## PARTICLE-BASED FLUID SIMULATION FOR INTERACTIVE APPLICATIONS

**Tiarnán McNulty, 13311241**

**31st 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 is covered by the Stable Fluids paper (Stam, 1999). The paper chosen for study is “PARTICLE-BASED FLUID SIMULATION FOR INTERACTIVE APPLICATIONS” (Müller, Charypar, & Gross, 2003), and this report attempts to provide some supplementary information to help fully understand it, as well as offer insight for starting your own implementation. While the paper is written very much from a mathematician’s background, this report attempts to look at it from a coder’s perspective.

Unlike previous attempts, the paper models fluid simulation using Smoothed Particle Hydrodynamics. SPH allows for the simulation of fluids with free surfaces, and is very visually inclined. The particles used for calculations can also be used to render the fluid, and this provides a very easy visualisation to confirm if an implementation is correct or not. The mathematical equations behind SPH is well explained in the paper, and (Melager, 2006) and (Staubach, 2010) both crafted detailed thesis which expand on it further, so the focus here will be on how the mathematical equations and processes correspond to matching code, functions and classes. As such, combining this report with the above should hopefully provide a robust introduction to Smoothed Particle Hydrodynamics.

# Paper Contribution

SPH was initially introduced in 1977 to solve astrophysical problems, and has been used in computer graphics since 1995, but this paper is notable for being the first to apply it to the simulation of fluids. Since each particle is given an unchangeable mass, a major benefit of SPH is that it guarantees the conservation of mass without the need to explicitly compute it, simplifying various calculations. SPH is also mesh-less, and creates fluids with free surfaces, e.g. there is no need to track boundaries as in grid-based implementations. Being the first paper to introduce the idea, it proves a comprehensive overview of the entire idea, including methods for using Navier-Stokes equations to calculate viscosity, pressure and surface tension forces. They also propose smoothing kernel equations explicitly designed for real-time applications, which can be translated into code and used directly in an implementation.

While more recent papers have suggest more efficient methods for handling SPH, such as offloading it to the GPU or allow it to handle bigger time-steps, they all assume a prior understanding of the source and so this paper’s contribution to the field of real-time fluid simulation cannot be understated.

However, that said, a downside to SPH is that the resolution of the fluid is directly tied to the number of particles, so optimisation to ensure a high particle count is key to any satisfactory implementation.

# 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.

# Particle

A particle needs a position and a velocity, with its position being updated based its velocity, and velocity being updated by acceleration. A particle’s acceleration is determined by the forces acting upon it, and in a SPH system there are three core forces which must be considered.

## Pressure Forces

The pressure force acting upon a particle is determined by the pressure of its neighbours, but to calculate the pressure of a particle we must first know its density. As each particle is representing a drop of fluid, the density of each particle can actually change as fluid moves around the volume. This is calculated using the initial smoothing kernel presented in the paper, but essentially means that very dense, high pressure particles will push each other apart as they represent a large concentration of fluid that wants to spread out over a wider area.

## Viscosity Forces

The viscosity force represents the internal friction of a fluid, but in SPH can be considered the attraction force, in contrast to the repulsive pressure force. The viscosity factor of a fluid, a user defined constant, determines how strongly particles will resist being pushed apart. This is represented by yet another smoothing kernel, but the implication is that sparse, low pressure particles will attract their neighbours in an attempt to reach a balance with all particles sharing the same density and pressure.

## Surface Forces

This, force represents the forces that would be acting on the surface and helps bring SPH into a natural rest state. Unlike the other SPH forces, surfaces forces only act on the surface particles, so a key component of applying this force is determining which particles should be affected by it. Again, the paper proves equations which translate well to an implementation.

## External Forces

A final force which is likely to appear in any implementation, but is actually separate from SPH itself, is an external one. This covers the cumulative impact of any forces which aren’t identified above, with gravity being the most common, although any collision forces would also fall under this bracket. External forces are applied directly to the particles themselves, much like in a rigid body simulation, and remain separate to any SPH implementation.

# Fluid Simulator

A good outline can greatly help the process for translating the various equations into a working project, but while the equations might initially seem confusing they pretty much entirely resolve into loops which iterate over each particle and its neighbours. This can be done in a brute force manner, where you check each particle against every other particle to see if they are close enough, or through the use of a more complex data structure, such as a hash map, to generate and store neighbours for each particle. This method greatly improves performance and is strongly recommended, but as the particles move around care must be taken to ensure the hash map is regenerated each frame. Of special note is that for the purposes of SPH a particle must occasionally also consider itself, so an array or neighbours must actually include itself too.

