VIEW DEPENDENT FLUID DYNAMICS

A Thesis

by

BRIAN ARTHUR BARRAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2006

Major Subject: Visualization Sciences

VIEW DEPENDENT FLUID DYNAMICS

A Thesis

by

BRIAN ARTHUR BARRAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Donald House
Committee Members,	Vinod Srinivasan
	John Keyser
Head of Department,	Mardelle Shepley

May 2006

Major Subject: Visualization Sciences

ABSTRACT

View Dependent Fluid Dynamics. (May 2006)

Brian Arthur Barran, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Donald House

This thesis presents a method for simulating fluids on a view dependent grid structure to exploit level-of-detail with distance to the viewer. Current computer graphics techniques, such as the Stable Fluid and Particle Level Set methods, are modified to support a non-uniform simulation grid. In addition, infinite fluid boundary conditions are introduced that allow fluid to flow freely into or out of the simulation domain to achieve the effect of large, boundary free bodies of fluid. Finally, a physically based rendering method known as photon mapping is used in conjunction with ray tracing to generate realistic images of water with caustics. These methods were implemented as a C++ application framework capable of simulating and rendering fluid in a variety of user-defined coordinate systems.

To my family, friends, and the Texas A&M Visualization Laboratory

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Computational Fluid Dynamics simulations of water, smoke, fire, and other natural phenomena are quite popular in films and games and these industries have increasingly embraced their use. The results are highly believable and the techniques and software used to create fluid effects are becoming more available for widespread use. In most films using fluid simulation effects, these effects are confined to glasses of water, river beds, goldfish bowls or other environments with fixed boundaries. There has been no published research in computer graphics and few films explore the dynamic simulation of large, effectively boundary-less, bodies of fluid.

An exception to this rule, the recent film *The Day After Tomorrow* used dramatic simulations of water interacting with a large-scale city environment, but the computational cost was extremely great. This is at least party because the fluid simulation technology employed a uniform simulation grid, yielding a constant density of detail in the fluid simulation regardless of how it affected the final image.

Another impediment to the simulation of large-scale fluid effects are the kinds of boundary conditions common in current techniques. In a goldfish bowl, fluid simulation within fixed boundaries is appropriate, and wave reflections off of these boundaries are consistent with reality. However, in a large-scale environment like the open ocean there are no walls and wave motion should be free to move into or out of the simulation without reflections.

To address the problems inherent in simulating large bodies of fluid, this thesis proposes extending computer graphics fluid simulation techniques in two ways. The first ex-

---

The journal model is *IEEE Transactions on Visualization and Computer Graphics.*

tension supports view-dependent level-of-detail simulation. View dependence presents a natural solution to the efficiency problem by simulating large bodies of fluid with decreasing detail as distance from the camera increases, thus providing constant screen-space detail across the simulation. The second extension replaces simulation boundary walls with boundaries that allow waves to pass freely, acting as sources or sinks to maintain fluid level within the simulation volume.

The conceptual difference between traditional computer graphics fluid dynamics and view dependent fluid dynamics is visualized below. Figure 1 shows how a traditional CG fluid simulation might be setup for rendering while Figure 2 shows a view dependent simulation setup. Note that the camera in the view dependent setup can vary its position slightly while still maintaining a close 1:1 simulation cell to pixel ratio.

