Physics 599 – Process Paper

BY

Joshua Davis

BS RTIS, DigiPen Institute of Technology, Winter 2010

[Chapter 1: Fluid Basics – The Navier-Stokes equations 2](#_Toc385357810)

[1.1 Lagrangian and Eulerian Fluids 3](#_Toc385357811)

[1.2 Incompressibilty 5](#_Toc385357812)

[1.3 Dropping Viscosity – The Euler equations 5](#_Toc385357813)

[Chapter 2: Smoothed Particle Hydrodynamics 6](#_Toc385357814)

[2.1 Density 7](#_Toc385357815)

[2.2 Pressure 8](#_Toc385357816)

[2.3 Pressure Force 8](#_Toc385357817)

[2.4 Viscosity 8](#_Toc385357818)

[2.5 External forces 9](#_Toc385357819)

[2.6 Surface Tension 9](#_Toc385357820)

[2.7 Rendering 10](#_Toc385357821)

[2.8 Spatial Partitions 10](#_Toc385357822)

[2.9 Collision Detection and Resolution 11](#_Toc385357823)

[2.10 Integration 12](#_Toc385357824)

[2.11 Summary 13](#_Toc385357825)

[Chapter 3: Eulerian Fluid 14](#_Toc385357826)

[3.1 Space Discretization 14](#_Toc385357827)

[3.2 Advection 16](#_Toc385357828)

[3.3 Incompressibility 17](#_Toc385357829)

[3.4 Extrapolation 20](#_Toc385357830)

[3.5 Surface Simulation 21](#_Toc385357831)

[3.5.1 Marker and Cell 21](#_Toc385357832)

[3.5.2 Level Sets 22](#_Toc385357833)

[3.6 Rendering 22](#_Toc385357834)

[3.7 Summary 22](#_Toc385357835)

[Chapter 4: Conclusion 24](#_Toc385357836)

[References 25](#_Toc385357837)

Chapter 1: Fluid Basics – The Navier-Stokes equations

Physical simulations are typically solved through the momentum equation: . Simulating fluid requires a more complex set of equations known as the Navier-Stokes equations. The Navier-Stokes equations for incompressible fluid are:

|  |  |
| --- | --- |
| (1) |  |
| (2) |  |

where is velocity, is gravity, is density, is pressure, and is the viscosity coefficient. The remainder of this chapter closely follows [Bridson07].

Equation (2) states that the amount of fluid flowing into a region must be the same as the amount of fluid flowing out. This is enforcing the incompressibility of the fluid.

To understand equation (1) imagine that the fluid is a particle system of fluid blobs. Each of these blobs has a mass , a volume , and a velocity . The sum of forces is computed for each particle and then integrated.

First, gravity acts on each particle: . Several other forces are caused by particles interacting with each other. Pressure causes particles to move away from each other and towards empty space, that is from high pressure to low pressure zones. The simplest way to measure the pressure force is with the gradient of the pressure: . To accurately compute the pressure force the integral over the entire volume needs to be computed. This can be approximated by multiplying with the volume: .

The viscosity force resists differences in velocity relative to nearby fluid. How far a quantity is from the average can be measured using the Laplacian operator (also written as ). The integral of this can also be approximated by multiplying with the volume. Along with the viscosity coefficient this yields: .

Combining all this into the basic set of forces acting on a fluid:

|  |  |
| --- | --- |
|  | . |

Taking the limit as the volume goes to zero makes the approximation become exact. Unfortunately the mass and volume drop to zero making every term disappear. This is avoided by dividing all terms by and remembering :

|  |  |
| --- | --- |
|  | . |

Re-arranging:

|  |  |
| --- | --- |
|  | . |

The kinematic viscosity further simplifies this:

|  |  |
| --- | --- |
|  | . |

The acceleration can then be re-written as to give:

|  |  |
| --- | --- |
|  | . |

This big derivative notation is commonly called the material derivative. To further understand this the difference between Lagrangian and Eulerian fluids needs to be explained.

