Fluid Dynamics using Smoothed Particle Hydrodynamics

David Brown

October 28, 2013

1 Introduction

Computational Fluid Dynamics (CFD) is the field of simulating fluids in a realistic and visually appealing manner. CFD is a large subfield of computer animation and is still an active research problem even though fluid simulations have been around for over 40 years [3]. New methods and improvements on old methods are constantly being developed with many new papers on the subject being produced each year.

Part of the reason CFD is such an active field is because of the vast applications of fluid dynamics. Fluids are everywhere and nearly any simulation or animation of some real world scene or phenomena will have to deal with them somehow. Another reason is that fluids are extremely difficult to simulate and animate. It is still not known to this day whether the Navier-Stokes equation in three dimensions always has a solution, and if there is always a solution if that solution is smooth [2]. Thus solving the equation can be very time consuming especially for large bodies of water.

The field of Computational fluid dynamics can further be divided into simulation and visualization. Research into simulation methods tends to attempt to improve the accuracy, speed and robustness of the simulation, while visualization research usually tries to improve the visual quality and speed of fluid visualization. In this project I focused on the simulation aspect of the problem and used a very simple visualization. After all, one must have a way to simulate fluids before one can represent the fluid visually.

2 Motivation

The simulation and visualization of fluids has very large variety of applications. Fluid simulations are necessary to simulate air flow and turbulence for use in aerodynamic designs. Meteorology uses CFD to help model weather patterns. Oceanography uses CFD to simulate ocean currents and other phenomena. Astrophysicists use CFD (specifically Smoothed-particle Hydrodynamics) to simulate galaxy formation, star formation, and supernovas.

The simulation and realistic visualization of fluids is also very important for computer animation. Fluids are present nearly everywhere in our world thus many animations have fluids in them. This combined with the issue that fluids are extremely difficult and time consuming to animate convincingly by hand makes CFD a valuable topic in computer animation.

3 CFD Methods

There are many methods for CFD but all of them in some way or another depend on the Navier-Stokes equation:

$$\rho\left(\frac{\delta \mathbf{v}}{\delta t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = -\nabla p + \nabla\cdot\mathbb{T} + \mathbf{f} \qquad (1)$$

$$\nabla\cdot\mathbf{v} = 0 \qquad (2)$$

The Navier-Stokes equation describes the motion of a fluid as a vector field in a grid. Equation (1) is basically conservation of momentum for a fluid. The left hand side is the product of mass densities and acceleration densities, and the right hand side is the sum of force densities on the fluid. The most common methods for CFD are Eulerian grid-based methods, Smoothed-Particle Hydrodynamics (SPH) methods and Lattice Boltzmann methods. The first two of these methods are directly based on the Navier-Stokes equations for fluid motion, while the Lattice Boltzmann method is based off the Boltzmann equation (however the Navier-Stokes equation can be derived from the Lattice Boltzmann equation).

3.1 Eulerian Methods

The Eulerian method is a direct application of (1) and (2). It divides the space where the fluid is simulated into cells, each cell containing some volume of fluid. Solving (1) and (2) numerically give the velocity of the fluid in each cell and we can then update the volume of fluid in each cell as well as calculate other physical quantities for the fluid.

The Eulerian method can produce some very accurate fluid simulations it has some limitations. For one it is limited by the grid used, so even if we are simulating a small amount of fluid, if we want to simulate over a large area we will still need a grid that covers the entire area, and in three dimensions even a coarse grid uses a very large amount of memory. However there are adaptive data structures that can enlarge the grid as necessary and use smaller amounts of memory [4]. Even with these lower memory data structures, numerically solving (1) and (2) is very time consuming so Eulerian methods still cannot be computed in real time.