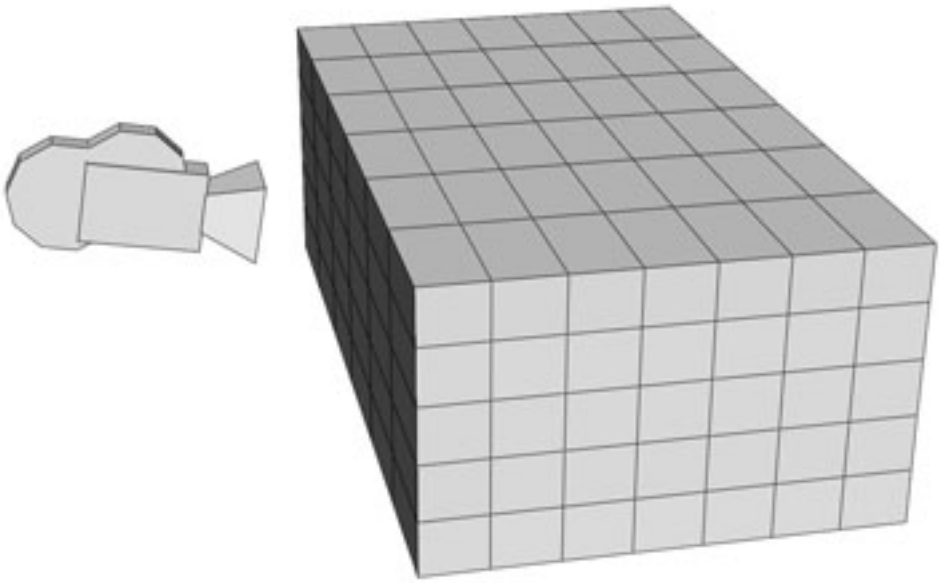

Fig. 1. Traditional CG fluid simulation setup.

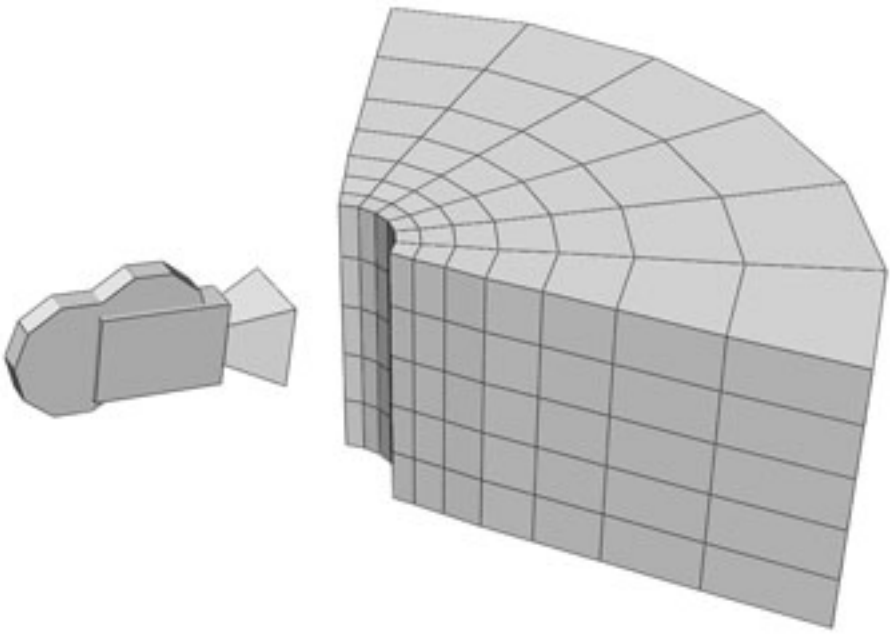

Fig. 2. View dependent simulation setup.

CHAPTER II

PRIOR WORK

Thompson et al. authored a book titled *Numerical Grid Generation* that describes general boundary-conforming curvilinear coordinate systems for use in grid generation and numerical applications. He describes how grids are generated using boundary segments that are split up equally in coordinate dimensions and extended throughout the physical domain. Because the mapping is to a rectangular transformation grid, the usual numerical techniques may be applied on this transformed grid. The additional effort required for the correct physical solution is to develop operators such as divergence, gradient, curl, laplacian, as well as arc, surface, or volume integrals on the transformed grid with meaning in the physical domain. Operators respecting time would also allow the grid structure to move throughout the simulation and conform to aspects of the flow [23].

Most computer graphics CFD methods follow the structure developed by Foster and Metaxas [7] who introduced the marker and cell (MAC) grid method developed by Harlow and Welch [10] within the context of a finite difference, Eulerian solver for incompressible fluid.

Stam [21] introduced a semi-Lagrangian method for dealing with instabilities inherent in the Foster-Metaxas approach, allowing larger time steps and higher viscosities. This has become the backbone for computer graphics fluid simulation because of its stability, adaptability, and ease of use. It uses a velocity transport step to enforce conservation of momentum, then applies external and diffusion forces to update the velocity field. Finally, this tentative solution is projected onto a divergence free solution by solving a Poisson equation arising from a Helmholtz-Hodge Decomposition of the velocity field. As a byproduct of this projection step the pressure is determined. Coupled with vorticity confinement [8], this

method can be used to produce interesting swirling smoke effects, but does not deal with the problem of fluid surfaces.

