Smoothed Particle Hydrodynamics with Position Based Dynamics

Leon Montealegre*

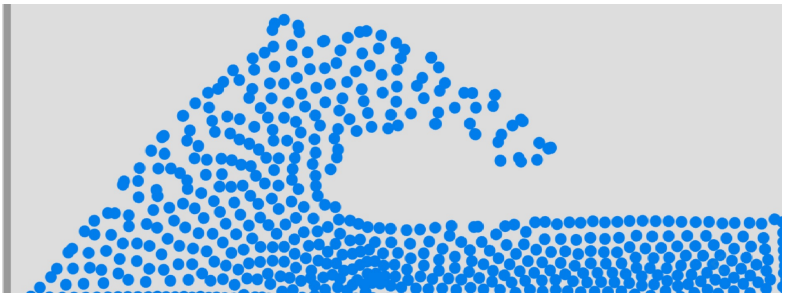

Figure 1: *A frame showing a wave*

Abstract

In this paper, I will describe my implementation of simulating fluid dynamics using Smoothed Particle Hydrodynamics (SPH) from the previous work of [Müller et al. 2003] and Particle Based Dynamics (PBD) from [Macklin and Müller 2013]. The goal is to have a quick and stable algorithm for the use of real-time applications.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: fluid simulation, SPH, PBD

1 Introduction

1.1 SPH

Smoothed Particle Hydrodynamics (SPH) is a generalized method of simulating different physical materials (fluids, gases, soft bodies, and even rigid bodies) through the use of discretized, finite particles. It was originally introduced by [Gingold and Monaghan 1977] for use in astrophysics but has recently been the target of real-time applications for physics simulation.
SPH has many attractive features. Due to its discretization as particles, it is a very intuitive way to simulate physics since each particle can be thought of as a small ball. Another great feature is that each particle is mostly independent after a certain step which means that it can be easily parallelized.

1.2 PBD

While SPH has many attractive features, the main drawback is that it's quite unstable. SPH relies on integration schemes to move particles from one time to the next which, by definition, introduce instability and accuracy errors.
Position based dynamics is a method introduced by [Müller et al. 2007] for use in cloth and soft-body simulation. Then, [Macklin and Müller 2013] built off of PBD and used it for fluid simulation. PBD aims to get rid of explicit integration schemes and replace them with a set of rules or "constraints" that are satisfied using Gauss-Seidel iteration which is a method quite similar to Provot correction.

*e-mail:leonm99@gmail.com

2 Motivation

My main motivation for choosing this topic and working on this project was due to NVIDIA's PhysX FleX system. FleX is NVIDIA's unified particle system. I initially saw their results a few years ago and I was blown away. I became obsessed with watching fluid simulation videos and crashing waves. I was still a pretty novice programmer at that point but I decided to try and see if I could implement my own fluid simulation. I could not. But it is pretty incredible to be able to look back now, after I've fully implemented it and see how much I've learned and grown as a programmer. I never thought I'd be at the point to understand the Navier-Stokes equation but here I am with a pretty general idea of what it represents.

3 SPH

This section will discuss the more technical aspects and details of SPH along with some of the mathematics used to implement it.

3.1 Idea

Since each particle is discretized, they represent an amount of volume in that fluid. From this, conservation of mass is a given which dramatically simplifies many calculations. Each particle has discretized "attributes" which are interpolated to evaluate anywhere in the system. These attributes are interpolated via normalized "smoothing kernel" functions denoted as $W(x, h)$.

3.2 Attributes

Each particle holds a few key attributes which are used to satisfy the underlying Navier-Stokes equation.

$$\rho \frac{\partial u}{\partial t} = -\nabla p + \mu \nabla^2 u + \rho g \tag{1}$$

In layman's terms, (1) is a $F = ma$ equation. u is the velocity of a particle, so $\frac{\partial u}{\partial t}$ is the acceleration of the particle. The right hand side of the equation is then the forces that dictate the motion of the particle and are what need to be solved.

From this, the two primary attributes can be found, ρ and p. ρ is the density of the particle and p is the pressure.

To evaluate an attribute, we use a weighted sum of the attributes from the surrounding neighbors of the current particle using the "smoothing kernel". We want particles that are closer to us to contribute more than particles that are further away from us, so the smoothing kernel falls off as a function of distance. And since particles very far away have little to no effect on us, we define a radius h which is a cut-off point for particles to have an effect. Therefore, we have a set of "neighbor" particles that we need to find to calculate an attribute. Formally, this can be written as:

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W(x_{ij}, h) \tag{2}$$

where $x_{ij} = x_i - x_j$ and $|x_{ij}| < h$

$\frac{m_j}{\rho_j}$ represents the amount of volume the particle holds, so bigger particles have a bigger effect on the attribute.