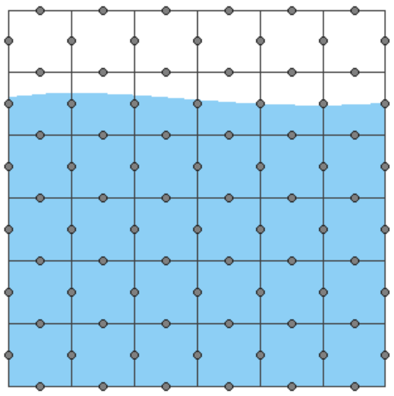

Figure 1: Eulerian method fluid representation. The dots represent discrete points in the velocity field.

4 SPH

Smoothed Particle Hydrodynamics is a Lagrangian particle based method and was originally developed in the field of computational astrophysics for simulating astrophysical phenomena [6]. It is based on the theory of integral interpolants. The value of a quantity A at a given particle is approximated by

$$A_S(\mathbf{r}) = \sum_j A_j V_j W(\mathbf{r} - \mathbf{r}_j, h) \qquad (3)$$

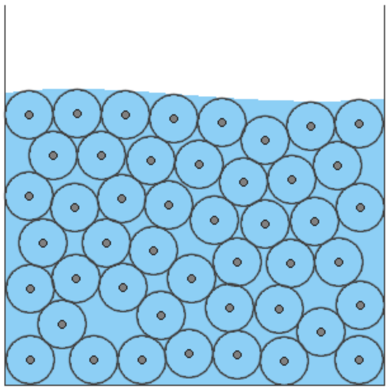

Figure 2: Lagrangian method fluid representation. The dots represent particles and the circles represent the volume of the particle

Where A_j is the quantity at particle j, V_j is the volume attributed to particle j, r is the location of the particle, W is a smoothing kernel, and h is the width of the kernel. We also substitute volume using the relation

$$V = \frac{m}{\rho} \tag{4}$$

to get

$$A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) \tag{5}$$

where m is the mass and ρ is the mass density. We can perform standard analytic differentiation with respect to $\mathbf{r}$ on (3) to get its gradient and Laplacian which are needed to calculate the fluid forces

$$\nabla A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h) \tag{6}$$

$$\nabla^2 A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r} - \mathbf{r}_j, h) \tag{7}$$

4.1 Smoothing Kernels

The choice of smoothing kernels is very important in SPH. Any kernel we use must satisfy the following [7]:

$$\int W(\mathbf{r}, h) d\mathbf{r} = 1 \tag{8}$$

$$\lim_{h \to \infty} W(\mathbf{r}, h) = \delta(\mathbf{r}) \tag{9}$$

$$W(\mathbf{r}, h) = W(-\mathbf{r}, h) \tag{10}$$

$$W(\mathbf{r}, h) \geq 0 \tag{11}$$

Where δ is Dirac's delta function.

When coming up with a new SPH equation the general rule is to try a Gaussian kernel first [7].

$$W_{gaussian}(\mathbf{r}, h) = \frac{1}{(2\pi h^2)^{\frac{3}{2}}} e^{-(\|\mathbf{r}\|/2h^2)}, h > 0 \tag{12}$$

However the Gaussian does not have compact support ($\forall \|\mathbf{r}\| \geq h : W(\mathbf{r}, h) = 0$) so I used an approximation of the Gaussian from [5]:

$$W_{default}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - \|\mathbf{r}\|)^3 & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \|\mathbf{r}\| > h, \end{cases} \tag{13}$$

4.2 Forces

The only physical quantity property of the particle that varies with time other than movement related quantities is the particle's mass-density. We compute the mass-density using equation (5) with ρ as the physical quantity and using the default kernel. We then use the mass-density to compute other forces

4.2.1 Pressure

We can compute the pressure using the ideal gas law

$$pV = nRT, \tag{14}$$

where we can then substitute $V = \frac{1}{\rho}$. If we assume a constant temperature and mass we can simplify (14) to

$$p = k\rho, \tag{15}$$

where k is our gas stiffness constant. Also we use a slightly different version of the gradient (6) so our pressure force is symmetric

$$\begin{aligned} \mathbf{f}_i^{pressure} &= -\nabla p(\mathbf{r}_i) & (16) \\ &= -\rho_i \sum_{j \neq i} \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) m_j \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right) & (17) \end{aligned}$$

