Real-Time Fluid Simulation using Control Point Based SPH

by

Eoin O'Grady, B.E. (NUI Galway)

Thesis

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Msc. Interactive Entertainment Technology

University of Dublin, Trinity College

09 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Eoin O'Grady

September 11, 2008

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Eoin O'Grady

September 11, 2008

Acknowledgments

I'd like to thank my dissertation supervisor John Dingliana for all of his help and guidance. I'd also like to thank all of classmates in IET for getting me through the late nights and hard times.

Eoin O'Grady

University of Dublin, Trinity College
09 2008

Real-Time Fluid Simulation using Control Point Based SPH

Eoin O'Grady, M.Sc.
University of Dublin, Trinity College, 2008

Supervisor: John Dingliana

Smoothed Particle Hydrodynamics has become the front-runner of methodologies used to simulate fluids in real-time due mainly to it's flexibility and robustness. "For the mathematician, the particles are just interpolation points from which properties of the fluid can be calculated. For the physicist, the SPH particles are material particles which can be treated like any other particle system." [1]. For my dissertation I propose a method that uses control points spaced evenly in three dimensional space, that form a grid, at which force densities are partially resolved using the SPH methodology and then interpolate those force densities for the individual particles and then resolve their accelerations. I also proposed an algorithm for rendering point based fluid simulations in screen space using a depth-map texture created on the GPU and pixel shader operations that render a smoothed depth map and a normal map from which the screen space silhouettes of a fluid-air interface can be rendered. My rendering algorithm is

based on the work done by Matthias Muller on screen space meshes [2].

Contents

List of Figures

Chapter 1

Introduction

Smoothed Particle Hydrodynamics' strength regarding fluid simulation lies in it's simplicity. It is a Lagrangian particle-based method that interpolates the forces of pressure viscosity and in some implementations surface tension [3] for each particle in accordance with the Navier Stokes Equation [4] . It does this by sampling from each particle within a range of h (known as the smoothing length of the kernel) in order to resolve density and force densities. Comparing SPH to alternative grid-based Eulerian methods [5] in terms of physics the main difference is that the particles in SPH represent physical "parts" of the fluid in essence clusters of fluid molecules. In the grid-based method the particles are used only to track the surface of the fluid. Mathematically SPH is easier to solve avoiding the problems of energy dissipation, mass conservation and solving the discrete poisson equation (see Stam's stable fluids [6]). The weakness of SPH is that the majority of the particles involved in the simulation don't make up the surface (the liquid-air interaction) of the fluid body. Although grid-based methods don't have this problem, computation is scaled only by the resolution of the grid involved not the number of particles, i.e the accuracy of the simulation depends solely on the resolution of the grid. This means that a coarsely defined grid will produce inferior splashing or spray. In contrast for SPH a lot of computation is used on areas which aren't of interest to the user of a real-time application because they don't contribute in a tangible manner to the animation. Efforts have been made to address this problem by having multiple resolution particle sizes and smoothing lengths [7] and by combining SPH with the MAC grid method in areas of higher density i.e. within the body of the

fluid [8]. These attempts to produce a more efficient fluid simulation however have a certain overhead associated with them that in effect treats the symptom but not the problem. For my contribution I proposed using a particle system with a grid of control points spaced at intervals in 3-d space at which the pressure and viscosity forces due to SPH could be resolved I then attempted to estimate using interpolation the force that each particle in the system would experience, using various techniques. The goal was that this would be a much more efficient SPH-based methodology albeit less accurate. In order to explain and understand fluid simulation techniques the navier-stokes equations need to be understood.

1.1 The Navier-Stokes Equations

In 1822 Claude Navier and in 1845 George Stoke formulated the famous Navier-Stokes Equations that describe the force vector fields of fluids in terms of their density pressure and viscosity. Two additional equations, the continuity equation describing mass conservation and the state equation describing energy conservation, are needed in computational fluid dynamics in order to solve for these force vector fields at a point in space. These Equations are continuous and hold for all points in space. Most methods for simulating fluids work by solving discrete values for these equations. Generally this involves selecting a finite number of points or areas in space and time at which a numerical solution to the equations is computed. Grid-based methods are said to be Eulerian in nature because they involve resolving these equations for areas in space which never move. SPH is considered Lagrangian because force is resolved for points in space that move with the particles in the simulation. My own method could be considered a combination of both as the equations are partially resolved at fixed points in space corresponding to the grid and fully resolved for each of the particles which are moving through space. The compressible form of the equation itself is shown in figure 1.1.

Where t is time, $\vec{u}$ is the velocity, ρ is density, μ is the viscosity coefficient, p is pressure, ∇ is the del operator and $\vec{g}$ is gravity. Two other equations are needed to satisfy certain fluid conditions. One is to ensure conservation of mass 1.2.

And the other is the equation of state that relates pressure to density by the gas constant (in practice for fluid simulation A user defined constant) 1.3.

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} + \frac{\mu}{\rho} \nabla \cdot (\nabla \vec{u}) - \frac{1}{\rho} \nabla p + \vec{g}$$

Figure 1.1: The compressible Navier-Stokes equation.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0$$

Figure 1.2: Conservation equation preserves mass in a compressible fluid.

$$p = f(\rho)$$

Figure 1.3: The equation of state

In most fluid simulations however a simplified form of the equation is used to resolve discrete forces for incompressible fluids this is achieved by assuming that the density field is constant. This means that the Continuity equation becomes simplified 1.4.

$$\nabla \cdot \vec{u} = 0$$

Figure 1.4: The simplified conservation equation to preserve mass in an incompressible fluid

And The Navier-Stokes equation itself can be simplified and rearranged into a much more easy to work with form 1.5.

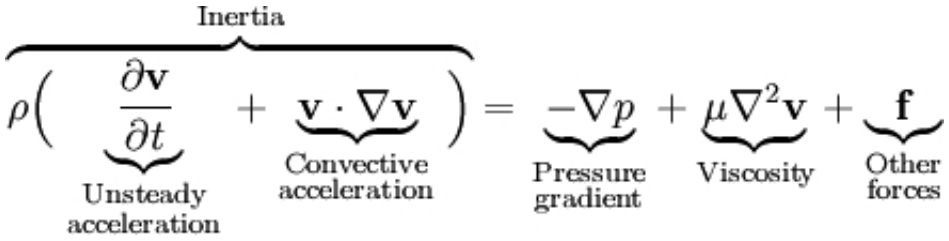

Figure 1.5: The incompressible Navier-Stokes equation.

Chapter 2

State of the Art: Current Fluid Simulation Techniques

2.1 The MAC Method - Grid Based Eulerian Fluid Simulation

The marker and cell (MAC) method was first introduced in 1965 by Harlow and Welch [9]. They proposed a grid-based method to solve the 3D Navier-Stokes equations using particles as the markers for the fluid surface. The first attempts to use this method for a physically based animation were by Foster and Metaxas [10]. This work was the first application of computational fluid dynamics techniques to the animation of 3D fluid flow. The problem with this method was that the integration over time had to be explicit and so a fixed small time step had to be used. Stam [6] relieved this problem by introducing an unconditionally stable model for fluids using semi-Lagrangian velocity advection. This method of semi-Lagrangian integration tends to dissipate a lot of energy due to the repeated averaging and interpolation within the discrete velocity field which in most cases is used as an implicit solver for viscosity. This was an important step towards achieving a real-time fluid simulation because the time step could now be large enough and not set to a fixed amount. There are a few problems with this method such as volume and energy dissipation which makes it difficult to simulate highly detailed fluid flow. In the MAC method the incompressible Navier-Stokes equation is used, as mentioned earlier, due to the simplification that

it lends itself to. Also the viscosity term is generally ignored due to the numerical energy dissipation. This simplifies the equation again to an inviscid fluid solver which is usually referred to as the Euler equations 2.1.

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p + \vec{g}$$

Figure 2.1: Euler's Equation.

2.1.1 The Projection Method

In order to calculate the velocity term for the next time step we can use the projection method of Chorin [11]. This means that we use forward differencing.The first step is to apply this method to the equation in 2.1 in order to get the equation in 2.2.

$$\vec{u}^{(n+1)} = \vec{u}^{(n)} + \Delta t \left(-\vec{u} \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p + \vec{g} \right)$$

Figure 2.2: Getting $\vec{u}^{(n+1)}$ using forward differencing.

Now we can compute an intermediate velocity field that ignores the pressure term as shown in 2.3.

$$\vec{u^*} = \vec{u}^{(n)} + \Delta t \left(-\vec{u}^{(n)} \cdot \nabla \vec{u}^{(n)} + \vec{g} \right)$$

Figure 2.3: The intermediate velocity.

Applying this to the equation in 2.2 we can rewrite the equation in accordance with forward differencing as shown in 2.4.

The problem is now that the pressure term is from the next time step. The reason for this is that Chorin's solution method is implicit in pressure and explicit in velocity.

$$\vec{u}^{(n+1)} = \vec{u}^{*} - \frac{\Delta t}{\rho}\nabla p^{(n+1)} \tag{3.10}$$

Figure 2.4: Getting $\vec{u}^{(n+1)}$ using forward differencing re-arranged to include the intermediate velocity.