1.1 Lagrangian and Eulerian Fluids

The two primary methodologies of simulating fluid are Lagrangian and Eulerian. In the Lagrangian viewpoint, the fluid is discretized as particles with fixed properties (velocity, density, etc…) that are allowed to move. The Eulerian approach looks at fixed points in space but allows the fluid quantities to vary.

To help understand the difference consider the example of measuring temperature. In the Lagrangian view the temperature is measured on a balloon that is floating with the wind. In the Eulerian view the balloon is fixed and measures the temperature of the air that flows past.

The relation between these two views can be seen through the material derivative. Observe the generic quantity in the fluid. In the Lagrangian view each particle simply has a value for . In the Eulerian view the value of can be sampled using the function for a specific time and position. The rate of change over time of is:

|  |  |
| --- | --- |
|  |  |

How the quantity changes with the vector field is called advection and is set to zero:

|  |  |
| --- | --- |
|  | . |

In the Lagrangian view this can be thought of as the particle moving around without its value of changing.

A 1D example is useful to explain this [Bridson07]. Let temperature on the 1D plane be represented by: , that is the temperature is zero at the origin and increases to the right. Now let there be a constant wind blowing at speed . Setting the advection to be zero, that is the air is moving but not changing temperature, the material derivative where is substituted for is:

|  |  |
| --- | --- |
|  | . |

This shows that with a wind speed of 1 the air temperature decreases by per second at a fixed location.

1.2 Incompressibilty

Pressure is used to enforce the incompressibility of the fluid. To relate pressure to equation (2) the divergence of equation (1) is taken:

|  |  |
| --- | --- |
|  | . |

Changing the order of differentiation, becomes . Equation (2) states that this is zero. This gives:

|  |  |
| --- | --- |
|  | . |

This equation is not actually solved for in the simulation, however it is useful to see as similar steps are taken in Chapter 3 when enforcing incompressibility.

1.3 Dropping Viscosity – The Euler equations

Viscosity is commonly ignored in water simulations. The Navier-Stokes equations without viscosity are called the Euler equations:

|  |  |
| --- | --- |
|  |  |
|  | . |

Chapter 2: Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) was first created by astrophysicists to solve problems of compressible fluid flow. At its heart, SPH is an interpolation technique that approximates values and derivatives of a continuous field from sample points. For a more in depth discussion of SPH see [Kelager06].

To evaluate the integral interpolant of any quantity over all space :

|  |  |
| --- | --- |
|  |  |

where is any point in the domain and is a smoothing kernel function of size . How does one implement this integral? Monte Carlo integration is a method to approximate an integral by sampling over the domain at a finite set of points:

|  |  |
| --- | --- |
|  | . |

Similar to this, integrals in SPH are approximated as:

|  |  |
| --- | --- |
|  |  |

where is the volume of particle . Substituting the definition of volume :

|  |  |
| --- | --- |
| (3) | . |

The kernel function is used to weight values of neighboring particles. Similarly to Monte Carlo integration, the Dirac delta function could be used as a start:

|  |  |
| --- | --- |
|  | . |

The Dirac delta function doesn’t interpolate values of nearby neighbors. The first golden rule of SPH is to assume the kernel should be a Gaussian. Gaussians are useful in SPH because they are smooth normalized functions that weight particle quantities based upon distance. There are two problems that prevent Gaussians from being used in SPH. Gaussians are more expensive to compute than other functions that behave similarly. Gaussians have an infinite kernel size while kernels for SPH need to smoothly evaluate to zero at the kernel size . Typically a high order polynomial function is used. Later sections will describe the different kernels used.

It is important to understand how derivatives work with the interpolation function, most notably the gradient and Laplacian. Since , , and are not functions of position:

|  |  |
| --- | --- |
| (4) | , |
| (5) | . |

Sometimes these definitions do not work well due to a lack of symmetry. For instance, when evaluating only two particles in the gradient, particle will only use the value and particle will only use . To uphold Newton’s 3rd law a symmetric version is sometimes used:

|  |  |
| --- | --- |
| (6) | , |
| (7) | . |

SPH can now be solved using the Navier-Stokes equation:

|  |  |
| --- | --- |
|  | . |

Note that forces used in SPH are actually force-densities. Normally force is written as , for the fluid force-densities they are instead written as . Also note that equation (2) is dropped as the particles have a constant volume and conserve fluid automatically.