Our default kernel (13) is also not suitable for the pressure force because as $\|\mathbf{r}\| \to 0, W_{default}(\mathbf{r}, h) \to 0$ which will lead to clustering so we use the spiky kernel in [5] with gradient

$$\nabla W_{pressure}(\mathbf{r}, h) = -\frac{45}{\pi h^6} \frac{\mathbf{r}}{\|\mathbf{r}\|} (h - \|\mathbf{r}\|)^2. \tag{18}$$

4.2.2 Viscosity

The viscosity force would also be asymmetric using (7), but if we use the realization that the viscosity force is only based on the difference in velocity we can use

$$f_j^{viscosity} = \mu \sum_{j \neq i} \left(\mathbf{v}_j - \mathbf{v}_i\right) \frac{m_j}{\rho_j} \nabla^2 W\left(\mathbf{r} - \mathbf{r}_j, h\right) \tag{19}$$

We also use a different kernel as suggested in [5] in which $\nabla^2 W$ is always positive. We do this because we don't want the viscosity to ever increase the velocity of a particle.

$$\nabla^2 W_{viscosity}(\mathbf{r}, h) = \frac{45}{\pi h^6}(h - \|\mathbf{r}\|). \tag{20}$$

4.2.3 External Forces

The only external force I implemented was gravity which is simply

$$f_i^{gravity} = \rho_i \mathbf{g}, \tag{21}$$

where $\mathbf{g}$ is the acceleration due to gravity. Some other external forces that could be added would be buoyancy (which replaces gravity for gasses) and surface tension. Surface tension is only applied to particles on the surface. To find these particles you could average the radius vector to nearby particles and if that average isn't close to 0 the particle must be at or near the surface. This method also gives you an approximation of the normal of the surface which is also needed to calculate the surface tension.

5 Implementation

My implementation of SPH simulates liquid water being poured into a few different containers. The system first initializes the constant values and creates any objects the particles will collide with and then sets the particle's initial positions and velocities. Then each simulation step first the mass-density is calculated for each particle. Next the mass-density is used to calculate the pressure. Next the internal pressure and viscosity forces are calculated and then any external forces are applied. Then the particles new positions are integrated from their current position, velocity, and computed force. Next any collisions resulting from the integration are handled. Finally the particles are rendered in some fashion.

5.1 Integrator

I used the Leap frog integrator for my time integration scheme. The leap frog integrator is an implicit Euler method where you use calculate the velocities at times halfway in between each time step and then use the velocity from half a time step in the future to calculate the position at the current time step. This leads to a more accurate integration that is still quite fast.

5.2 Neighborhood Search

To find the particles within a given particle's support radius I used a hash space. A particle is hashed using the equation

$$hash(\hat{\mathbf{r}}) = (\hat{\mathbf{r}}_x p_1 \oplus \hat{\mathbf{r}}_y p_2 \oplus \hat{\mathbf{r}}_z p_3) \bmod n_H, \quad (22)$$

where n_H is the size of the hash table, $p1, p2$ and $p3$ are very large prime numbers and

$$\hat{\mathbf{r}}(\mathbf{r}) = (\lfloor \mathbf{r}_x/h \rfloor, \lfloor \mathbf{r}_y/h \rfloor, \lfloor \mathbf{r}_z/h \rfloor)^{\mathrm{T}} \quad (23)$$

where h is the support radius.

5.3 Collision Handling

I only implemented simple plane/point collision handling for my simulation. However it is possible to construct many situations using simply planes. My collision responses were also very simple. Collisions with the plane simply resulted in a semi-elastic reflection of the particles velocity off the normal of the plane. The elasticity of the collisions was given by c_R where $c_R = 0$ was a completely inelastic collision and $c_R = 1$ was a completely elastic collision. I chose $c_R = .5$ for my examples.