This means that the pressure field at the next time step is computed directly from the intermediate velocity field. Knowing that the velocity field at the next time step should be divergence-free i.e. incompressible, we can find a way to solve for the pressure term. By maintaining the simplified conservation equation 1.4 for an incompressible fluid this equation can be re-written as shown in 2.5

$$\nabla \cdot \vec{u}^{(n+1)} = \nabla \cdot \left(\vec{u}^{*} - \frac{\Delta t}{\rho}\nabla p^{(n+1)} \right) = 0$$

Figure 2.5: Applying the incompressible continuity equation.

This equation can be re-arranged to form a known mathematical pattern known as the Poisson equation shown in figure 2.6.

$$\nabla \cdot \left(\frac{1}{\rho}\nabla p^{(n+1)} \right) = \frac{1}{\Delta t}\nabla \cdot \vec{u}^{*}$$

Figure 2.6: The Poisson equation form for $p^{(n+12)}$

By solving this equation for which many solvers exist, we can find the pressure field for the next time step.The equation in figure 2.4 can then be solved by subtracting the gradient of the pressure field from the intermediate velocity field to produce a final velocity field that satisfies both the momentum equation and the incompressibility condition for the next time step. Notice that, since the pressure field is defined implicitly

from the intermediate velocity field $\vec{u}^*$ and the incompressibility constraint, the scale of the pressure field doesn't matter as long as we treat it as the same value between the poisson equation and the equation in figure 2.4. This allows a simplification of these equations by removing the density variable completely which makes sense because it is constant through out the fluid. We can define a scaled pressure of $p^* = p^{(n+1)}\Delta t/\rho$ Applying this to the poisson equation it can be re-written as shown in figure 2.7.

$$\nabla^2 p^* = \nabla \cdot \vec{u}^*$$

Figure 2.7: The poisson equation re-written to remove density

And the equation for the velocity in the next time step can be re-written as

$$\vec{u}^{(n+1)} = \vec{u}^* - \nabla p^*$$

Figure 2.8: Simplified equation for the next velocity

2.1.2 MAC Grid Discretization

In order to solve the Euler equations in a discrete manner, the fluid's domain is approximated for the field values of $\vec{u}$ and p by a finite number of equally spaced samples. This makes it easy to evaluate spatial derivatives and interpolate quantities at arbitrary points within cells. However, instead of sampling the velocity and pressure at the same location, the MAC method uses a special grid arrangement that is designed to conserve mass. In a two-dimensional case, the pressure samples are located at the centre of grid cells, the horizontal component of velocities are located at the midpoints of vertical cell edges, and the vertical component of velocities are located at the midpoints of horizontal cell edges. It follows that in the three dimensional case that the x-axis component of velocities is located at the midpoint of x-minimum cell face aligned and so on for the y and z axis components. This configuration of field variables is referred to as the MAC grid. The projection method can now be applied to update the velocity field from $\vec{u}^{(n)}$ to $\vec{u}^{(n+1)}$. First, the intermediate velocity field $\vec{u}^*$ is computed

as shown previously. Then using the equation in 2.7 we can compute the pressure field p^*. Finally, the new divergence-free velocity field $\vec{u}^{(n+1)}$ is computed from $\vec{u}^*$ and p^* using the equation in figure 2.8. The easiest way to perform these steps is to discretize all of the differential operators that appear in these equations using a combination of forward and central difference approximations as done in [10]. In particular, the discretization of a Laplacian operator (see figure 2.7) results in a linear equation at each pressure grid point, coupling its value to its neighbours. This system of simultaneous equations can be written as a matrix equation of the form $\mathrm{A}\vec{x} = \vec{b}$, where A is a matrix of integer pressure coefficients, $\vec{x}$ is a vector of unknown pressures, and $\vec{b}$ corresponds to the right hand side of then equation in figure 2.7. Conveniently, this system has the properties of being symmetric and positive definite. A large number of standard techniques can be used to solve systems of this type. The most common method used by modern simulators is the incomplete Cholesky conjugate gradient algorithm [12], this is an iterative method with excellent convergence properties. Unfortunately solving the pressure Poisson equation is still, by far, the most computationally expensive part of MAC-based solvers.

2.1.3 Boundary Conditions

In order to facilitate application of boundary conditions in a MAC simulation, grid cells need to be categorized at the beginning of each time step. Each cell can either be a liquid cell, an air cell, or a solid cell. Voxelizing boundaries in this way requires them to coincide exactly with the edges of computational cells which greatly simplifies the enforcement of boundary conditions. However, this is a rather poor approximation for most surfaces, especially on coarse grids. Although only those cells marked as liquid need to be updated each time step, it is still necessary to set physically meaningful velocity and pressure values in all cells adjacent to liquid cells since these may be accessed by the stencils used in finite difference approximations, interpolation operators, etc. The appropriate values can be inferred from the equation in figures 2.9 and pressure as it only appears as a gradient can be set to zero for an air liquid boundary.

For example, in the context of the Poisson solver, equation 2.9 is respected by requiring that the normal pressure gradient equals zero at edges that lie between solid and liquid cells. In cases where the fluid does not fill the entire computational domain,

$$\vec{u} \cdot \vec{n} = \vec{u}_{solid} \cdot \vec{n}$$

Figure 2.9: Boundary Condition

it is necessary to track the location of the liquid-air interface as it moves around. By tracking the location of this moving boundary, we can determine which cells are air and which are liquid and then place boundary conditions at the appropriate locations. The original MAC method uses massless marker particles to represent the liquid. These are advected based on the fluid velocity locally interpolated at their positions. Liquid cells can then be identified as those cells that contain at least one particle. An alternative to marker particles that has become very popular in recent years is to capture the liquid-air interface as the zero level set of a dynamically signed distance function ϕ that is sampled on the simulation grid [5]. This provides a smoother representation of the liquid surface that gracefully handles extreme deformations and topological changes. In order to advance the liquid surface based on the fluid velocity field, the discrete ϕ samples need to be updated according to the equation in figure 2.10.

$$\partial\phi/\partial t \; = \; -\vec{u}^{(n+1)} \; \cdot \nabla\phi$$

Figure 2.10: The advection equation.

The grid cells can then be classified based on the sign of ϕ at each cell. The downside of the implicit surface representation is that noticeable volume loss occurs due to numerical dissipation when solving the advection equation. This problem can largely be overcome however by using the particle level set method [13] which augments the signed distance field with particles in order to correct under sampling errors in the position of the zero level set.

2.1.4 Semi-Lagrangian Advection

Previously I showed how to use finite differencing to resolve the spatial derivatives of the advection term $-\vec{u} \cdot \nabla\vec{u}$. When this is used in conjunction with an explicit time integration, this discretization imposes a severe restriction on the size of the time step

that can be used to advance the simulation. If the time step is too large then the velocity values will oscillate and the simulation will become unstable. To stabilize the time step it must adhere to the Courant-Friedrichs- Lewy (CFL) condition which limits the size of the time step as shown in figure 2.11. where Δx is the spacing between grid points.

$$\Delta t = \frac{\Delta x}{\max\left(\left|\vec{u}\right|\right)}$$

Figure 2.11: The Courant-Friedrichs-Lewy condition

This condition means that if a field quantity moves across a grid, it must be integrated using a time step small enough to ensure that no grid points are skipped. Fortunately no time step restriction is associated with the pressure derivatives because when Chorin's projection method was applied the pressure field was resolved implicitly to satisfy the governing equations. Obviously for a real-time application we'd like to be able to choose a time step that's as large as possible in case of a drop in frame rate etc. . An unconditionally stable technique known as semi-Lagrangian integration for computing the advection term was proposed by Stam [6]. This allows any time step to be used without limitation by the CFL condition (though obviously accuracy is dependent on the time step). This is done by using a Lagrangian technique called particle back tracing. This simply bridges the gap between time step for the fixed points in space that any quantity q is sampled at and the "floating" point in space that these quantities were at before a single time step. Figure 2.12 shows this in an easier to understand visual form.

Since x(n) will not typically coincide with a grid point, interpolation is necessary to evaluate q(n). Using linear interpolation for this purpose ensures stability because the new field will never be larger than the largest value of the previous field no matter what time step is used. But this also has the undesirable effect of smoothing out the advected field variables. This can be alleviated to some extent by using higher order interpolation schemes. But numerical dissipation is inherent in the semi-Lagrangian method and can not be defeated entirely. The equation to calculate q^{n+1} is shown in figure 2.13.