In two papers, Foster, along with Fedkiw's group developed the *Particle Level Set Method* to track the liquid surface interface [6, 5], giving a suitable smooth surface that captures fine detail for simulation and rendering. Coupling the advantages of an Eulerian grid-based approach, this method utilizes an implicit surface representing the air-liquid interface that is advected using the underlying velocities coming from the fluid simulation. Massless marker particles initially positioned around the surface are tracked and used to reconstruct fine detail in the implicit surface that would otherwise be lost to numerical error.

The semi-Lagrangian technique has also been applied to recreate viscoelastic effects such as goop and jelly [9]. Most recently, choreographing the animation of fluid has been accomplished using the adjoint method [15] and breaking waves have been animated using a volume-of-fluid based technique [14].

One limitation of these techniques is the high demand on computational resources required for high-detail 3D fluid simulation. To address this, Losasso et al. [13] developed a level-of-detail approach using an unrestricted octree method to simulate water and smoke. Their results allow extremely high detail in areas of interest by adaptive subdivision of the simulation. Departing from this approach, this thesis explores the use of a non-Cartesian, stable fluid method solver to achieve view-dependent level-of-detail.

CHAPTER III

METHODOLOGY

III.1. Coordinate Transformations

A Cartesian coordinate system is not always an optimal fit to some dynamics problems, such as flow around an airfoil or through a pipe. In these cases it is useful to define a new coordinate system that fits the problem at hand, mapping it to a regular grid for the purpose of computation. Anderson [1] describes a coordinate system transformation that can be used to map the semi-Lagrangian, stable fluid method from Cartesian coordinates to any regular, orthogonal coordinate system.

There are two principles that motivate the coordinate transformation. The first is the concept of transformed variables. To solve the equations for fluid flow using transformed variables, for example, values respective to a cylindrical coordinate grid, we must transform the governing equations (i.e. the Navier-Stokes momentum and continuity equations) to use these new variables. This transformation introduces new terms to the Navier-Stokes equations. These terms are known for a small set of standard coordinate systems that have previously been derived. The second concept is that of transformed coordinates. Here, the variables such as velocity, pressure, and density are all defined in physical space but are located on a transformed grid in physical space. In this case the governing equations do not need to be transformed, only the numerical operators need to be redefined. Using a coordinate transformation there are an infinite number of coordinate systems available so long as each can be analytically or numerically expressed.

For a view dependent simulation of a large body of water, a cylindrical coordinate system with the camera positioned on the central axis is a natural fit. In this coordinate system, a grid is built to resemble the camera view frustum. The computational grid con-

structed provides fine detail close to the viewing position and geometrically reduces detail with distance from the viewer.

The primary question is which method, transformed variables or transformed coordinates, would be the best fit for view dependent simulations of large bodies of water? To more readily access the current body of research in computer graphics and industry experience with current methods a transformed coordinates approach is used in this thesis. In the remainder of this section the transformed coordinates method as well as grid generation methods are examined for applicability in the view dependent fluid dynamics model.

III.1.1. Transformed Coordinates

In order to redefine the fluid simulation in transformed coordinates, we need to construct the appropriate coordinate transformations. All the dynamics problems are solved in this new computational space then mapped back to Cartesian space when needed. The analysis below describes this method in 2D coordinates, however it easily generalizes to 3D. Figure 3 shows an example of a physical space grid compared to a computational space grid.

There are two approaches to the coordinate transformation: a direct transformation and an inverse transformation. Given a regular, orthogonal grid defined in Cartesian space the direct approach requires a set of one-to-one functions that map Cartesian coordinates to a rectangular computational domain:

$$\xi = \xi(x, y),$$

$$\eta = \eta(x, y).$$

If we have a field $u(\xi, \eta)$ defined in the computational space, applying the chain rule