5.4 Paramaters

There are several parameters that must be tuned to get a good looking simulation out of this system. I tried to base as many of my parameters as possible on real world values. First mass, volume, and number of particles are set based on the amount and volume of fluid you want to simulate and how detailed you want it to be. In my simulations I used between 25 and 45 liters of water and 10,000 particles. Since I was simulating water I used a density of 1,000 $\frac{kg}{m^3}$ to calculate the mass. The next parameter is the kernel support radius. The support radius is more intuitively selected by how many particles you want to be in the support radius on average. From this plus the number of particles and volume the system can compute a good support radius.

Two values that are also closely related are the timestep and the stiffness coefficient k. The stiffness coefficient is directly related to how incompressible the simulation will be. So for liquids you would want as high a stiffness coefficient as possible. However if the stiffness coefficient is too high they simulation will become unstable. This can be alleviated by lowering the timestep, which will then make the system run slower. So for liquids it is best to choose a stiffness coefficient that is as large as possible without making the system unstable for you chosen timestep. I used a stiffness coefficient of 500 and a timesetp of 3 ms.

The final paramater that must be choosen is the viscosity coefficient μ. This can be made up for some imaginary liquid or chosen from actual measured quantities. Since I was simulating water I used water's viscosity which is 3.5 $Pa \cdot s$ at room temperature.

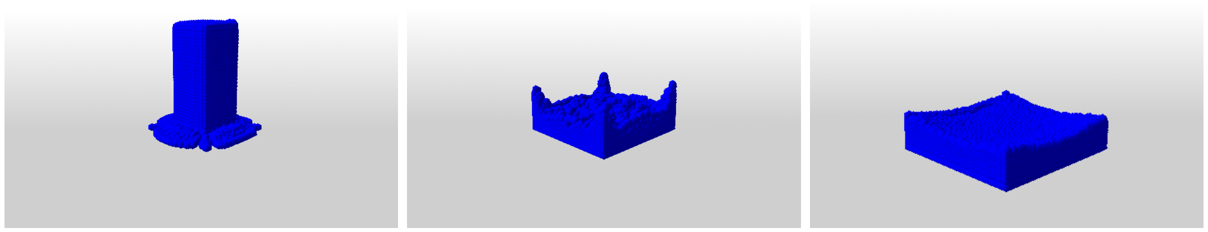

Figure 3: Dam break

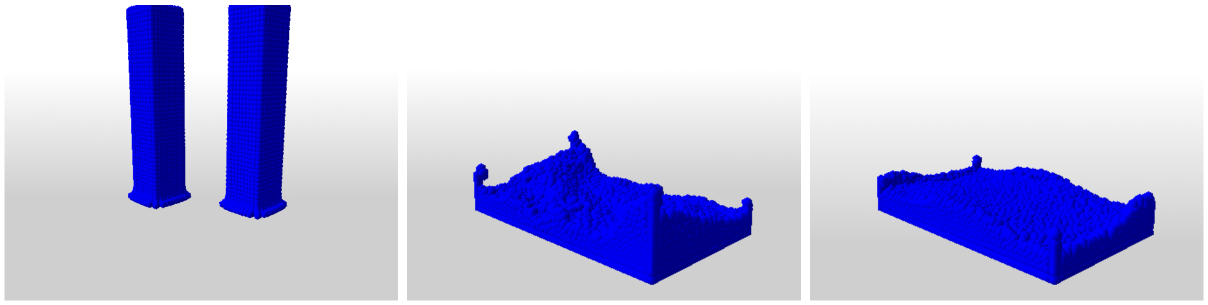

Figure 4: Double Dam break

6 Limitations and Problems

There are several limitations of my system. Foremost among them was it's speed. The system ran incredibly slowly. Simulating 6 seconds of fluid using 10,000 particles and a timestep of 3 *ms* took over 8 hours to finish, even with my speed improvement measures of using a hash space. I did not have the time to look into and fix this issue but I expect it is due to some poor implementation detail like copying a large vector between functions or similar.

Another limitation which is inherent to this SPH implementation is that the liquids simulated in it are slightly incompressible. SPH liquids are not intended to be extremely accurate simulations so this is not a serious problem but it is a limitation of SPH.