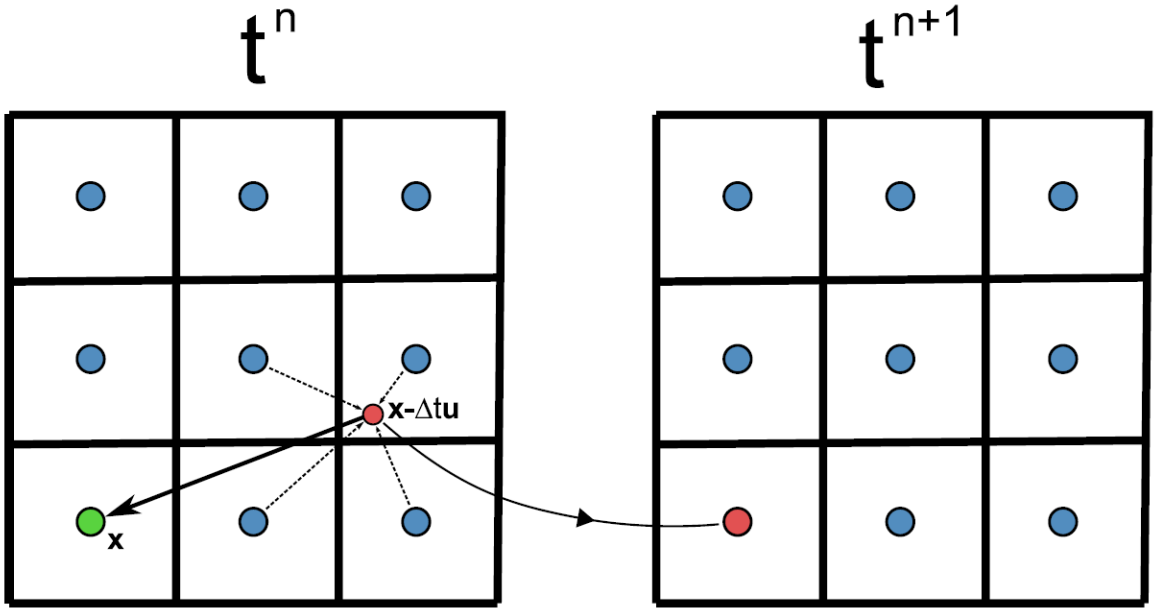

Figure 2.12: A field q is sampled at the centre points of grid cells. To compute $q^{(n+1)}\vec{x}$, we trace backwards through the velocity field and interpolate $q^{(n)}$ at the old point that q was at, $\vec{x} - \Delta t\vec{u}(\vec{x})$.

$$q^{(n+1)}\left(\vec{x}^{(n+1)}\right) = q^{(n)}\left(\vec{x}^{(n+1)} - \Delta t\vec{u}\left(\vec{x}^{(n+1)}\right)\right)$$

Figure 2.13: How to calculate a quantity using particle back tracing.

Also for non-constant velocity fields, forward Euler integration can be very inaccurate and unstable. Therefore, it is usually desirable to use more accurate integration methods to better preserve rotational motion in the flow field. A good alternative is the midpoint method which can be implemented by replacing $\vec{x}^{n+1} - \Delta t\vec{u}(\vec{x}^{n+1})$ with $\vec{x}^{n+1} - \Delta t\vec{u}(\vec{x}^{n+1} - \frac{1}{2}(\vec{x}^{n+1}))$.

2.2 Smoothed Particle Hydrodynamics

A definition for SPH taken from [3]:"SPH is an interpolation method for particle systems. With SPH, field quantities that are only defined at discrete particle locations can be evaluated anywhere in space. For this purpose, SPH distributes quantities in a

local neighbourhood of each particle using radial symmetrical smoothing kernels."
The paper "Particle-Based Fluid Simulation for Interactive Applications" was first presented at Siggraph 2003 by Matthias Muller, David Charypar and Markus Gross, it was the first published paper on using SPH to simulate fluids. The paper's focus was on implementing a real-time interactive simulation of a fluid by using a particle system to simulate the behavior of water. This was achieved by implementing the Navier-Stokes equations using the Smoothed Particle Hydrodynamics methodology. SPH was first developed to deal with simulating astrophysical phenomenons [14] but the method is general enough to allow it to be applied to the simulation of fluid. SPH had previously been used by Jos Stam and Eugene Fiume who first introduced SPH to the graphics community to simulate fire and other gaseous phenomena [15]. More work would be done in this area by Desbrun who used SPH to animate highly deformable bodies [16]. SPH for use in fluid simulation is an extension of Debrun's method focusing on resolving force due to pressure and viscosity. In order to do this, the equations to resolve force due to viscosity and pressure are derived directly from the Navier-Stokes equation for implementation through SPH. For the purpose of stability, special purpose smoothing kernels were designed as the sampling functions for resolving different field values.

2.2.1 The SPH methodology

According to SPH, a scalar quantity A is interpolated at location r by a weighted sum of contributions from all particles within distance h as shown in 2.14 .

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

Figure 2.14: Interpolating a scalar quantity using SPH.

The function W(r;h) is called the smoothing kernel with core radius h. Kernels with finite support h are used so that we only need to sample from particles within that support radius. There are various techniques to efficiently find these neighbours. If W is even (i.e. W(r;h) =W(-r;h)) and normalized, the interpolation is of second

order accuracy. The kernel is normalized if it's integral for a value r with respect to r is equal to 1 as shown in 2.15.

$$\int W(\mathbf{r})d\mathbf{r} = 1.$$

Figure 2.15: Property of a normalised kernel function.

Another important property given the nature of the Navier-Stokes Equation and SPH itself is that density can be found using the equation shown in 2.16.

$$\rho_S(\mathbf{r}) = \sum_j m_j \frac{\rho_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) = \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h).$$

Figure 2.16: Calculating density using SPH.

Again for use in the Navier-Stokes Equation we need to be able to find the gradient and laplacian terms which are calculated simply by using the gradient and the laplacian of the kernel function. So the gradient is calculated using the equation shown in 2.17.

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

Figure 2.17: Calculating the gradient of a value using SPH.

And the Laplacian is calculated using the equation shown in 2.18.

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

Figure 2.18: Calculating the laplacian of a value using SPH.

2.2.2 Why use SPH for Fluid Simulation

SPH lends itself naturally to simulating fluids because of the physical properties that define the behavior of a fluid. Physically speaking a fluid is made up of an extremely large number of particles, essentially the molecules in the fluid. By definition in the Navier-Stokes Equations each of these particles' movements are effected by force vectors which are in turn produced as a result of the properties of the fluid in that point in space and by the properties of the fluid in the surrounding area which determine the forces being exerted in that area. By interpolating these quantities in a local neighborhood for a point in space using a smoothing kernel these forces can be resolved in a much more stable manner than they are for the grid-based methods. They are also resolved in a more efficient manner completely avoiding grid descretization issues and the Poisson equation. Another benefit is that SPH doesn't suffer from the boundary issues that Grid-based methods do. Collision detection and response can be calculated on a particle by particle basis with static scene geometry without producing any artifacts in the simulation. The short-comings of SPH are mainly down to the problem that lots of particles are needed to give sufficient detail to a body of fluid as well as the fact that the particles are compressible.

2.2.3 Resolving the Navier-Stokes Equations using SPH

To resolve the field force vectors a simplified version of the Navier-Stokes equation is used as shown in 2.19.

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v}$$

Figure 2.19: The Navier Stokes Equation to which SPH is applied for fluid simulation.

This describes an incompressible fluid, where p is pressure, $\mathbf{g}$ is gravity, $\mathbf{v}$ is velocity, t is time and μ is the viscosity term. But in practice in SPH because of the varying density, and by relation pressure, the volume for each particle is compressible because volume=mass/density and each particle has a fixed mass. The use of particles instead

of a stationary grid simplifies resolving Navier Stokes equation substantially. First, because each particle has a constant mass, mass conservation is guaranteed and the continuity equation can be ignored. Second, the expression on the left hand side of the equation multiplied by density can be replaced by the substantial derivative $\frac{D\mathbf{v}}{Dt}$. This is because the particles move with the fluid, therefore the substantial derivative of the velocity field is simply the time derivative of the velocity of the particles meaning that the convective term is not needed for particle systems. So the Navier-Stokes Equation can be reduced to the equation shown in 2.20.

$$\mathbf{f} = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v}$$

Figure 2.20: Reduced Navier Stokes Equation for a lagrangian method.

Where f is $\rho\frac{Dv}{Dt}$ known as the force density term so to get the acceleration for each particle in the system we can use the formula shown in 2.21.

$$\mathbf{a}_i = \frac{d\mathbf{v}_i}{dt} = \frac{\mathbf{f}_i}{\rho_i}$$

Figure 2.21: How to calculate acceleration from Force density.

Now that we have a method for resolving the acceleration of each particle we can now work on resolving the force density terms due to pressure and viscosity.

2.2.4 Resolving The Field Force Vectors

Fluid particles are simulated with force vectors that move with the particles. Density is calculated using the SPH formula shown in 2.16. The field force density vector of pressure is calculated for each particle by sampling other particles within distance h and smoothing their contribution using SPH methodology and the equation in 2.22.

Where p_j is the pressure of the particle being sampled. Because these samples are discrete contributions from physical "pieces" of fluid, which do not have a common

$$\mathbf{f}_i^{\text{pressure}} = -\nabla p(\mathbf{r}_i) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h)$$

Figure 2.22: How to calculate force density due to pressure using SPH.

pressure, there is need to preserve Newton's third law for every action there is an equal and opposite reaction. This means modifying this equation to make it symmetrical, this will also increase the computational efficiency of the algorithm. Muller found that this simple modification, using the average of the pressures for the two particles, works best in achieving a stable fluid simulation as shown in 2.23.

$$\mathbf{f}_i^{\text{pressure}} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h)$$

Figure 2.23: How to calculate symmetrical force density due to pressure using SPH.