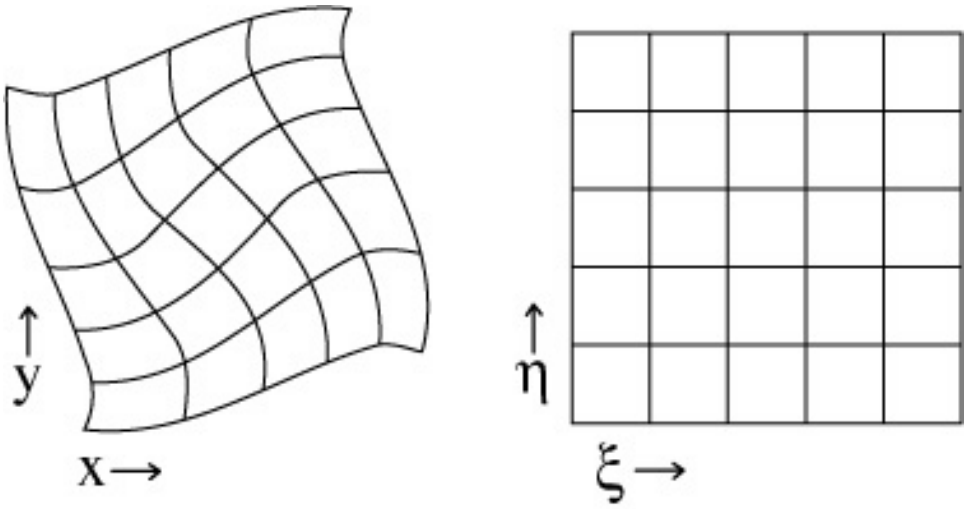

Fig. 3. Physical space grid vs. computational space grid.

yields

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial x},$$
$$\frac{\partial u}{\partial y} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial y},$$

which provides Cartesian partial derivatives for u in terms of the transformation. Second order derivatives may be found similarly.

However, it is not always possible to find a forward map whose partial derivatives are in convenient form. In this case, it might be desired to begin with the inverse map

$$x = x(\xi, \eta),$$
$$y = y(\xi, \eta),$$

which yields first order partial derivatives

$$\frac{\partial u}{\partial x} = \frac{1}{|J|}\left(\frac{\partial u}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial u}{\partial \eta}\frac{\partial y}{\partial \xi}\right),$$
$$\frac{\partial u}{\partial y} = \frac{1}{|J|}\left(\frac{\partial u}{\partial \eta}\frac{\partial x}{\partial \xi} - \frac{\partial u}{\partial \xi}\frac{\partial x}{\partial \eta}\right).$$

Here $|J|$ is the determinant of the Jacobian matrix J of the partial derivatives of the inverse mapping functions. In practice, this transformation is required whenever we need to make use of computational values in Cartesian space, such as when determining normal vectors for rendering.

III.1.2. Grid Generation

For view dependent fluid simulation we define the computational space using a modified cylindrical coordinate system (r, θ, h). Then the inverse map back to Cartesian coordinates is

$$x(r, \theta, h) = r\cos\theta,$$
$$y(r, \theta, h) = r\sin\theta,$$
$$z(r, \theta, h) = h,$$

where we take the x-y plane to be parallel to the ground plane, and z to be height.

To construct a computational grid on which to build our solver, we assume that the viewer is at the origin of the cylindrical system, looking down some radial axis. View depth then corresponds with radial coordinate r, viewing direction corresponds with angle θ, and elevation with h. In constructing the gird, we keep the proportions of each cell consistent to avoid grid bias, by maintaining a constant ratio of the depth of each cell to the arc length produced by sweeping its angular increment. The height of a cell can simply be kept constant. This produces a convenient, geometrically increasing function to define

our view dependent grid. Numbering from $n = 0$, assume that the center of the first grid cell is at radial distance r_0 from the origin, and that each cell's angular dimension is $\Delta\theta$. If the cell is to have uniform spatial dimensions (i.e. be the equivalent of a square), then the radial depth of the cell should be $r_0\Delta\theta$. If we want all cells to have these proportions, we arrive at the system of difference equations

$$\begin{aligned} r(0) &= r_0, \\ r(n+1) &= \left(1 + \frac{r_0\Delta\theta}{r_0}\right) r(n), \end{aligned}$$

for the radial coordinate of center of cell n from the origin. Thus, in closed form

$$r(n) = r_0(1 + \Delta\theta)^n.$$

Figure 4 shows a small sample of a 2D computational grid produced using this method, together with the relationship between grid array indices (i, j) and coordinates (r, θ). The polar coordinates of the center of cell (i, j) are simply $(r(i), j\Delta\theta)$. Edges of the cell correspond with array indices ± 0.5. Following the staggered grid approach to fluid representation, we keep velocity components separately in the center of each cell face, and pressure at the center of each cell.

