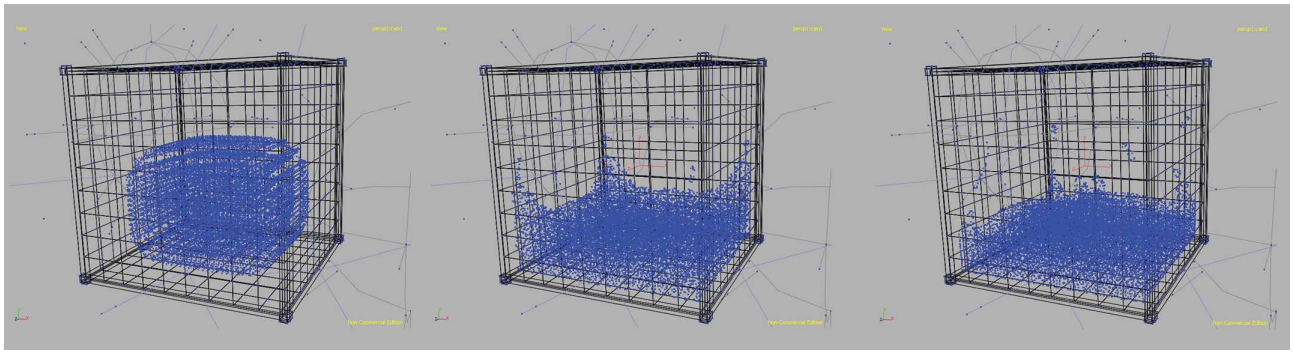

Figure 1: *Fluid simulation results displayed as wireframe in Houdini's viewport. 14 000 particles were simulated and cached 1 frame/sec. The result was used for surface creation*

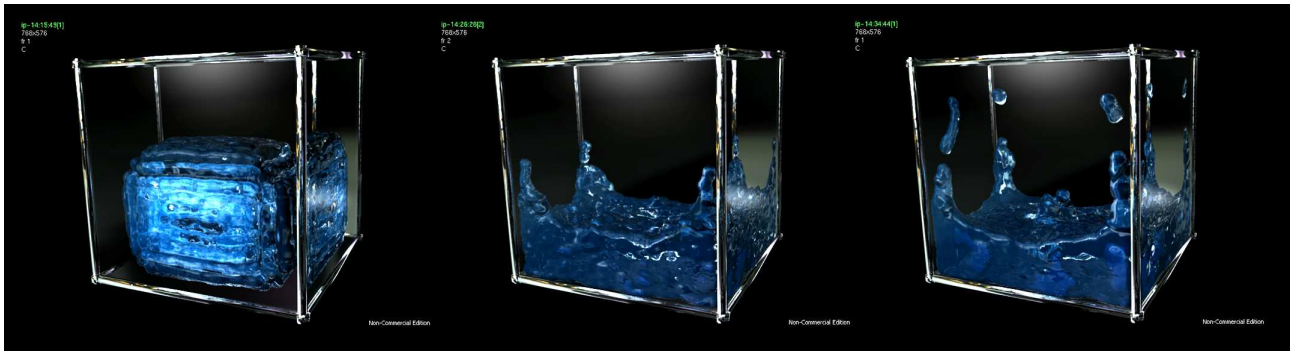

Figure 2: *Fluid simulation results rendered in Houdini's Mantra. The fluid shape is based on 14 000 particles. A 3D Texture was generated for every frame using metaballs in the particle positions and displayed with the Isosurface SOP*

The advantage of this kernel is that $r$ only appears squared which means that it can be evaluated without computing square roots in distance calculations. Debrun's *spiky kernel* solves the problem of our basic kernel, clustering under high pressure. For pressure computing we use the following expression:

$$W_{spiky}(r,h) = \begin{cases} \frac{15}{\pi h^6}(h-r)^3 & 0 \leq \mathrm{r} \leq \mathrm{h} \\ 0 & \text{otherwise} \end{cases}$$

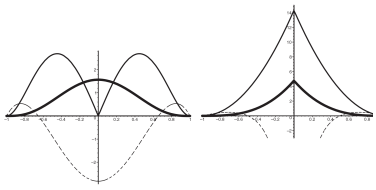

Figure 3: *The smoothing kernels used in our simulations. $W_{poly6}$, $W_{spiky}$ and $W_{viscosity}$. The thick curves represent the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian. Smoothing length parameterer: h = 1*