Pressure can be calculated from density using the gas constant formula: $p{=}k\rho$.

However to provide numerical stability throughout the particle system it's better to have an offset for the pressure term which can be considered the pressure at rest i.e. when a particle's pressure doesn't contribute to the force density due to pressure for itself or any other particle. In reality this is when a particle's pressure value is equal to zero. A good way to choose this offset is to use a multiple of the density for a particle when the only particle sampled is itself i.e r is equal to zero. I found that by using twice this value for the rest density that this helps keep the system stable. In order to resolve the pressure for each particle we should now use the formula: $p{=}k(\rho - \rho_0)$.

The scalar value k corresponds to the gas constant of the fluid we are trying to simulate but because the particles represent "pieces" of fluid and not fluid molecules, it's best to use a user defined constant and tweak it according to how volatile you would like the fluid to appear. In order to calculate a symmetrical force density vector due to viscosity for a single particle, other particles within distance h are sampled and their contribution is smoothed using SPH methodology and the equation in 2.24:

Where $\mathbf{v}_j$ and $\mathbf{v}_i$ are the velocity of the particle being sampled and the velocity

$$\mathbf{f}_i^{\text{viscosity}} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h)$$

Figure 2.24: How to calculate symmetrical force density due to viscosity using SPH.

of the particle being resolved respectively. By examining the formula it's easy to see that this term is really just to smooth the effects of the pressure force by subtracting a particles velocity from that of its neighbors and producing a force that smoothes a particle's velocity according to the area it's occupying. It can be interpreted that μ here is another scaling factor that helps a system find stability quicker but scientifically speaking this is known as the viscosity coefficient.

2.2.5 The Smoothing Kernels

The stability, accuracy and speed of the SPH method highly depend on the choice of smoothing kernels. The most important factor for picking a kernel was that they displayed the trends desired for the variable they were being used to calculate. Obviously they also need to be symmetrical and normalized in order to fulfill the rules of SPH. For density the kernel shown in 2.25.

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases}$$

Figure 2.25: Smoothing kernel for density.

This was used because it not only provided a smooth roll off from r=0 to h and was zero when h=r but also because r (the distance) only appears squared removing the need to use an expensive square root operation when calculating density.

In figure 2.26 the thick line is the kernel function the thin line is the gradient towards the centre and the dashed line is the Laplacian, h=1 for all of these diagrams. It is clear from the graph that the *poly6* kernel function is not suitable for the pressure

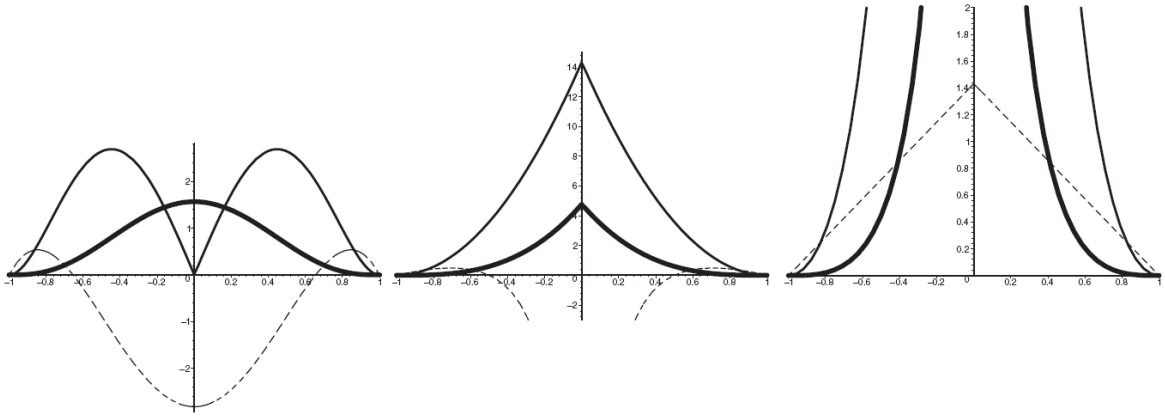

Figure 2.26: A graph of the kernel functions used in SPH.

term as its gradient is zero at r=0 so another kernel function known as the DeBrun's spiky kernel was used shown in figure 2.27.

$$W_{\text{spiky}}(\mathbf{r},h) = \frac{15}{\pi h^6}\begin{cases}(h-r)^3 & 0 \le r \le h \\ 0 & \text{otherwise,}\end{cases}$$

Figure 2.27: The Desbrun spiky kernel function.

It can be seen from the diagram that this kernel function gives a nice distribution over the gradient and doesn't go to zero at r=0. Unfortunately neither the poly6 nor the spiky kernels are suitable to be used for the viscosity term as their laplacian's are negative at low r values. So a third kernel function was designed for the viscosity term that has a positive laplacian value for all values of r between -h and h. This function is shown in figure 2.28.

$$W_{\text{viscosity}}(\mathbf{r},h) = \frac{15}{2\pi h^3}\begin{cases}-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \le r \le h \\ 0 & \text{otherwise.}\end{cases}$$

Figure 2.28: The viscosity kernel function.

One further property of this kernel is that it's laplacian value shown in figure 2.29

is very similar to that of the magnitude of gradient of the spiky kernel shown in figure 2.30.

$$\nabla^2 W(\mathbf{r}, h) = \frac{45}{\pi h^6}(h - r)$$

Figure 2.29: The laplacian of the viscosity kernel function.

$$\nabla W(\mathbf{r}, h) = \frac{45}{\pi h^6}(h - r)^2$$

Figure 2.30: The magnitude of the spiky kernel.

This makes computation more efficient.

2.2.6 An efficient SPH algorithm

In order to make the SPH methodology as efficient as possible it's important to implement an efficient algorithm for solving the acceleration for each particle and updating them every frame. The first thing to do in every frame is to sort the particles using spatial hashing method into grid cells in 3-d space that are the same width as the smoothing length being used in the SPH calculations. This makes populating a pair list more efficient, the reason we use a pair list is because as already stated the force density between two particles within a smoothing radius is symmetrical i.e. the contribution for $particle_j$ of a pair is the negative of the contribution for $particle_i$. For density the contribution is added to both particles (obviously as there can't be a negative contribution to density) that form the pair. This will effectively cut the amount of time spent resolving contributions for each particle in half as described in [17]. The pair list is populated by searching for neighbours for each particle in it's own cell and it's neighbouring cells taken care not to add the same pair. Then density is resolved for each particle followed by the force densities and acceleration for each particle (two stages). The particle's position and velocity is then updated and any collision detection and response can also be applied at this stage. This algorithm is repeated for each frame.

2.2.7 Augmentations to the SPH method

Not many additional changes have been made to the SPH method since it's inception one that has been made, however, is the use of adaptive particle sampling [7]. This is a method of splitting and merging particles into particles of different mass to produce different sized particles. The particles are split or merged depending on their distance from the medial axis which needs to be calculated each time the split and merge procedure takes place. In their implementation the authors used a 4 tier system to produce 4 differently sized particles with 4 different smoothing lengths and related the latter so that spatial hashing could be conducted efficiently when looking for neighbouring pairs to compute contributions for. They also altered Mullers equations to preserve symmetrical forces these are shown in figure 2.31.

$$\mathbf{f}_{ij}^{\text{pressure}} = -V_i V_j (P_i + P_j)(\nabla W(\mathbf{x}_{ij}, r_i) + \nabla W(\mathbf{x}_{ij}, r_j))/2,$$

$$\mathbf{f}_{ij}^{\text{viscosity}} = \mu V_i V_j (\mathbf{v}_j - \mathbf{v}_i)(\nabla^2 W(\mathbf{x}_{ij}, r_i) + \nabla^2 W(\mathbf{x}_{ij}, r_j))/2,$$

Figure 2.31: Calculating force density using adaptive SPH.

This method certainly reduces the number of particles needed to model a volume of fluid (and thus the computation) with adaptation to allow a higher level of detail for the fluid surface. It actually helps with the problem of compressibility as well because of the adaptive nature of the smoothing length. Unfortunately the means of extracting the median axis and in particular finding pairs of particles to split or merge proved computationally expensive. The authors, in a bid to find a middle ground, decided to implement the split and merge procedure only once every 5 frames but still failed to yield much of an increase in performance over regular SPH. The additional computational expense for the adaptive density force algorithms probably contributed to this. Although for this approach it is difficult to compare performance against regular SPH. Specifically the question must be asked, is a smaller number of adaptive particles the equivalent of a larger number of regular particles if they fill the same volume? It's hard to compare the two unless you conduct some sort of perceptual testing. Another augmentation was a two way coupled SPH and particle level set (A

grid-based method) simulation. This combined the superior performance of SPH in diffuse areas with the enforced incompressibility of the particle level set method but this methodology is not a real-time application.