The great thing about this formulation of attributes is that the gradient and laplacian of them are trivial to compute:

$$\nabla A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla W(x_{ij}, h) \tag{3}$$

$$\nabla^2 A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W(x_{ij}, h) \tag{4}$$

3.2.1 Density

For density, $A_i = \rho_i$

$$\Rightarrow \rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W(x_{ij}, h) = \sum_j m_j W(x_{ij}, h) \tag{5}$$

3.2.2 Pressure

From, the ideal gas state equation:

$$p = k\rho \tag{6}$$

where k is a gas constant that depends on the temperature. A modified version is suggested by the paper [Müller et al. 2003]

$$p = k(\rho - \rho_0) \tag{7}$$

where ρ_0 is the rest density of the particle.

3.3 Forces

Since the left-hand-side of the Navier-Stokes equation (1) is the acceleration of the particle, the right-hand-side describes the forces.

3.3.1 Pressure

The pressure term is described as $-\nabla p$ which yields

$$f_i^{pressure} = -\nabla p(x_i) = -\sum_j \frac{m_j}{\rho_j} p_j \nabla W(x_{ij}, h) \tag{8}$$

This term, however, is not symmetric which is a problem for numerical accuracy and stability. So, with some hand-waving the force can be described as:

$$f_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(x_{ij}, h) \tag{9}$$

Which is the average of the two pressures.

3.3.2 Viscosity

The viscosity term is described as $\mu\nabla^2 u$ which also yields asymmetric forces

$$f_i^{viscosity} = \mu\nabla^2 u(x_i) = \mu \sum_j \frac{m_j}{\rho_j} u_j \nabla^2 W(x_{ij}, h) \tag{10}$$

Which can be symmetrized as follows:

$$f_i^{viscosity} = \mu \sum_j m_j \frac{u_j - u_i}{\rho_j} \nabla^2 W(x_{ij}, h) \tag{11}$$

3.3.3 External

The dominating external force is gravity, which is simply modeled by

$$f_i^{gravity} = \rho_i g \tag{12}$$

where g is the acceleration due to gravity.
For collisions, I implemented basic collision forces with planes defined as "Walls". Each wall has a position, x_i, and normal n_i and the force on a particle i from wall j is

$$f_i^{walls} = \sum_j (K n_j (x_{ji} \cdot n_j) + D n_j (u_i \cdot n_j)) \tag{13}$$

where K is a sort of "spring constant" for the wall that dictates how much it pushes on the particle and D is a damping coefficient that describes how much energy the particle will lose ($D = -1$ is a perfectly elastic collision)

3.3.4 Other

There are some other, more complex forces that are described by [Müller et al. 2003] such as the surface tension force and some other stabilizing forces like "vorticity". I did implement the surface tension force but the math behind it is beyond the scope of this paper.

$$f^{surface} = -\sigma\nabla^2 c_s \frac{n}{|n|} \tag{14}$$

where σ is a surface tension constant, $n = \nabla c_s$ and is the surface normal pointing into the fluid and

$$c_s(x) = \sum_j m_j \frac{1}{\rho_j} W(x_{ij}, h) \tag{15}$$

4 PBD

This section will briefly discuss the more technical aspects and details of PBD, skipping over the more complicated mathematics.

4.1 Idea

The general idea of PBD is to replace explicit integration schemes through the use of "constraints". I found it easiest to understand constraints through the simplest example, distance constraints.

4.2 Constraints

A constraint is denoted as $C_i(p_1, \ldots, p_n)$ where $p_i, \ldots, p_n$ are the positions of the particles that this constraint applies to. The constraint is also setup to be satisfied when $C_i = 0$.

4.2.1 Distance Constraint

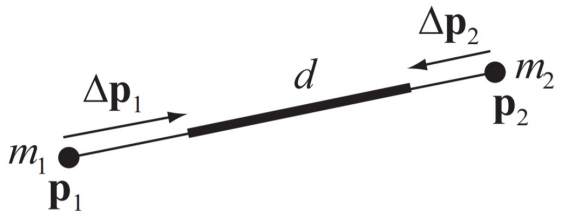

Figure 2: *Distance Constraint*

For a distance constraint, the constraint is simply as follows:

$$C(p_i, p_j) = |p_i - p_j| - d_0 \tag{16}$$

Where d_0 is a "rest distance". This constraint is satisfied when $|p_i - p_j| = d_0$, in other terms - when the distance between particle i and j is d_0.
This is a more formal form of Provot correction.
The constraint solver then aims to satisfy all of these constraints. So for cloth, there would be a set of constraints for each spring (structural, shear, and flexion) and the solver would iterate through each and solve each one individually.