2.1 Density

Before evaluating the Navier-Stokes equation, the density at each particle is needed. Equation (3) is used to evaluate the density:

|  |  |
| --- | --- |
|  |  |

which simplifies to:

|  |  |
| --- | --- |
| (8) | . |

The density kernel used is a 6th degree polynomial [Muller03]:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |

2.2 Pressure

Pressure is most simply calculated using the ideal gas law: . Defining as the volume per unit mass, can be substituted as to get . Re-writing the right hand side as the constant : . This equation only allows repulsion forces. Typically,

|  |  |
| --- | --- |
| (9) |  |

is used, where is the rest (or target) density of the fluid, to allow attractive forces as well. Note that acts like a spring stiffness coefficient.

2.3 Pressure Force

Pressure force-density is calculated from the pressure at each particle:

|  |  |
| --- | --- |
|  | . |

Equation (4) is not symmetric so Newton’s 3rd law is not preserved. Instead equation (6) is used:

|  |  |
| --- | --- |
| (10) | . |

Pressure force-density uses the spiky kernel [Muller03]:

|  |  |
| --- | --- |
|  | . |

To prevent a zero division:

|  |  |
| --- | --- |
|  | . |

is used.

2.4 Viscosity

Viscosity force-density is represented as:

|  |  |
| --- | --- |
|  |  |

where is the viscosity coefficient. To preserve symmetry, equation (7) is used instead of equation (5):

|  |  |
| --- | --- |
| (11) | . |

The viscosity kernel is defined as [Muller03]:

|  |  |
| --- | --- |
|  | . |

2.5 External forces

There are two simple external force-densities: gravity and user forces. Gravity is simply defined as:

|  |  |
| --- | --- |
|  | . |

User forces are any force-densities that the user applies. This is just a sum of force-densities:

|  |  |
| --- | --- |
|  | . |

User forces can be anything, ranging from mouse manipulation to animation.

2.6 Surface Tension

Surface tension makes nearby particles stick together and tries to minimize the surface area of the fluid volume. To compute surface tension force-densities, a color field is used. The color field is a field where in equation (3) is 1 at particle locations and 0 everywhere else:

|  |  |
| --- | --- |
|  | . |

The smoothed surface normal is computed from the gradient of the color field:

|  |  |
| --- | --- |
| (12) | . |

Finally, the surface tension force-density is:

|  |  |
| --- | --- |
| (13) |  |

where is the surface tension coefficient and the laplacian is computed using equation (5). The length of the normal should first be checked to prevent a division by zero. In practice, the surface tension force is only applied when where is some threshold value relating to particle concentration.

Surface tension uses both the gradient and the Laplacian of the 6th degree polynomial kernel function [Muller03]:

|  |  |
| --- | --- |
|  | , |
|  | . |

2.7 Rendering

SPH can be rendered using many different techniques, e.g. marching cubes. For simplicity (especially when debugging), each particle in this project is rendered as a sphere. To aid visualization, each particle’s size is changed based upon its density. Knowing that and the radius can be computed as:

|  |  |
| --- | --- |
| (14) | . |

2.8 Spatial Partitions

As described above, SPH is an problem. Every particle must check against every other particle. However, the kernel function returns 0 beyond the support radius . All calculations beyond this range are unnecessary. An obvious optimization is to ignore any particles that are too far apart, that is skip all calculations where . This is still and a large amount of time is spent checking the distance between two particles. Instead a spatial hash is used to pre-filter out objects resulting in complexity.

The spatial hash sorts particles into buckets based upon their positions. When checking for nearby particles, only nearby buckets have to be checked. To do this, the spatial hash needs to have cells of size or larger. Only neighboring cells need to be checked because any further away cell will have a contribution of zero. Instead of a simulation of fixed size or having to deal with a growing grid structure, a hashing function is used to allow arbitrarily sized simulations. The hash function described by [Teschner03] is used:

|  |  |
| --- | --- |
| (15) |  |

where , , , and. The table size is set to where is a function that returns the next prime number greater than the value passed in and is the number of particles in the scene.