Viscosity is a phenomenon that is caused by friction decreasing the fluid's kinetic energy by converting it into heat. For two particles that are close to each other, the Laplacian of the smoothed velocity field can cause negative result in forces that increase their relative velocity. For the computation of viscosity forces a third kernel was used because of stability problems:

$$W_{viscosity}(r,h) = \begin{cases} \frac{15}{2\pi h^3}(\frac{-r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1) & 0 \leq \mathrm{r} \leq \mathrm{h} \\ 0 & \text{otherwise} \end{cases}$$

whose Laplacian is positive everywhere with the following expressions:

$$\nabla W(r,h) = \frac{45}{\pi h^6}(h-r)$$

$$W(|r| = h, h) = 0$$

$$\nabla W(|r| = h, h) = \mathbf{0}$$

### 3.6 Improving Performance

Keeping scalability with high particle number is a key factor. In a simulation step the most expensive part is to find the neighbouring particles. We have implemented a simple but effective searching structure. The space where particles can move is divided to grid cells parallel with the axises.

Every grid cell contains references for the included particles. Physically every particle interacts with all the others. Particle neighbours outside of a given distance play an irrelevant role in the computation. For better performance our implementation only considers points inside a predefined radius. This core is quintuple of the particle radius. Cell size is set up based on this core radius. Searching the neighbouring particles needs only checking of the neighbouring cells.

Space partitioning also helps speedig up the calculation of particle-object collision analyzing only the cells close to the geometry's surface.

# 4 Implementation of the SPH Application

We implemented an engine based on the Smoothed Particle Hydrodynamics approach. The stand-alone prototype was written in C++ using OpenGL framework. The application handles more fluid containers interacting with each other. It can display the fluid molecules with different colors depending on the amount of pressure, density or forces. A later version can also simulate rigid bodies. The simulation parameters are read from a file and can be saved to numbered *TGA* sequences. The engine can log simulation parameters and calculation times. External forces and fluid volume were burnt into the code. For this paper we simulated water using 100 000 particles. Each frame took 2.5 *sec* to render with 1 GB RAM on an Intel P4 3 GHz single-core processor without the use of GPU. [7] showed that the use of GPU is not efficient enough in these simulations. An other disadvantage is that today's renderfarms usually do not have GPU included.

A stand-alone system has some limitations. We should import highly complex geometry and scene data for high-end quality simulations like characters deformed by bones in a pool representing 10 millions of particles. Handling this complexity and keeping scalability is a huge task. Our goal was to test multiple methods before integrating the algorithms into a system as a plugin.

# 5 Implementation of the SPH Houdini Plugin

Side Effect's Houdini [8] is maybe the most complete visual effects solution available. This award-winning and production-proven product handles the most challenging shots using a node-based procedural workflow. The Apprentice version lets you learn about the possibilities with some limitations. With the learning edition users have access to the Houdini Development Kit which gives the freedom adding new tools. This section contains details about how developers can widen the power of the software using third-party plugins like fluid solver.

## 5.1 Effect Creation Using Procedural Workflow

The base concepts behind Houdini are different from other 3d animation tools. In Houdini effects are created using node operators connected into each other like an acyclic graph. There are several node types based on their functions like surface operators *(SOPs)*, particle operators *(POPs)*, shader operators *(SHOPs)* etc. Each operator has an output that can be passed to the next operator as input. Operators optionally have one or more inputs. For example, a *Copy* operator has a *primitive* input and a *template* input. The operator's output is a list of primitives placed on the template positions. If the primitive is a sphere and the template is a particle system, the output is spheres placed in the location of the particles. The strength of this method is that each step of the effect can be changed and complex effects can be mixed from the nodes usually without programming knowledge.

## 5.2 Results

We have been implementing a *SOP* Houdini plugin using the application's SDK toolkit compiled in Visual Studio 2003 as a DLL with the toolkit's custom compiler script. The plugin is loaded during the startup and can be used with other *SOP* nodes.

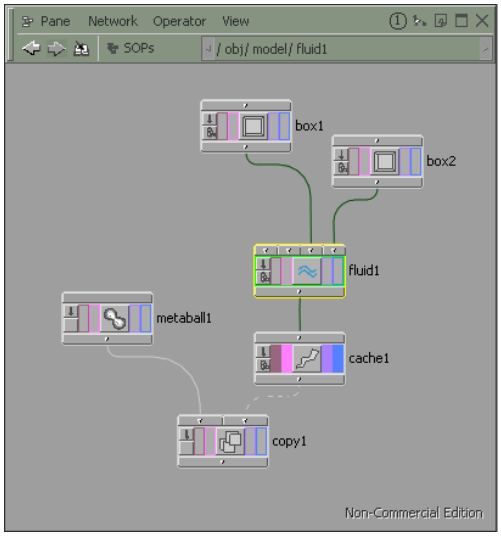


Figure 4: *The Fluid node with a box source geometry and a bounding box collision object. Simulation results are cached and metaballs are placed in the position of the particles*