To solve the constraint, we attempt to find a Δx such that

$$C(p + \Delta p) = 0 \tag{17}$$

The math is a bit messy, but the general idea is to do a Taylor series expansion and use Newton's method to approximate a solution. This leads to the following

$$\Delta p_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(p_{ij}, h) \tag{18}$$

where

$$\lambda_i = -\frac{C_i(p_1, \ldots, p_n)}{\sum_k |\nabla_{p_k} C_i|^2 + \epsilon} \tag{19}$$

where ϵ is a "relaxation constant" which dictates how forceful the constraint solver is.

5 Algorithm

In this section I will outline the algorithm I use for the fluid simulation along with explanations of important details.

Algorithm 1: Simulation Loop

foreach *particle i* **do**
  find neighbors N_i
  calculate ρ, p
end
foreach *particle i* **do**
  calculate $f_{pressure}, f_{viscosity}, f_{surface}, f_{external}$
end
foreach *particle i* **do**
  apply forces $v_i = v_i + \Delta t f_{total}$
  predict position $x_i^* = x_i + \Delta t v_i$
end
while $iter < solverIterations$ **do**
  foreach *particle i* **do**
    calculate λ_i
  end
  foreach *particle i* **do**
    calculate Δp_i
  end
  foreach *particle i* **do**
    update predicted position $x_i^* = x_i^* + \Delta p_i$
  end
end
foreach *particle i* **do**
  update velocity $v_i = \frac{1}{\Delta t}(x_i^* - x_i)$
  update position $x_i = x_i^*$
end

5.1 Finding Neighbors

Finding neighbors is the most computationally expensive part of SPH and PBD. I began with a general, brute-force $O(n^2)$ approach of looping through each other particle in the scene and performing a distance test. This performs decently enough for particles amounts under 400 and so I stuck with it for a majority of my debugging. Eventually, as performance costs rose and I began to make the move from 2D to 3D, this approach was not going to cut it.

There are many ways of going about this efficiently, but the most obvious approach is some sort of spatial data structure. I chose to use an octree due to its simplicity to implement and fairly decent performance. There are definitely faster methods of neighbor searching such as a radix sort or counting sort which partition neighbors into a flattened array for easy GPU look-up.

6 Rendering

For my 2D scenarios, rendering is trivially done through JavaScript's Canvas API and each particle is drawn as a circle.

For the 3D scenarios, rendering is done with OpenGL and GLFW. Each particle is drawn as a 3D sphere with basic diffuse shading. Initially I had a render call for each particle but then I implemented instance rendering so all the particles are drawn in a single render call.

There are definitely much prettier methods of rendering such as Ellipsoid Splatting but I did not have the time to implement them.

7 Future Work + Reflection

There are definitely many potential extensions for this simulator. The most exciting is porting the simulation steps onto the GPU using compute shaders. This has been done in most of the papers I have used and they support millions of particles in real time which is incredible.
Due to my use of PBD, including other types of materials such as rigid bodies, soft bodies, and even gases would be a fairly trivial extension of the existing code.
It would also be very interesting to compare the running times of different scenes with different material properties.

Overall, this project took well over 80 hours (maybe even 100). I did everything and honestly it was pretty fun.

8 Known Bugs

PBD's main goal is to eliminate compressibility in the fluid and enforce the rest density. Most of my results are pretty bouncy as many people have pointed out. This is partly due to the lack of clarity and detail on a majority of my resources. There are a few places where it's ambiguous as to whether or not a particle should include itself as a neighbor and changing that seems to make the simulation a lot less compressible but a lot less stable as well. I believe if I were able to simulate tens of thousands of particles this problem would resolve but the maximum number I was able to reach was around 2500.

And while it's a bit too bouncy, it is incredibly stable and very rarely blows up which I personally believe is more important.

9 Kernel Functions and Constant Values

9.1 Kernels

There are different kernel functions for different forces. The kernels are chosen fairly arbitrarily but must be normalized (i.e. $\int_0^\infty W(x,h)dx = 1$) and must be $= 0$ at $|x| = h$.

