## PARTICLE-BASED FLUID SIMULATION FOR INTERACTIVE APPLICATIONS

**Tiarnán McNulty, 13311241**

**26th March, 2014**

A video overview of the entire demo can be viewed at: <http://youtu.be/pskGb-IP1qE>

# Introduction

Fluid simulation takes the form of one of two approaches: Lagrangian or Eulerian. Lagrangian is particle-based and is used here, while Eulerian is grid-based and won’t be covered. However, the Stable Fluids paper (Stam, 1999) provides an excellent introduction to Eulerian methods, if you find that area to be of interest. This report aims to provide an overview for implementing your own fluid simulation using Smoothed Particle Hydrodynamics. SPH allows for the simulation of fluids with free surfaces, and the particles can be additionally used for rendering the fluid. The maths behind SPH is well explained in the paper, so the focus here will be on how the mathematical equations and processes correspond to matching C++ code and functions.

The theory will be touched on throughout this report, but the original paper remains required reading (Müller, Charypar, & Gross, 2003) as it provides a comprehensive understanding that cannot otherwise be conveyed. Additionally, (Melager, 2006) and (Staubach, 2010) both crafted detailed thesis which cover all the topics covered by the original paper, and more. The main focus for this report will be in the implementation details, the optimisation methods, and the rendering process, as this is something that is not covered too well by the other sources. As such, combining this report with the above should hopefully provide a robust understanding of the hows and whys of Smoothed Particle Hydrodynamics.

# Background

Like with a lot of real-time methods, SPH takes the concept of Computational Fluid Dynamics and simplifies it. This simplification makes it significantly less accurate, but it retains a high level of stability and is still precise enough to be suitable for use in games. The final result is a simulation which runs in real-time, while still behaving much like a user would expect.

SPH, much like CFD, make extensive use of the Navier-Stokes Equations that describe fluid dynamics. In addition to this, a continuity equation to describe mass conservation and a state equation for energy conservation are used.

# Paper Contribution

As already mentioned, the paper proposes a method for fluid simulation which builds upon previous SPH work. The impact of viscosity and pressure forces are derived using Navier-Stokes, and surface tension is also modelled. A set of highly efficient smoothing kernels, e.g. a technique for estimating a value by using its noisy observation, are also presented, to achieve interactive frame-rates.

# Smoothed Particle Hydrodynamics

SPH provides a method for interpolating particle systems. It is not possible to simulation individual droplets of fluid in real-time, so each particle in the system is instead simulating a much larger collection of elements. However, SPH allows for quantities that are only defined at each discrete particle location to be evaluated anywhere in 3D space, allowing for the individual particles to collectively act as one much larger, and fuller, fluid volume.

Using SPH, a scalar quantity for a given field (e.g. pressure field) can be interpolated to location by using the sum of a weighted contribution of all particles in the system as follows:

is a iterator over all particles, is particle ’s mass, its density, its position, and is the quantity of your desired field at position . is the field’s smoothing kernel, where is the kernel’s support radius and if r is greater than h, then particle can be considered to have no impact on .

As mentioned previously, the particles are collectively representing a volume of fluid. As such, the total volume of the fluid is:

The mass of the volume, remains constant (as we are simulating incompressible fluids?) but the total volume, and thus the total density,, can change. The mass of the volume, and thus mass of each particle, can be calculated at run-time, but the density of each particle will vary and so needs to be evaluated for each time step.

While I said that this report would focus on implementation, I wanted to spend a brief moment presenting the initial SPH equation in its traditional form, as this, and all the other equations, can be coded in much a similar fashion.

Essentially, we can use these equations to find out our variables, and the format of them can indicate the structure for our program. For instance, is the mathematical equivalent of a for loop, while is a function. To that end, the remainder of this report will simply present coding equivalents for the equations used in the paper, which can be cross-referenced as needed.

# Basic Outline

The very first thing our simulation needs is the particles, and we already know from the above equation what a particle needs to contain.



This might initially seem like a large number of variables, but as optimisation can greatly increase the number of particles a system can support the reality is that only a few core variables are key, while the others are combinations of these variables stored to save multiple calculates per particle per frame.

Essentially, we use the pressure, viscosity and surface forces to calculate the new pressure, density, position and velocity of the particle. You might recall that the smoothing kernels use a support radius, , to determine if one particle influences another. This could be an operation but the vector of neighbour particles only contains influences, so this is another optimisation.

Now that we have our particles, we need another class to handle the simulation itself. It would be ideal to be able to simulate various different fluids, so it is recommended to create an abstract base fluid simulation class to hold the simulation logic, which can then be inherited by different fluid types which provide their own parameters. To that end, the class should look something like this:



This might seem somewhat overwhelming, but it will be explained in due time. Again, a large number of variables are either for optimisation or are constants that control how different fluid types behave.

# Simulation Update

A time step begins with the gathering of all of a particles influential neighbours, and storing them in its neighbours vector. A hash map works very well for this, as spatial hashing provides an efficient way of finding adjacent particles.

Once gathered, iterate through all the particles in the scene and begin to accumulate the particle’s new density.

The smoothing kernel for density can be calculated using the following method:



Where the DefaultKernelConstant (one of our pre-calculated optimisations) is:



KernelPow9 is also pre-calculated, and is the smoothing kernel radius raised to the given power.

Once density has been accumulated, the pressure acting on a particle is simply its density multiplied by a stiffness constant. Once we know the density and pressure for all particles , we can calculate the pressure, viscosity and surface forces by taking this knowledge into account as we iterate over all the particles again.

To calculate the pressure and viscosity forces we compare each particle to all its neighbours, but crucially we don’t, unlike in the initial pressure and density calculations, compare the particle against itself.



The pressure and viscosity forces can be calculated within a single iteration of neighbours:

Once the viscosity force has been accumulated, it must then be multiplied by a viscosity constant to ensure the fluid behaves as its parameters indicate. The corresponding kernals are calculated as below:



Next up, we want to calculate the inward surface normal. This will tell us if a particle is a surface particle and if it is then we can apply an appropriate surface force to it. Iterating over all neighbours, this time including the particle itself, the surface normal is accumulated by the following:

Once fully accumulated, if the magnitude of the surface normal is above a pre-calculated surface threshold, then we identify it as a surface particle and proceed to calculate the surface force using yet another smoothing kernel. Again, we iterate over all neighbours, including itself.



Once accumulated, the surface force is derived from:



Now that all the necessary variables have been derived, we must update our simulation’s state. The particle’s new acceleration is generated from:

Leapfrog integration is then used to update the particle’s position and velocity, as follows:



And that’s it! Well, not quite, there is still a lot of work to do in regards to making sure everything runs correctly, but that’s a crash course on all the core elements you need to start your fluid simulation running.