Particles are inserted into an array at the index provided by equation (15). To find possible nearby particles, each neighboring cell position is hashed to find their index in the array.

2.9 Collision Detection and Resolution

Two approaches to detecting collision were attempted: a continuous and a discrete method. A continuous method is used to prevent particles from entering inside of solid objects. A discrete method moves particles and then removes any penetration created afterwards.

A continuous method can be achieved with a raycast. By keeping track of where the particle was and where it is going to end up a raycast can be used to find the first time the particle will collide with a solid boundary. For simplicity, the particle is moved only up until the first contact and the remaining frame time is ignored.

The discrete method performs a point-inside-of-shape-test and calculates the penetration depth and normal. The particle is pushed out in the direction of the normal by the penetration depth to remove overlap. If large enough time steps are allowed (or particles travel fast enough), the projected position can be inside of another solid boundary. For simplicity, this edge case is ignored.

Continuous and discrete methods have advantages and disadvantages. Continuous detection handles fast movement better than discrete. Even with continuous it is not always possible to prevent overlap. Errors need to be fixed in a discrete manner. This is most notable with dynamic objects. In this project the fluid simulation is interleaved with a rigid body simulation. Rigid bodies are updated and then particles are updated. A rigid body can fall on top of a particle such that the particle starts the fluid simulation in an invalid scenario.

To resolve collisions, the general impulse equation is used with simplifications for point masses. Assuming object 1 is the rigid body and object 2 is the particle:

|  |  |
| --- | --- |
|  |  |
|  |
|  |

where is the inverse mass, and is the inverse inertia tensor. Then

|  |  |
| --- | --- |
|  |  |

where is the coefficient of restitution, is the vector from the center of mass of object 1 to the point of contact and . When the rigid body is static then its mass terms drop to zero:

|  |  |
| --- | --- |
|  |  |

where .

2.10 Integration

Leapfrog integration is used to advance particles forward in time. Leapfrog is defined as:

|  |  |
| --- | --- |
| (16) |  |
| (17) | . |

Leapfrog looks just like a semi-implicit Euler integration scheme. For the majority of the simulation, the velocity being offset by half a timestep can be ignored. This only matters if the velocity at timestep is needed, which occurs in viscosity and collision resolution. The velocity at timestep can be estimated using a midpoint approximation:

|  |  |  |
| --- | --- | --- |
|  |  | . |

2.11 Summary

SPH is relatively simple to implement. The biggest challenge is in making it performant. The biggest savings to be had is to reduce the number of particles being checked with a spatial-hash broadphase. SPH is also a very thread-able algorithm.

One primary drawback to SPH is that is suffers from the stiff problem inherent to all harmonic oscillators. To achieve incompressible fluid the stiffness coefficient needs to be set higher. This produces oscillations that can blow up the simulation if the timestep is not small enough. For this reason, SPH tends to produce better simulations of slime and other viscous materials, as the extra damping due to viscosity improves numerical stability.

Chapter 3: Eulerian Fluid

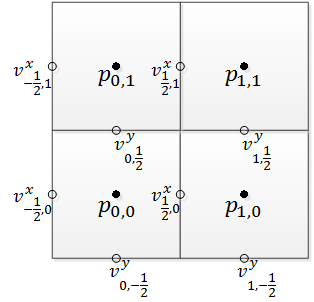
The Eulerian approach of fluid simulation discretizes space, typically into cells, and measures fluid quantities flowing through fixed points in space. Eulerian fluids commonly ignore viscosity in the Navier-Stokes equations and are broken into three main steps:

|  |  |  |
| --- | --- | --- |
| 1. |  | (advection) |
| 2. |  | (gravity) |
| 3. |  | (pressure/incompressibility). |

This order is important as the advection step requires an incompressible velocity field. Before talking about these equations in detail, the discretization of the grid needs to be discussed. For a more in depth discussion of Eulerian fluids, see [Bridson07].