2.3 Rendering: Screen Space Meshes

Many methods have been proposed for rendering fluids but few are suitable for real-time applications. There is of course the infamous marching squares algorithm [18] that is far too costly. Point splatting [19] which doesn't look good in diffuse areas. In [7] they used their median axis method to produce a distance to the fluid surafce method that worked well but was computationally expensive. Out of all the methods one jumps out as the most suitable for real-time applications which is what I am going to detail in this section. Screen space meshes were first used to render the air-fluid interface in screen space of an SPH simulation by Muller in 07 [2]. The paper itself describes the mesh generation being done on the CPU. Effectively a 2-d depth map of nodes (the resolution is chosen by the user) of the particle system is created from the point of view of the camera i.e. in screen space. The particles themselves are treated as spheres with a user chosen radius and their radii are also projected into screen space. Muller projected each particle's centre into screen space and checked then for 2-d intersection with the nodes surrounding the centre of the particle using the projected radius. If an intersection occurs the Z-value for that point on the sphere representing the particle is calculated with simple geometry using the projected radius and the distance from the 2-d intersection to the centre of the Sphere as well as the particle's distance from the camera (i.e it's depth). Any particle that contributes a depth value to a node that is less than that of the previous occupant obviously replaces it. This way these nodes are populated with values that represent the silhouettes of an approximate air-fluid in screen space. Smoothing is used to remove bumpiness which is a problem with techniques that use spheres to estimate a surface represented by points in space. Smoothing is performed on the depth map of the nodes before transforming them back to world space using a kernel function (they used a gaussian kernel in just the x and y direction with a radius of 3 nodes using 2 passes the first in the x direction and the second in the y direction). The kernel function's size is limited at the edge of each silhouette to make sure it doesn't smooth over the edge. Marching squares algorithm

is used to construct meshes with normals from the silhouettes which are then projected back into 3-d space. This 3-d mesh can now be rendered with reflection/refraction and lighting easily.

Chapter 3

Control Point Smoothed Particle Hydrodynamics

Control Point SPH is an attempt to combine features of both Grid-based methods and normal SPH. For my dissertation I proposed using a particle system with a grid of control points spaced at intervals in 3-d space at which the pressure and viscosity forces could be partially resolved based on the SPH methodology as described in the previous section. The particles in the system used used as they are in a regular SPH implementation as mass carrying "pieces"' of fluid that have their own velocity, pressure and density, see figure 3.1. Density and pressure would be calculated at each control point and then interpolated for each particle in the system. Then the acceleration due to pressure and viscosity forces experienced can be partially resolved (explained later)) at the control points. These force densities are then interpolated for each of the particles and the acceleration for the given particle is resolved. This method is an attempt to combine the robustness and simplicity of SPH with interpolation methods in order to increase the speed at which a fluid simulation can be conducted. My hope was this would be a much more efficient SPH-based methodology albeit less accurate.

3.1 The Algorithm

The algorithm for implementing control point based SPH follows the same steps as regular SPH; find neighbours (for the control points), resolve density, resolve acceleration

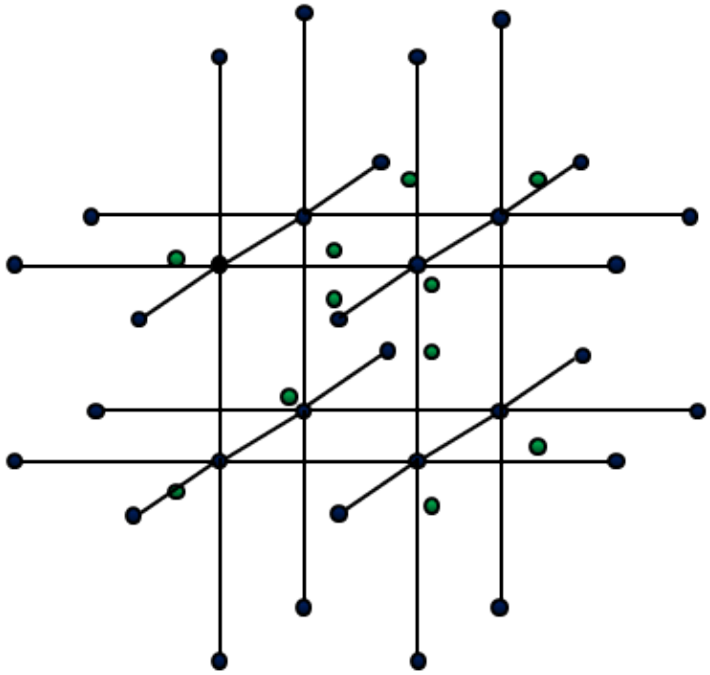

Figure 3.1: A visual representation of the control point SPH methodology. Here the Green dots are the particles in the system and the black dots set in the grid are the control points.

and update particles. But there are also intermediate steps for interpolating SPH field values for the particles. What follows is a description of each of the steps in the order they should be executed in. Some of these steps have issues associated with them that will be elaborated upon later.

3.1.1 Binning the Particles

The particles are binned into several nodes based on their position in the grid using spatial hashing. Unlike regular SPH we can cull any particles that aren't definitely within the smoothing length of a control point because the control points are fixed in place. The particles' distances from the control points should be stored at this point in order to make computation more efficient. Control points should be spaced at an interval less than or equal to h, but h should be a multiple of this grid width in order to ensure maximum efficiency when binning the particles. This means that if the grid

width is equal to h a particle's distance must be tested for 8 possible nodes for which it may be within the smoothing length of. For a grid width of h/2 there are 64 possible nodes for which it may be within the smoothing length of, and so on. Figure 3.2 shows a visual depiction of this.

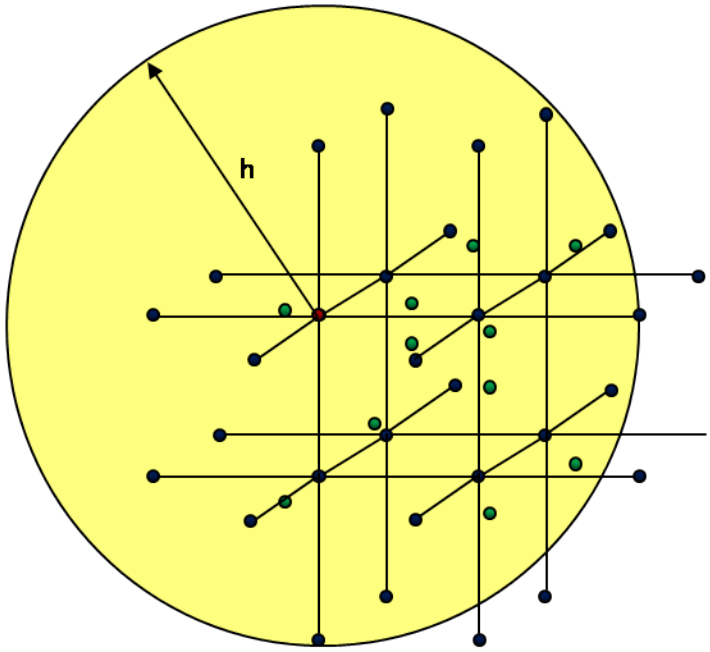

Figure 3.2: Relationship between smoothing length and grid width regarding binning.

Issues and Optimizations

Each of the particles must be linked to the eight surrounding nodes of it's cell for use when interpolating the density and acceleration for a given particle from the values computed at the control points. In my implementation I found that it is best to link the particle's to the control points just by maintaining pointers to the values for density and to structures containing the partially resolved force densities for the control points. This is because the nodes representing the control points will contain a buffer of pointers to the particles that they are linked to and this will thrash the cache with unnecessary data when trying to interpolate values for the particles. When binning to the grid nodes a list of active nodes should be populated, active grid nodes are

only those that particles are linked to i.e just those from which the particle's will be interpolating values from.

3.1.2 Resolving Density

Density and pressure is resolved for each control point using the SPH methodology as described in the previous section. Worth noting here is that in a regular SPH implementation particles will have a density contribute due to themselves (the density contribute when r is equal to zero) which obviously can't happen in this implementation because the control point is not a particle. As explained in section 4.1 this contribution can be pre-computed easily and if used as the initial value for the grid node's density when summing the contributions from the neighbouring particles this problem is easily overcome. Once density has been resolved for all the active control points it is interpolated for the particles the interpolation practice is discussed in section 3.2.1. Pressure is then calculated for the particles in the normal SPH fashion.

3.1.3 Resolving Acceleration using Partially Resolved Force Densities

Force density due to pressure and viscosity is partially resolved for each grid node. Because the contribution to force densities for both pressure and viscosity resolve to zero when a particle is paired with itself we don't have to worry about initial values for the force densities they are simply the zero vector. The reason why the force density for viscosity is only partially resolved is because the grid node has no velocity. This is overcome by splitting the equation for calculating the viscosity using the two equations as shown in figure 3.3.

$$\mathbf{f}_i^{\text{viscosity}} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h).$$

$$\mu \sum_j m_j \frac{\mathbf{v}_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h). \; - \; \mathbf{v}_i \; \mu \sum_j m_j \frac{1}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h).$$

Figure 3.3: The viscosity force density equation can be split in this manner because v_i is constant.