A third limitation is the collisions are very naive. In one of my examples I tried to simulate a wave machine. The wave machine had one wall that moved forward and back with a sin curve. However the collision system assumed that the planes were stationary so no momentum was transfered to the particles when they were pushed by the moving wall, it was the same as if they had hit a stationary wall. This led to some strange artifacts where the particles hit the moving wall of the wave machine. However the wave machine did work reasonably well.

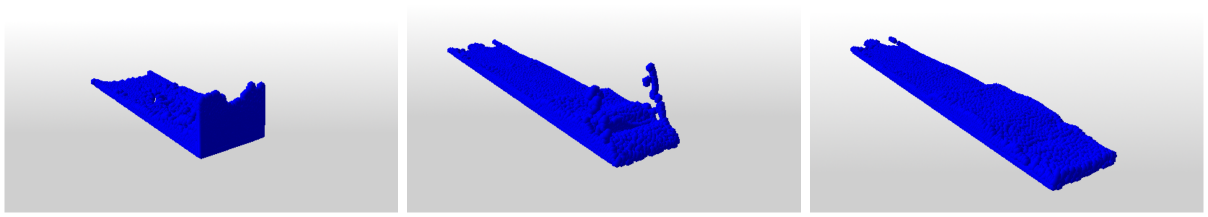

Figure 5: Wave Machine

7 Results

I made three situations to test my water simulation. The first was simply dropping a column of water into a square container and letting the water settle. The water behaved reasonably like water, although the splashes it made were difficult to distinguish because of the low number of particles. The second was dropping two columns of water into opposite corners of a rectangular container. The idea was to have the two waves of water meet in the middle and splash up against each other. This sort of happened but the splash together in the middle did not go as high as I was expecting. Finally my third example was a simple wave machine as discussed earlier. The waves did not crash as I was hoping they would. This was probably a combination of the low volume of liquid and low number of particles. However there were visible waves that propagated through the liquid as you would expect.

8 Future Work

There is a great deal of future work that could be done on this project. First thing I would do would be to track down the performance issues so it runs in a reasonable amount of time. Another thing to add would be surface reconstruction so the liquid would look more like a liquid. Improvements could also be made to the quality of the simulation. For starters adding a surface tension force would be a nice improvement and perhaps make splashes in the system look better. Also the improved pressure force calculation from [1] could be added to further improve the quality of the simulation.

Interaction with other objects would also be a useful addition. First thing would be to make collisions with moving objects transfer momentum to the particles. Rigid bodies could also be added to the simulation that could float, sink, and otherwise interact with the fluids. Some user interaction with the fluid would be nice also, such as applying forces to make waves or splashes, or moving the container of the fluid around.

9 Conclusion

This system manages to perform a reasonable simulation and visualization of a liquid. There are issues with the appearance of splashes, but waves and other effects in the fluid look convincing. Although the fluid is compressible, with a high enough stiffness constant it is not very noticeable. The algorithm is a pretty simple one to both understand and implement and yet it still produces pleasing results. Overall SPH seems like a valuable technique for fast simulation of good looking fluids where accuracy is not a priority.

References

[1] Becker, M., and Teschner, M. Weakly compressible sph for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), SCA '07, Eurographics Association, pp. 209–217.

[2] Fefferman, C. L. Existence and smoothness of the navier-stokes equation. http://www.claymath.org/millennium/Navier-Stokes_Equations/navierstokes.pdf.

[3] Harlow, F. H., and Welch, J. E. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids 8*, 12 (1965), 2182–2189.

[4] Houston, B., Nielsen, M. B., Batty, C., Nilsson, O., and Museth, K. Hierarchical rle level set: A compact and versatile deformable surface representation. *ACM Trans. Graph. 25* (January 2006), 151–175.

[5] Kelager, M. Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics. January 2006.

[6] Lucy, L. B. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal 82* (December 1977), 1013–1024.

[7] Monaghan, J. J. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30* (1992), 543–574.