3.1 Space Discretization

In the Eulerian representation of fluid, space is discretized into grids. Each grid cell contains pressure, velocity and force. A special kind of structure called the MAC (Marker-and-Cell) grid can be used to store this information [Harlow65]. The pressure is stored at the center of the grid while the velocity and force are stored on the faces of the grid as shown in Figure 1.

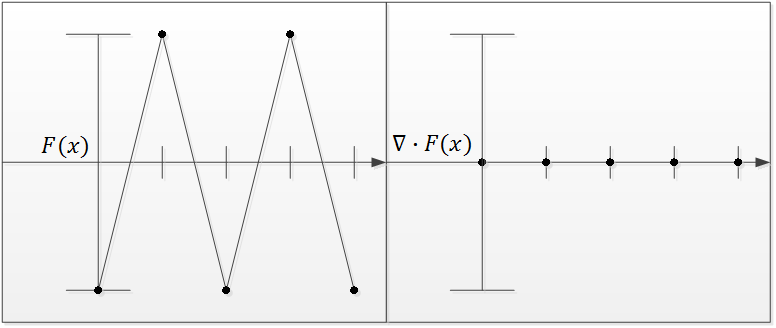


|  |
| --- |
| Figure 1: The pressure is stored at cell centers while the x and y velocity is split on the bottom-left face. |

The velocity of any point within the cell is calculated with a bi-linear (or tri-linear in 3d) interpolation. To understand the necessity of the staggered grid the pressure divergence needs to be examined. There are three main ways to compute the derivative of a value in a cell:

1. Forward differences:
2. Backward differences:
3. Central differences:

Preferable to the forward and backward differences, the central differences is unbiased towards left or right. The value of a property in cell is not used in computing its derivative. In the incompressibility equation , this property of central differences allows functions to evaluate to zero that should not as seen in Figure 2.



|  |
| --- |
| Figure 2: The divergence of this saw-toothed pattern incorrectly evaluates to zero using central differences. |

This is known as having a non-trivial null-space, that is functions which evaluate to zero numerically when they should not. An incorrect null-space will allow other functions to satisfy equation (2) and will not preserve incompressibility. Not preserving incompressibility will cause the fluid to lose volume and can cause large instabilities. The MAC grid solves this issue. Central differences with the velocity defined at the halfway points is: . This is unbiased and doesn’t skip over values which restricts the null-space to the correct set of functions.

3.2 Advection

Advection is the process in which the velocity field is integrated forward in time. Simply put, velocity flows through the field from one cell to another. To avoid instability issues that are addressed in [Bridson07], advection is performed using a semi-Lagrangian approach [Stam99]. In the Lagrangian view it is trivial to integrate the fluid forward in time; the particle is advanced using an integration technique (e.g. Euler or RK2). The same approach is used in the semi-Lagrangian method but with imaginary particles. To avoid instabilities, an implicit integration is performed by tracing backwards in time. The velocity for a cell is sampled and traced backwards in time to find where it originated from. This position will likely not be at a grid position. Velocities need to be interpolated from neighboring cells. The cell’s velocity is then updated with the new velocity. Note that the semi-Lagrangian technique cannot be performed in-place.

The semi-Lagrangian advection is made more complicated by the MAC grid structure. A cell does not contain a velocity that can be traced back from one position. Each component of the velocity must be traced back. To update the x velocity the y velocity needs to be computed at the position of the x velocity. This requires a bilinear interpolation in 2d:

|  |  |
| --- | --- |
|  | . |

This velocity is traced back to a new position where the x velocity is sampled using bilinear interpolation. The same steps are taken for the y velocity: a bilinear interpolation to calculate the x velocity, tracing back to a new position, and another bilinear interpolation to compute the new y velocity. This is one of the trickiest parts of using the MAC grid structure. It is highly recommended to sit down with a piece of paper and trace this out until it is fully understood.

There is still one issue to be addressed in advection; what happens if the advection step traces back to a grid that does not contain fluid? The simple answer is to extrapolate the velocity from the fluid to this position. More details will be covered in Section 3.4.