Now we end up with two terms for the viscosity force density resolved at a control point. One is a float, corresponding to the series on the right, which is to be multiplied by the particles velocity before interpolation, corresponding to v_i. This term is then subtracted from the partially resolved force density on the left (a vector) in order to fully resolve the force density for viscosity. This effectively means that the force density due to viscosity can be resolved each time a particle is interpolating from the control point as if the control point had that particle's velocity. The equation for pressure force density should not be split to allow the pressure for the control point to be substituted for the pressure for the particle, as the pressure term is specific to that location in space. Also when resolving the acceleration for a given control point to be used for interpolation, the force densities should be divided by the control point's density and not the density of the particle interpolating from it because the density is specific for that point in space. There is a reason why the acceleration at the control point is not resolved until the force densities are being interpolated for each particle and that is discussed in section 3.2.1.

3.1.4 Updating the Particle's Position and Velocity

Once the force densities are interpolated and the acceleration resolved for a particle and gravity is added it's velocity and position can be updated in a usual manner. I used the leap-frog approach where the velocity applied to the particle's position is half a time step ahead of the velocity used to calculate the SPH acceleration. This makes for a smoother simulation. Then collision detection and response is applied as described in section 4.1.

3.2 Interpolation of SPH Values

When deciding how to interpolate the SPH values for each particle I considered several options including a sampling function based on distance and the bi-cublic interpolation [20] method. I had custom designed a sampling function but I found it could lead to either a dissipation or a exaggeration of the SPH values when interpolating from the 8 surrounding nodes. The sampling function I used was based on distance to derive a weight between 1 and 0 for the 8 nodes surrounding the particle. Unfortunately

despite trying several high order polynominals in most cases these weights would not add up to exactly one hence the dissipation or exaggeration problem. Then I looked into using the bi-cubic interpolation method but unfortunately I found that in a field of high gradients such as a field of SPH values that the bi-cubic method would smooth the values for acceleration too much and would lead to very uniform fluid motion. So eventually I decided to use a very simple algorithm in order to produce 8 weights for the 8 nodes the sum of which is guaranteed to be one. These weights are calculated as the normalized inverse distance from each corner. The intermediate weight for each corner is calculated as: (Total Distance from Corners)/(Distance from This Corner). The normalized weight for each corner is calculated as: (Intermediate Weight for This Corner)/(Sum of Intermediate Weights). I found this method to work best during my implementation of control point SPH.

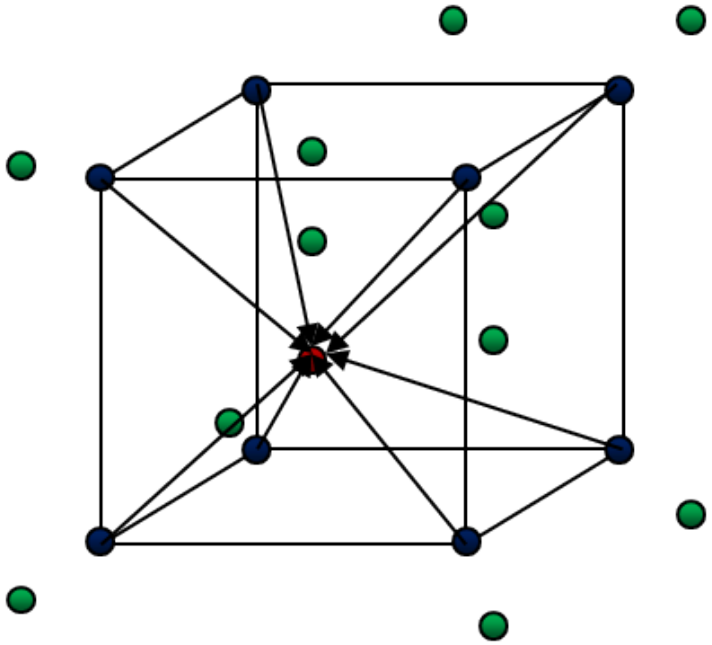

Figure 3.4: Each control point contributes an SPH value (density or a force density) to the red particle and is weighted by it's inverse distance from the particle. These weights are normalized so that they always sum to one.

3.2.1 Considerations for Interpolation of Forces

When resolving the pressure force density using regular SPH the gradient of the sampling function becomes the zero vector at a distance of zero i.e. when a particle is sampling from itself. Likewise when resolving the viscosity force density the difference in velocity between a particle and itself is zero therefore making no contribution. What this means in terms of physics is that one particle cannot exert a force on itself. If this isnt accounted for when interpolating from the control points particles will find equilibrium much quicker than intended due to the extra force exerted by itself, on itself. Visually speaking this leads to unrealistic behaviour, i.e. once a particle's velocity falls below a certain magnitude it'll stop dead and oscillate slightly. An easy way around this is to maintain the contributions that a particle makes to its eight surrounding nodes for density, the pressure force density term and the two terms needed to resolve viscosity force density. Then when interpolating the force densities due to pressure and viscosity from a control point subtract these contributions for the particle in question from the nodes force densities due to pressure and viscosity. Then use the control point's density to resolve this adjusted acceleration and carry out interpolation using the weight system. This fixes the problem of a particle exerting force on itself but unfortunately the system can never find a true equilibrium due to lack of symmetric forces between the particles and not the interpolation method.

3.3 Loss of Symmetry and Other Issues with the Methodology

The main problem with this methodology is that the accuracy for interpolated values for each particle becomes extremely skewed when there are a number of other particles within a single grid cell. SPH fluid simulation relies on symmetry in order to produce stable fluids i.e. "For every action there is an equal and opposite reaction". For particles in a regular SPH implementation the preservation of this symmetry for the forces resolved is more important than the physical accuracy due to the smoothing length of the algorithm. This symmetry effectively pushes two particles away from each other so by proxy if you increase the smoothing length of the algorithm you can also increase the "size" of the particles because it's local neighborhood now increases in size.

In control point SPH symmetry is lost completely for particles within the same cell as they interpolate their accelerations from the same nodes and so cannot mathematically produce a symmetrical contribution between each other. Outside the cell a symmetrical contribution is maintained between two particles because the particles sample from different nodes in space but the numerical accuracy may not be perfect. The numerical inaccuracy is because the interpolated contribution won't exactly match a contribution derived from the regular SPH methodology due the difference between sampling kernels and the linear nature of the weighting algorithm. This numerical inaccuracy makes little difference however as long as the contribution is symmetrical the system will find an equilibrium as the particles move in space. In fact when using control point SPH it's possible to get contributions even from particles outside the intended smoothing length of the SPH methodology but these contributions will be relatively tiny in comparison to the particles closest to the particle and because they are symmetrical we can afford to ignore them. The lack of symmetry for forces between particles in the same cell cannot be ignored however as these are the particles which should have the biggest effect on the overall force experienced by the particle.

3.3.1 How to Solve this Problem

One way to solve this problem perfectly is to increase the resolution of the Grid until such a stage that only one particle could possibly fit in the cell. This is done by reducing the grid width to the size of the radius of the smallest particle in the system. The volume of a particle can be found using the equation Volume=mass/density. We can then assume that the particles, which as already discussed are "pieces" of fluid, are spherical and then we can derive the radius using the formula $r = (V\frac{3}{4\Pi})^{\frac{1}{3}}$. Unfortunately density is not constant and obviously the highest density possible should be found (i.e. The smallest particle). Statistically speaking it is difficult to determine the highest density in a SPH simulation because it depends on two many factors; K, the smoothing length, the number of particles involved, the type of container they are in, the mass of the particles and more. Its easier just to find out the highest possible density using a regular SPH implementation. At this resolution, however, control point based SPH could not possibly run in real-time. I found that reasonably good simulations can be achieved using a grid interval of half the smoothing length although

there will be some instability in the denser parts of the fluid. A better solution might be found by combining control point SPH with regular SPH and I have detailed my hypothesis for this solution in future work.

3.3.2 Linear Distribution of Density

In a regular SPH implementation density is calculated for each particle using the SPH methodology i.e. by sampling from surrounding particles. It's logical to conclude that in an evenly distributed particle system, particles found towards the centre of the system would have a higher density than those within the smoothing length's distance of the surface of the particle system (this includes those at the bottom of system). This is why the fluid particles are compressible in SPH particularly for particles at the bottom of the container. In order to exert a force strong enough to counteract gravity and suspend the particles above them the particles at the bottom need to be compressed i.e. the distance between the bottom particles and the particles above them will decrease in order to increase the density of the lower particles and reach an equilibrium. This holds true for each stratum of particles in the system thats within the smoothing length of a boundary. In a lot of real time simulations this can mean the majority of particles because deep pools of fluid means extra computation that won't contribute to the surface animation which is what we're generally interested in. An example of this compression is shown in figure 3.5 .