The node has 4 inputs. 2 fluid container inputs help building different water volumes creating a tree from the nodes. There are also a source geometry and a collision input. The node has point output, the location of the particles are passed to the next node. The output can be cached to hard disk using the *Cache SOP* operator (Figure 4),

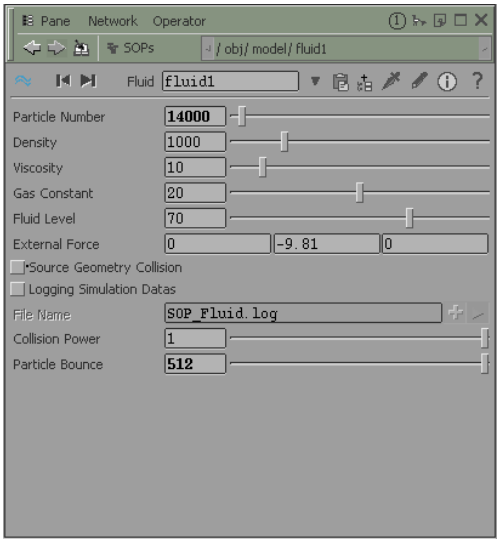


Figure 5: *User Interface of the Houdini fluid simulation plugin. Users can change fluid parameters, collision properties and are able to log simulation details*

surfaces can be generated on the points with the help of shaders and the *Isosurface* node.

Fluid properties can be modified on the interface of the plugin (Figure 5). The user can control the particle number, the fluid density, viscosity and a constant parameter called gas constant. External forces can be set and changed in time. Fluid volume is determined by a geometry input, which is filled with fluid using the given *Fluid Level* parameter. It can be set whether the fluid interact with this volume. Strength of object-particle collision can be controlled.

Figure 6: *Water collides with a torus inside a box. Fluid is represented by 30 000 particles. A simulation step was calculated in 2 secs*

Controllable fluid shape was necessary in the Houdini plugin. This - of course - resulted decrease of the performance. The calculation time of a frame became double than the stand-alone prototype, which was optimized for tube geometry. To offset this loss we have implemented two types of collision detection. In Houdini there are several primitive types. For the polygon primitives we use the simple mechanism of particle collision similar to the stand-alone version. We always predicate the next position of a particle before moving and check in which half of the space is it by the side of the primitive. If we detect an emigrant particle then it's acceleration vector is modified, so elements can not leave the container's geometry. For computing the return force we just need the distance of the next position of the particle from the surface of the primitive that can be detemined with a dot product. The other method of the collision detection works with rays for NURBS- and Bezier- primitives (Figure 6).

## 6 Conclusion and Future Work

The Houdini plugin is still under development. We are planning to extend it's capabilities. In the near future the plugin will have an additional force input that can handle complex forces - like wind, spiral and vortex forces - generated with the application's *Force* node. Particle-object collision algorithm is planned to reimplement with the extensive use of the grid structure only calculating collision for the points that are close to the surface.

The stand-alone implementation can simulate basic rigid body scenes [6]. Rigid bodies are constructed with a particle filling method. The geometry behaving like rigid body is filled with particles representing the rigid body properties. The *Fluid* node will be extended to handle rigid body simulation.

Our plan is to run the fluid simulation distributed on multiple machines. The volume-based distribution will simulate complex fluids by dividing the fluid with the grid cells.

Future tests will include scenes above 100 000 particles and complex geometries.

## References

[1] G. K. Batchelor. An Introduction to Fluid Dynamics. Cambridge University Press, 1967.

[2] D. Pnueli and C. Gutfinger. Fluid Mechanics. Cambridge Univ. Press, NY, 1992.

[3] M. Müller, D. Charypar, M. Gross. Particle-Bsed Fluid Simulation for Interactive Applications. *E*urographics/SIGGRAPH Symposium on Computer Animation, 2003.

[4] W. T. Reeves. Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics 2(2)*, pp. 91-108, 1983.

[5] R. Bridson, M. Müller-Fischer, E. Guendelman. Fluid Simulation. *(*Siggraph 2006 Course Notes), pp. 2-11

[6] T. Takahashi, U. Heihachi, A. Kunimatsu, and H. Fujii. The Simulation of Fluid - Rigid Body Interaction. ACM Siggraph Sketches & Applications, July 2002.

[7] K. Fatahalian, J. Sugerman, P. Hanrahan. Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication. *Graphics Hardware*, 2004.

[8] Houdini 3D Animation Tool. http://www.sidefx.com