3.3 Incompressibility

To enforce incompressibility, equation (2) needs to be satisfied. This is enforced by computing the correct pressure forces. Using the Euler equations, the pressure force is:

|  |  |
| --- | --- |
|  | . |

Using forward differences to expand :

|  |  |
| --- | --- |
| (18) | . |

This is used to update velocity based on pressure. It is important to note that this equation can reference pressure outside of the fluid. At the fluid boundary, can be air or solid. Dealing with these boundary conditions is one of the most complicated aspects of Eulerian fluids.

Commonly, air cells’ pressure is assumed to be some small constant. For simplicity zero is used as the density of water is several orders of magnitude larger than the density of air. How exactly a cell is determined to be air or fluid will be explained in chapter 3.5, for now assume this as known.

Solid cells are more complicated. Assuming the solid cell has the velocity , the boundary condition needs to make the cell’s velocity satisfy . This is called a free-slip boundary and keeps the fluid from flowing into or out of the cell but does not affect the velocity perpendicular to the surface. This is enforced by computing the correct pressure inside the cell. This pressure is computed by re-arranging equation (18):

|  |  |
| --- | --- |
| (19) | . |

While it might seem odd that the velocity on these boundaries is not just set to the correct velocity, this shows how values other than velocity are updated at the boundary.

To enforce incompressibility, equation (2) must be satisified (). This can be broken up as . Using the MAC grid with central differences gives:

|  |  |
| --- | --- |
|  | . |

The future time velocities can be substituted for using equation (18):

|  |  |
| --- | --- |
|  |  |

which when simplified gives:

|  |  |
| --- | --- |
| (20) | . |

How does this equation change when there are boundary conditions of air or solid? For example, assume cell is solid and is air. In air the pressure is assumed to be zero, so becomes zero:

|  |  |  |
| --- | --- | --- |
|  |  | . |

The solid cell’s pressure can be substituted with equation (19):

|  |  |
| --- | --- |
|  |  |

which simplifies to:

|  |  |  |
| --- | --- | --- |
|  |  | . |

From this we can see some simple rules. If the cell is in air then drop the pressure term to zero; If the cell is solid then drop the pressure term to zero, decrement the coefficient in front of and replace the velocity term on that border with the solid’s velocity (.

This gives a very sparse LCP to solve for pressure. The LCP solved is (traditionally written as ) where is the matrix of coefficients, is the pressure being solved for, and is a constant coefficient. First note that equation (20) can re-arranged slightly to move a constant multiplication to the right side:

|  |  |
| --- | --- |
|  |  |

where . In matrix form this is:

|  |  |
| --- | --- |
|  | . |

More generically, the LCP is:

|  |  |
| --- | --- |
|  |  |

where all of the in-between terms in the matrix are zero. In the previous example with cell as solid and as air:

|  |  |
| --- | --- |
|  | . |

where . Every row in the matrix corresponds to one cell’s copy of equation (20).

This LCP can be solved with a variety of LCP solvers. The simplest is Gauss-Siedel. After getting the basics working, it is highly recommended to switch to a faster converging method such as Conjugate Gradient. If this step is not solved sufficiently, the fluid will lose volume quickly. In a slower converging algorithm like Gauss-Siedel, incompressibility can be enforced but at the cost of a large number of iterations, e.g. in a 40x40 grid, Gauss-Siedel took over 1000 iterations to solve what took Conjugate Gradient only 50 iterations.

3.4 Extrapolation

In the advection step, velocity outside of the fluid may need to be sampled. The simplest way to sample the velocity to extrapolate the velocity of the fluid out into neighboring cells. A simple extrapolation technique was used on each axis independently:

|  |  |
| --- | --- |
|  | for each axis (x,y,z): |
|  | for each cell: |
|  | if the cell neighbors a fluid cell -> set layer to 0 |
|  | Else -> set layer to -1 |
|  |  |
|  | for i to iterations: |
|  | for each cell ‘C’ that is layer -1: |
|  | compute the average velocity of all cells of layer == iteration |
|  | For all neighbor cells ‘N’ of layer -1: |
|  | set N’s velocity to the average velocity |
|  | set layer of C to i |
|  |  |
|  |  |

Note that each axis is extrapolated independently. After extrapolation, solid cell boundaries need to be set back to the correct velocity.