9.1.1 Wpoly6

Used for most instances of the kernel function.

$$W_{poly6}(x,h) = \frac{315}{64\pi h^9}\begin{cases}(h^2-|x|^2)^3 & 0 \le |x| \le h \\ 0 & \text{otherwise}\end{cases} \quad (20)$$

$$\nabla W_{poly6}(x,h) = \frac{-945}{32\pi h^9}x\begin{cases}(h^2-|x|^2)^2 & 0 \le |x| \le h \\ 0 & \text{otherwise}\end{cases} \quad (21)$$

$$\nabla^2 W_{poly6}(x,h) = \frac{-45}{\pi h^6}\begin{cases}(h^2-|x|^2)(3h^2-7|x|^2) & 0 \le |x| \le h \\ 0 & \text{otherwise}\end{cases} \quad (22)$$

9.1.2 Wspiky

Used only as a gradient for the calculation of the force for the pressure term.

$$\nabla W_{spiky}(x,h) = \frac{-45}{\pi h^6}\begin{cases}\frac{(h-|r|)^2}{|r|} & 0 \le |x| \le h \\ 0 & \text{otherwise}\end{cases} \quad (23)$$

9.1.3 Wviscosity

Used only as a laplacian for the calculation of the force for the viscosity term.

$$\nabla^2 W_{viscosity}(x,h) = \frac{45}{\pi h^6}x\begin{cases}(h-|r|) & 0 \le |x| \le h \\ 0 & \text{otherwise}\end{cases} \quad (24)$$

9.2 Constants

Here's a list of all the constants and values for each that I use in a majority of my demos.

$h = 0.0457$
$\Delta t = 0.01$
$solverIterations = 5$
$\epsilon = 5000$
$\sigma = 0.0728$
$k = 3$
$\rho_0 = 998.29$
$\mu = 10.5$
$m = 0.02$

with 3 walls:
$x_0 = (0,0), n_0 = (1,0)$
$x_1 = (0.8,0), n_1 = (-1,0)$
$x_2 = (0,0), n_2 = (0,1)$

References

COROS, S. Particle-based fluids. `http://www.cs.cmu.edu/~scoros/cs15467-s16/lectures/11-fluids2.pdf`.

GINGOLD, R. A., AND MONAGHAN, J. J. 1977. Smoothed particle hydrodynamics. *Monthly Notices of the Royal Astronomical Society*.

MACKLIN, M., AND MÜLLER, M. 2013. Position based fluids. *SIGGRAPH*.

MACKLIN, M., MÜLLER, M., CHENTANEZ, N., AND KIM, T. Y. 2014. Unified particle physics for real-time applications. *SIGGRAPH*.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *Vis. Comun. Image Represent.*.

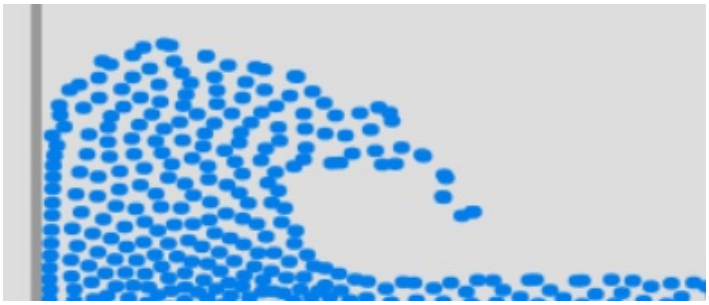

Figure 3: *Screen shot of a demo showing a wave*

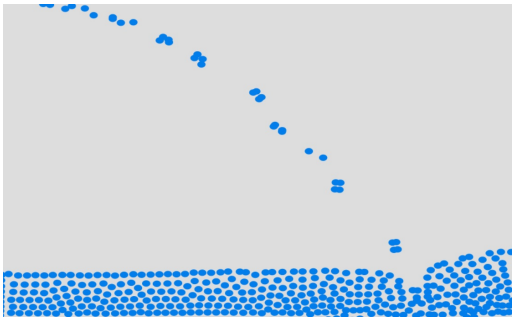

Figure 4: *Screen shot of a demo showing a spout of water hitting a pool of water*