In other words an SPH implementation can never achieve equilibrium as an evenly distributed particle system. This means most of the particles in the system aren't contributing anything to the surface animation which would be acceptable in some cases if they were providing an efficient increase in volume but they are not. In control point SPH, however, density at each particle is interpolated from the density at the control points. This has a smoothing effect to the change in density from the particles found in the space within the fluid volume, out to the particles at the surface and by association their pressures. This has a knock-on effect of reducing the compressibility factor when calculating the force densities at the control points, this is apparent from the formula to calculate force density due to pressure in figure 2.23. The compressibility factor is not completely removed of course but it is reduced as shown in figure 3.6 resulting in a substantial increase in volume for the same number of particles while having only

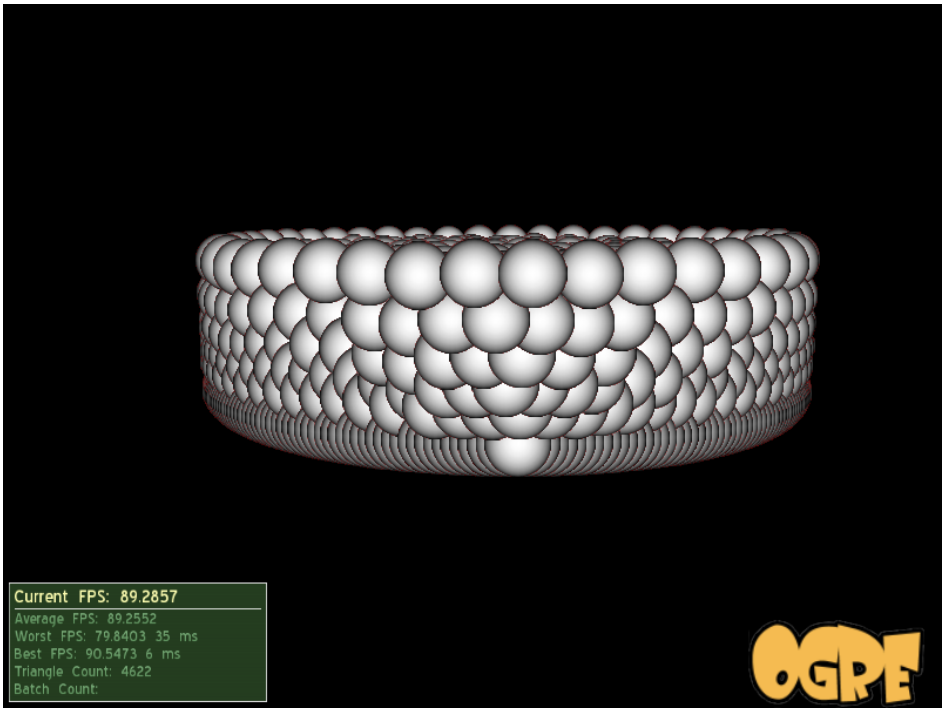

Figure 3.5: The compressed particle strata of a regular SPH implementation.

slight negative effects on the surface detail of the air-fluid interface as shown in figure 3.7.

3.3.3 Other Considerations

The loss of symmetry also makes the computational expense for each control point more expensive than for each particle in a regular SPH implementation. Symmetrical forces means that instead of resolving density, pressure and viscosity forces for each individual you can resolve individual contributions on a pair by pair basis. This effectively means that you dont do the same calculation twice. A pair list can be populated at the start of a regular SPH simulation by conducting an efficient neighbour search using spatial partitioning. This is not possible with Control Point SPH because particles are paired with the control points and not each other.

Figure 3.6: The less compressed particle strata of a control point SPH implementation

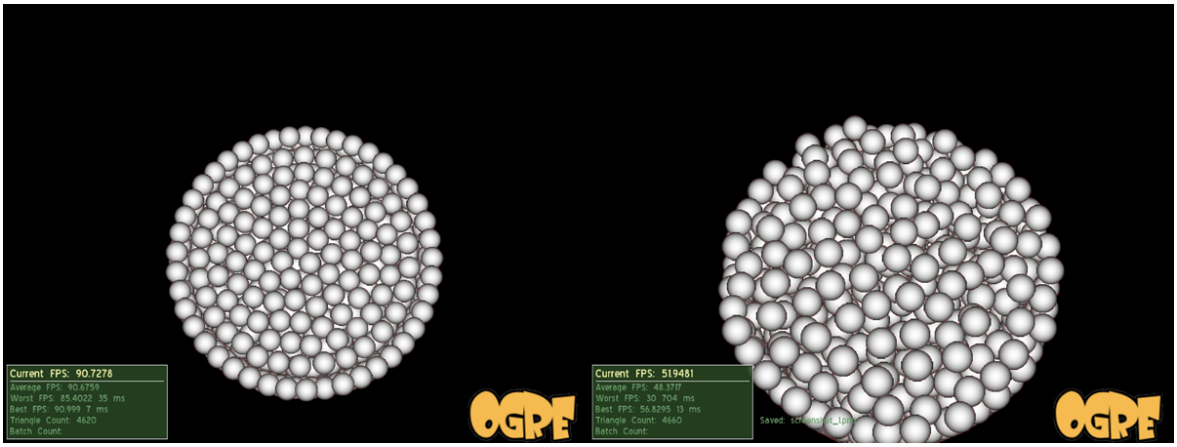

Figure 3.7: A comparison of the surface detail of SPH (left) and of control point SPH (right)at equilibrium. In SPH the particles are spread evenly across the surface due to perfect symmetrical forces but the number of particles at the surface for SPH and control point SPH are relatively the same and so the level of detail obtained when the fluids are in motion is effectively the same.

Chapter 4

Results

I developed two fluid simulators, one using my control point SPH methodology and another using a regular SPH methodododology for comparison. Both were developed in C++ using the open source rendering API, Ogre. Through the OGRE rendering API I used billboards in order to render the particles. I had planned to use these billboards to render a screen space fluid-air interface, as described in my future work section. The test results detailed in the next section were obtained while running my application on a Dell XPS with an Intel Core 2 CPU with a clock speed of 1.86Ghz and 2.0 GB of RAM using the Windows XP SP2 operating system. The graphics card is a NVidia GeForce 8600 GTS.

4.1 Results

To give a critical analysis of my results I implemented an efficient SPH implementation using the pair list method described in section 2.2.6 in order to compare it to my control point SPH method. Both simulate the movements of 2200 particles and have the same smoothing length (0.1 which is 1/10 the radius of the cylinder). In order to make the regular SPH as efficient as possible I made my own alterations to the algorithm to provide the most critical comparison possible. I then compared the performance of my control point SPH to the regular SPH implementation in terms of behaviour and speed. These alterations started with how to efficiently populate the pair list after sorting the particles into their respective cells. I found that the best way to populate the pair

list without reproducing the same pair twice is to iterate through a list of active cells (obtained from when the particle's were sorted). By 'bubbling up' through the list of particles for each cell you can avoid putting the same pair into the pair list from within the same cell. You should also save the value for distance (or distance squared) in the pair list to reduce computation for later. Then for each particle in the cell I searched in any active neighbouring cells for pairs. You can then make the current cell inactive as all the pairs possible for the particles in this cell have been found. By doing this for each cell on the active cell list the pair list will be populated quickly and perfectly. Now density needs to be resolved for each particle so iterate through the pair list resolving a single contribution at a time and adding it to the density of the particles that form the pair. It's important to note that in this pair list a particle cannot be paired with itself because the contribution to force densities for both pressure and viscosity resolve to zero when a particle is paired with itself (i.e. a waste of time). However the contribution does not resolve to zero for density, luckily this contribution can be pre-computed. You can just resolve the density contribution when r is equal to zero and use this as the initial value for density for each particle at the start of each frame. Now that density has been determined for each particle we can solve the acceleration for each particle due to pressure and viscosity. By iterating through the pair list again we work out the acceleration contribution for each pair using the formulas shown in section 2.2.4 and add it to $particle_i$'s acceleration and subtract it from $particle_j$'s acceleration. Now you can just update the particle's velocity with the acceleration (not forgetting to add gravity) and it's position with the velocity using the time step for that frame. At this stage collision detection and response can be applied for each particle. I found that the most suitable collision response in order to maintain a stable simulation is to push them out of static geometry (the container) and reflect the velocity component perpendicular to the normal at the point of collision, with dampening of course. Control point SPH was implemented in the manner described in the previous section with a grid width of half the smoothing length. Both implementation's used a cylindrical container to hold the fluid particles. This container could be moved in a 2-dimensional plane in order to allow interaction with the fluid and observation of it's behaviour. A 6 degrees of freedom camera allows the fluid to be viewed from any angle In terms of speed the regular SPH implementation runs at approx. 85 frames per second while the control point SPH implementation runs at approx. 60 frames per second. This

made up for however by the volume that the control point SPH can simulate with the same level of detail at the surface as the regular SPH implementation. The volume appears to be twice that of the regular SPH implementation for the control point SPH implementation as shown in figure 4.1. By experimentation I found that 5000 particles will make up approximately the same amount of volume if used in a regular SPH implementation with the same parameters for k and density zero i.e. the same level of surface detail. With 5000 particles the frame rate drops to approx. 25 frames per second in the regular SPH implementation. It's clear that the reduced compressibility exhibited by control point SPH is a desirable feature in a real-time fluid simulation. Unfortunately despite using a fine grid (there is an average of approx. 2100 active nodes every frame) in areas where the particles are tightly packed there are some artifacts and instability in the control point SPH implementation. Occasionally this will effect the particles at the surface but generally these will be held in a pseudo equilibrium. For the regular SPH implementation there is no real instability issues due to the symmetrical forces but occasional the fluid will "pop" due to a the addition of a new stratum of particles, from the compressed particles at the bottom of the container. However this only happens as the fluid finds it's equilibrium at rest. Both implementations perform quite well while being interacted with, they can be swirled and sloshed convincingly with no real difference in performance visually. The figures 4.2 and 4.3 show the fluid being interacted with but this can only be fully appreciated when viewed in motion. All figures feature the control point SPH on the left and regular SPH on the right.