3.5 Surface Simulation

In order to properly simulation an Eulerian fluid the distinction between air and fluid needs to be made. There are two primary methods of updating the surface: Marker and Cell, and level sets.

3.5.1 Marker and Cell

The simplest way to update the surface is with the Marker and Cell method. Marker particles are inserted into the grid to follow the velocity field. These particles are updated with an integration technique such as Euler or RK2. RK2 is recommended as it preserves spiral motions. The marker particles are not part of the advection or incompressibility, they are used to determine where there is fluid. If a cell contains a marker particle then it contains fluid, if there are no marker particles then the cell is air.

With marker particles it is simple to implement sources and sinks. Sources emit particles over a volume at fixed timesteps. Sinks destroy any particles that enter them. The major drawback of marker particles is that they can congregate and get stuck in corners and edges. There is no repulsion forces between marker particles to keep them from grouping.

3.5.2 Level Sets

A level set is a scalar function defined at the center of each grid cell. The surface of the fluid is defined to be where the level set is zero. Different functions can be used for level sets but typically the signed distance function is used where positive values are outside the fluid and negative values are inside. To update the level set, the signed distance values are advected similar to any other quantity at the cell center.

As with marker particles, level sets are used to denote where there is fluid and where there is air. Unlike marker particles, level sets denote more than if a cell has fluid or not; the level set defines where the exact surface of the fluid is. Unfortunately, level sets tend to lose fluid volume quickly without extra care in the advection step. This can be mitigated using a particle level set method [Enright04].

3.6 Rendering

The simplest way to render a grid-based fluid is by changing a cell’s color based upon its contents: fluid, air, or solid. When using the Marker and Cell method it can also be beneficial to debug draw the marker particles to show inner motion of the fluid.

A level set can be rendered by drawing triangles at the zero level set. The easiest way to render this surface is with the marching cubes algorithm [MarchingCubes14]. Marker particles can also be rendered with marching cubes by computing a signed distance.

3.7 Summary

Eulerian grid simulations work well for simulating incompressible fluid such as water. The grids also computationally scale better than SPH. Grid simulations suffer from needing to have a grid everywhere in order to keep track of the fluid there. To avoid consuming enormous amounts of memory, spatial hashes can be used to only store cells that are solid, fluid, or air bordering fluid.

Another issue with Eulerian grids is handling non-grid-aligned boundary conditions. The pressure computations need to change depending on how much of the cell is air, fluid, or solid. Without non-grid-aligned boundaries, slopes can only be represented as staircases which will cause fluid to pool.

Chapter 4: Conclusion

Real-time fluid simulation is still in its infancy. The two primary techniques used are too expensive for large scale use in games and interactive simulations. Eulerian fluid performance scales better than SPH but is significantly more complicated to implement. SPH is massively parallelizable which seems to be where the future of processing power is going. SPH works very efficiently on a GPU, unfortunately most graphics systems don’t have enough GPU cycles to spare. New techniques for simulating fluid will need to combine the parallelization of SPH with the incompressibility of Eulerian grids.

References

[Bridson07] R. Bridson and M. Müller-Fischer. Fluid Simulation. SIGGRAPH Course notes. 2007.

[Enright04] D. Enright, F. Losasso, and R. Fedkiw. A Fast and Accurate Semi-Lagrangian Particle Level Set Method. Computers and Structures 83, 479-490. 2005.

[Harlow65] F. Harlow and J. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. Phys. Fluids, 8:2182–2189, 1965.

[Kelager06] M. Kelagar. Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics.

[MarchingCubes14] <http://en.wikipedia.org/wiki/Marching_cubes>. Accessed March 30, 2014.

[Muller03] M. Müller, D. Charypar, and M. Gross. “Particle-Based Fluid Simulation for Interactive Applications”. Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation, pp. 154-159, 2003.

[Stam99] J. Stam. Stable Fluids. In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series*, August 1999, 121-128.

[Teschner03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranerts, and M. Gross. “Optimized Spatial Hashing for CollisionDetection of Deformable Objects”. In proceedings of Vision, Modeling, Visualization, pp. 47-54, November 19-21, 2003.