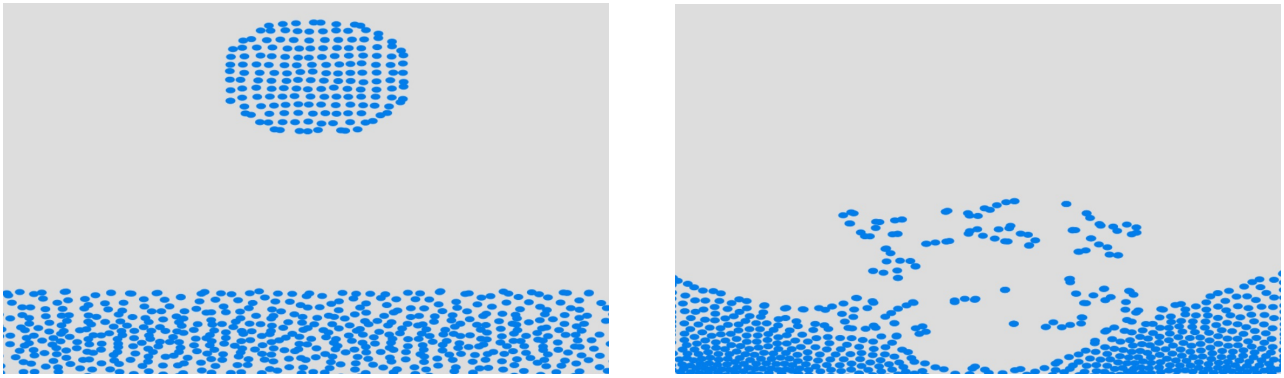

Figure 5: *Screen shots of a demo showing a droplet of water splashing into a pool*

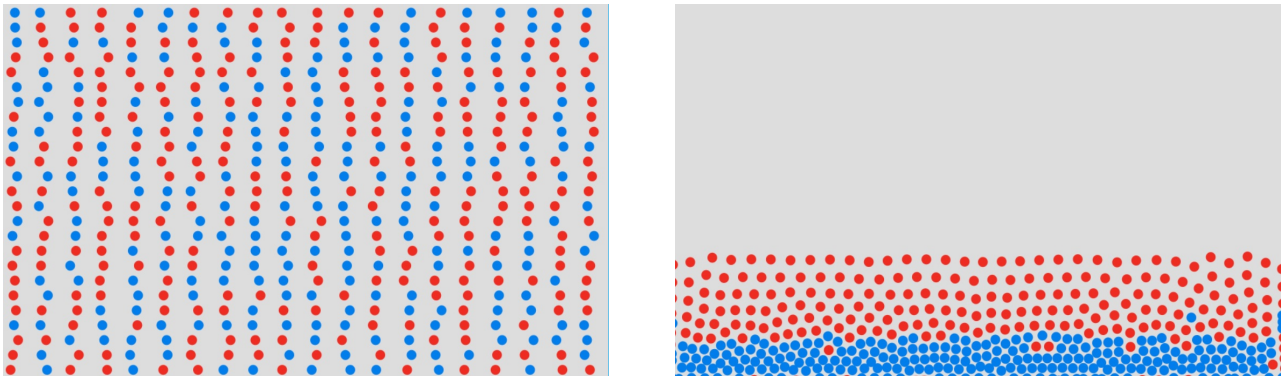

Figure 6: *Screen shots of a demo showing how particles of different density naturally separate,* $\rho_{red} \approx 0.5\rho_{blue}$

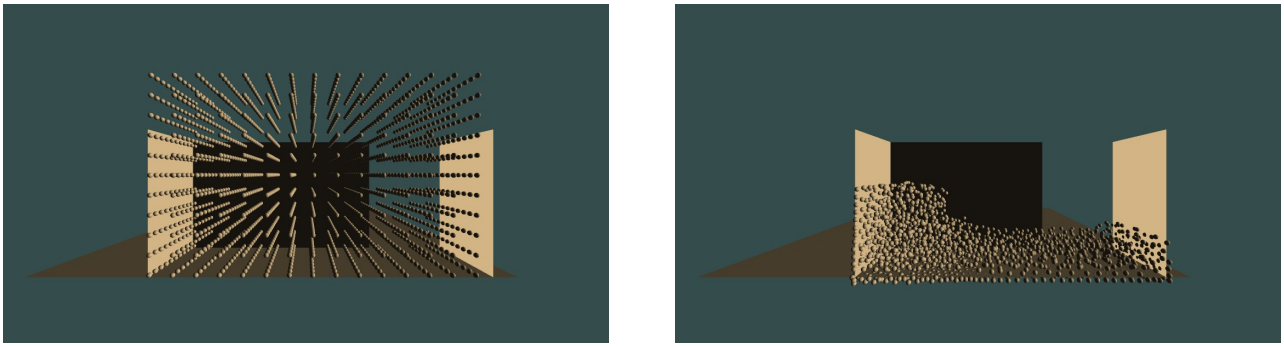

Figure 7: *Screen shots of a 3D demo*