## Taking a Time Step

An interesting quirk of SPH is that it doesn’t use the same delta time step that your program might use. To ensure it remains stable, your fluid simulation should have its own smaller time step which it advances along each update. (Explain why)

## Calculating Particle Pressure

As mentioned above, the first thing your want to do is gather the neighbours for each particle. Once that is complete, you want to iterate over each particle, and its neighbours, to accumulate its density, using the default smoothing kernel. Once complete the pressure can be calculated with a simple sum of pressure = StiffnessConstant \* (density).

## Calculating Internal Forces

As the forces require the pressure of particle neighbours, you must do a full iteration of each particle to generate the pressure and then begin another entirely new iteration to calculate the internal forces acting upon it. To determine if a surface force needs to be applied, you need to calculate the inward surface normal (e.g. from the particle towards the fluid) and if the magnitude is above a given threshold (this can be computed, but a constant works just as well in a lot of cases) it can be considered on the surface and the force applied to bind the fluid together. The smoothing kernels for all these forces is provided, but a potential pitfall arises as the pressure and viscosity forces do not consider the current particle (so it should be skipped in the neighbour iteration) while the surface normal calculations do.

## Calculating External Forces

As surprising as it might seem, the above forces are all you need for your SPH. However, external forces prove to be a crucial element in how we perceive realistic fluids, so the next step is to accumulate those. We don’t consider collision forces at this point, so only “constant” forces such as gravity, wind etc. are added together at this point. Ensure that you consider the density of a particle when applying gravity, if you want it to remain a constant.

## Leapfrog Integration

Once you’ve calculated your external forces, even within the same loop if you would like, you can calculate the particle’s acceleration. As you might have surmised, this is simply acceleration = (forceInternal + forceExternal) / density.

Once you’ve calculated the acceleration it is time to integrate and advance the simulation by a single step. The paper recommends leapfrog integration and proves no further context, but it is actually a pretty simply concept. Its name derives from the fact that the velocities leap over positions, (and vice-versa) and requires that you sold both the current and old particle velocities. The calculation of a new velocity uses newVelocity = oldVelocity + FluidStep \* acceleration which is not the same as using newVelocity = currentVelocity + FluidStep \* acceleration even though currentVelocity is what is used in some smoothing kernel calculations. This can be tricky to get right, and as this is the one area the paper doesn’t provide any insight some pseudocode is provided.



Once integrated you can then do any collisions checks, as you normally would, and apply them as you see fit. One important caveat, however, is that any collision force needs to not only update the current velocity but the previous one as well. Since both are used in the integration, as shown above, only altering one will cause the particle’s ensuing state to be incorrect.

## Fluid Simulator Structure

And that’s it, essentially. Although the equations might initially seem daunting, knowing the rough structure of an implementation can greatly ease the translation process. It is advised to create both Particle and Simulation classes, to encapsulate all information. The paper presents a large number of constants in which you can simply choose values which work for the fluid you would like to create, and the referenced thesis present some good default values for these constants, although iteration will be needed to find the combination that works best for any given implementation, However, one benefit to this is that simply inheriting from the simulation base class and changing the constants allows for the quick creation of a variety of different fluid types.

# Additional Implementation

Any implementation is going to need some form of collision detection, as otherwise the fluid will simply flow uncontrollably. A box is traditionally used, but a capsule proves to be an interesting alternative that can be moved and rotated freely as a clever approach ensure the collision checks aren’t too intensive. That said, even the simplest of collision checks will have a noticeable impact on the amount of particles your simulation can handle, as each particle needs to be checked in the method used here. If you assume a capsule is defined simply by a start point, end point and a radius, then collisions can become almost as simple as sphere checks. Taking the start point and end point as a line, then for a given point, x, we want to uncover the closet point, cp, on the capsule line which is the shortest distance to x. Comparing the distance (|x – cp|) against the capsule radius tells us if x is inside, on, or outside the capsule, and (x – cp) even becomes te collision normal to use in a response.

You can then create container capsules by simply ensure the particles remain within its radius, and collider capsules by ensuring particles remain outside them.

# Optimisation

It cannot be stressed enough about how important optimisation is for an SPH implementation. When dealing with thousands of particles even the smallest of improvement inside a loop can have a massive impact. For example, the smoothing kernels which the paper presents have a lot of constants which can be calculated once and stored. Similarly, kernels also share some calculations, such as distance or pressure differences, which use expensive divisions or exponential operations but can be computed once and then passed into them all.

# Works Cited

Melager, M. (2006). *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics.* Denmark: Department of Computer Science, University of Copenhagen.

Müller, M., Charypar, D., & Gross, M. (2003). Particle-based fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '03)* (pp. 154-159). Aire-la-Ville, Switzerland: Eurographics Association.

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

Staubach, D. (2010). *Smoothed Particle Hydrodynamics - Real-Time Fluid Simulation Approach.* Erlangen, Germany: Department of Computer Science, The University of Erlangen-Nuremberg.