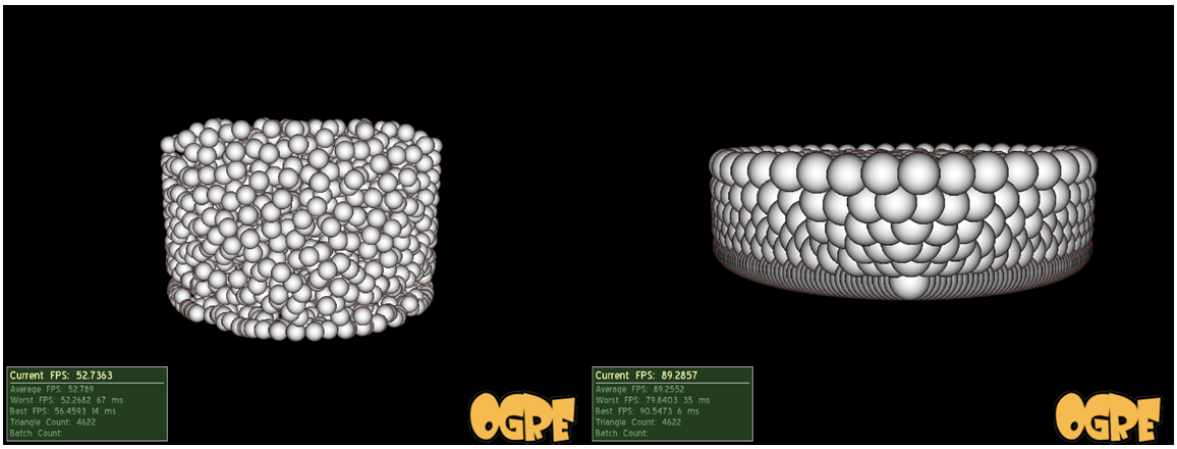

Figure 4.1: The comparative size of the two fluid volumes.

Figure 4.2: A comparison of the two fluid simulators when the container is being sloshed from the right to the left.

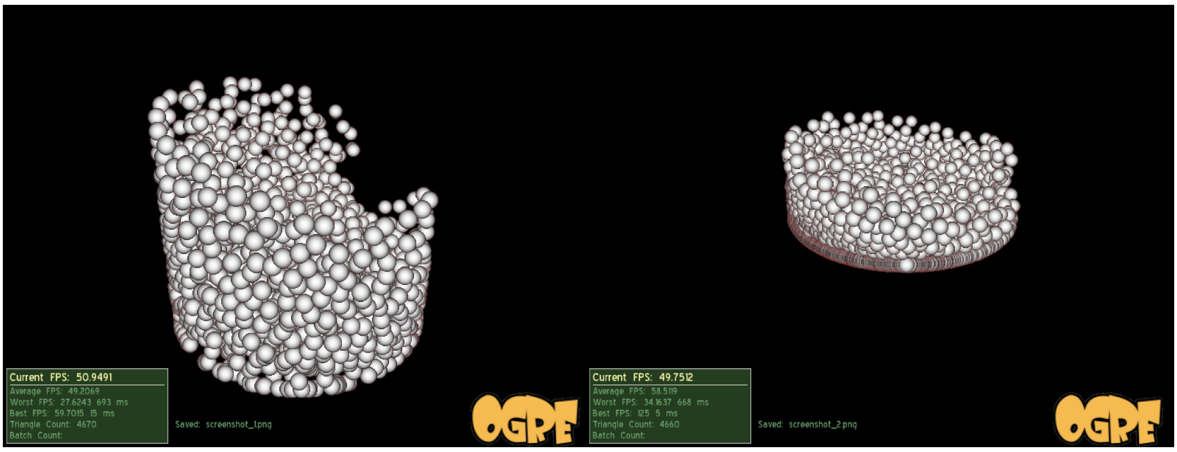

Figure 4.3: A comparison of the two fluid simulators when the container is being swirled in a clockwise direction.

4.2 Conclusions and Future Work

Control point based SPH has some benefits such as the linear distribution of density but proves unstable unless a very high resolution grid is used. This would mean that it would become significantly slower than regular SPH. For this reason I believe that control point SPH is not a great alternative on its own to an efficient SPH implementation. This due to the lack of symmetry for the forces exerted on particles within a given cell. One possible benefit control point SPH may have for fluid simulation is

if it were used to interpolate the density and forces for only the particles outside of a grid cell. If a cells maximum axis is the same length as the smoothing distance of the kernels then we can say for certain that every particle within the cell is within the smoothing length of every other particle in the cell. Regular pair-based SPH forces can be efficiently resolved for the particles within a single cell. A low resolution Control Point SPH will resolve the forces from particles outside the cell. This will reduce the size of the pair list for regular SPH dramatically as well as reduce the amount of time spent looking for pairs as it completely eliminates the need to search for pairs outside a single cell compared to searching up to 27 cells using regular methods. Obviously there would be some computational expense for running the relatively low resolution Control Point SPH .But this method may run just as quickly as regular SPH with the same number of particles due to the reduction in pairs and the efficiency with which pairs can be found. This methodology should hopefully resolve symmetrical forces and reduce the compressibility of the fluid particles. For my dissertation I had hoped to use screen space rendering methods run entirely on the GPU to render the fluid-air interface. I researched this methodology paying particular attention to the paper "Screen Space Meshes" by Muller [2] as discussed in section 2.3. The algorithm I proposed is as follows. Render to texture a depth map of the iso-surface of the air-fluid interface using billboards set to the particles' positions. These billboards will have depth maps of a hemi-sphere as a texture to tweak the pixel's depth according to it's position on the sphere's "face". In the pixel shader the depth value sampled from the texture must also be projected into screen space before it is used to create the final depth value that is stored at that pixel in the render target. In a second rendering pass another pixel shader program will smooth these depth-values as described in Muller's paper and render a new texture. In a third rendering pass yet another pixel shader program that has the inverse view and projection matrix passed to it will enable the GPU to render a normal for each pixel to a third texture by sampling from the surrounding pixels in the smoothed depth map. When rendering to screen we can then use the normal map texture and smoothed depth map texture with a pixel shader program to recreate the ISO surface of the fluid and allow us to apply lighting reflections and refractions etc. . I intend to finish implementing this algorithm in the coming weeks in order to see if there are noticeable difference between control point SPH and regular SPH when they are both rendered with an iso-surface.

Bibliography

[1] J. J. Monaghan, "Smoothed particle hydrodynamics," *Reports on Progress in Physics*, vol. 68, no. 8, pp. 1703–1759, 2005.

[2] M. Müller, S. Schirm, and S. Duthaler, "Screen space meshes," in *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (Aire-la-Ville, Switzerland, Switzerland), pp. 9–15, Eurographics Association, 2007.

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

[4] Wikipedia, "Navierstokes equations — wikipedia, the free encyclopedia," 2008. [Online; accessed 21-August-2008].

[5] N. Foster and R. Fedkiw, "Practical animation of liquids," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 23–30, ACM, 2001.

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

[7] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas, "Adaptively sampled particle fluids," *ACM Trans. Graph.*, vol. 26, no. 3, p. 48, 2007.

[8] F. Losasso, J. Talton, N. Kwatra, and R. Fedkiw, "Two-way coupled sph and particle level set fluid simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 4, pp. 797–804, 2008.

[9] F. H. Harlow and J. E. Welch, "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *Physics of Fluids*, vol. 8, no. 12, pp. 2182–2189, 1965.

[10] N. Foster and D. Metaxas, "Practical animation of liquids," in *Graphical Models and Image Processing*, pp. 23–30, 1996.

[11] A. J. Chorin, "Numerical solution of the navier-stokes equations," *Mathematics of Computation*, vol. 22, no. 104, pp. 745–762, 1968.

[12] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," tech. rep., Pittsburgh, PA, USA, 1994.

[13] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, "A hybrid particle level set method for improved interface capturing," *J. Comput. Phys*, vol. 183, pp. 83–116, 2002.

[14] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics - theory and application to non-spherical stars," *Mon. Not. Roy. Astron. Soc.*, vol. 181, pp. 375–389, November 1977.

[15] J. Stam and E. Fiume, "Depicting fire and other gaseous phenomena using diffusion processes," pp. 129–136, 1995.

[16] M. Desbrun and M. paule Gascuel, "Smoothed particles: A new paradigm for animating highly deformable bodies," in *In Computer Animation and Simulation 96 (Proceedings of EG Workshop on Animation and Simulation*, pp. 61–76, Springer-Verlag, 1996.

[17] A. Takashi, *Game Programming Gems 6: Chapter 16 Real-time Fluid simulation (Book & CD-ROM) (Game Development Series).* Rockland, MA, USA: Charles River Media, Inc., 2006.

[18] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, 1987.

[19] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Surface splatting," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 371–378, ACM, 2001.

[20] Wikipedia, "Bicubic interpolation — wikipedia, the free encyclopedia," 2008. [Online; accessed 10-September-2008].