Note the placement and direction of the velocity components with respect to the grid. The radial component of the velocity u is parallel to the normal of the bottom cell face while the angular component of the velocity v is parallel to the normal of the right cell face.

III.2. Fluid Dynamics

To solve the view dependent dynamics problem we modify the commonly used "stable fluid" method introduced to the computer graphics community by Stam [21]. This method

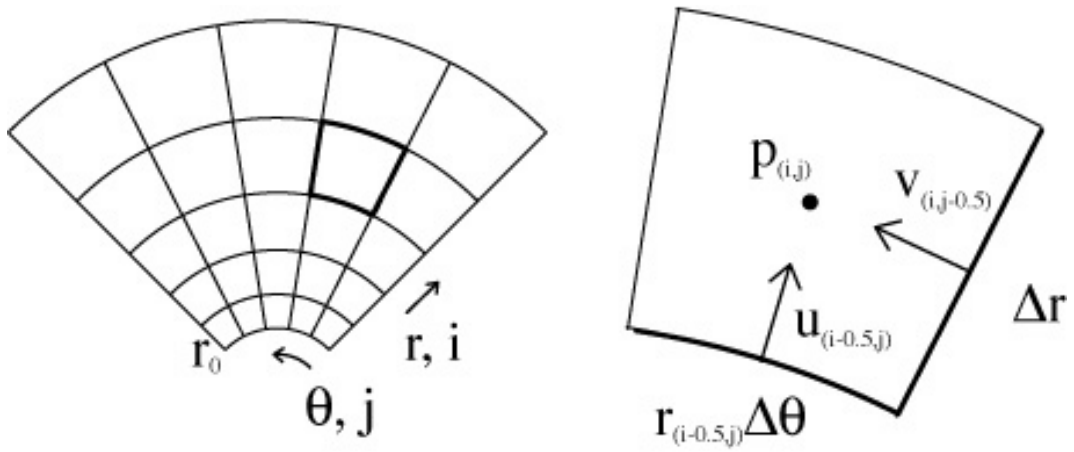

Fig. 4. View-dependent polar computational grid.

solves the Navier-Stokes equations for incompressible flow

$$\dot{\mathbf{u}} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \frac{\eta}{\rho}\nabla^2 \mathbf{u} + \mathbf{f}$$

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

that determine the time rate of change of fluid velocity $\mathbf{u}$, and the pressure p given fixed fluid density ρ, fluid viscosity η, and externally applied forces $\mathbf{f}$. The first equation accounts for conservation of momentum, pressure gradients, diffusion, and forces such as gravity. The second equation imposes the constraint that, since the fluid is incompressible, the divergence of fluid velocity must vanish everywhere.

This section proceeds as follows: first, the stable fluid method is described. Second, we detail the changes required to complete the divergence calculation on the view dependent grid. Lastly, we detail a method for computing the pressure and velocity correction terms used in the stable fluid method.

III.2.1. Stable Fluid Method

The stable fluid method is a semi-implicit method introduced by Stam that uses a semi-Lagrangian advection scheme and an implicit projection method to guarantee unconditional stability and divergence free flow. It was created as an alternative to the less stable iterative scheme introduced in [7] that had difficulties with viscous fluids and was unstable with large timesteps.

The method is outlined as follows. First, a tentative velocity field $\mathbf{W}^0$ is created using the velocities from the previous timestep $\mathbf{U}_n$.

$$\mathbf{W}^0 = \mathbf{U}_n$$

Next, the external and body forces are applied using Euler integration to create the second intermediate velocity field

$$\mathbf{W}^1 = \mathbf{W}^0 + \Delta t \mathbf{f}.$$

Then, to preserve momentum, a transport routine is used to carry fluid flow. This step satisfies the $-(\mathbf{u} \cdot \nabla)\mathbf{u}$ term found in the momentum equation. Starting at the center of each cell on the grid, Euler integration is used to integrate backwards through the velocity field one timestep. The velocity at this location is then taken as the velocity located at the center of the current cell. Because velocities are arranged on a staggered grid, these intermediate velocities must be averaged out to the cell faces to create the third intermediate velocity field $\mathbf{W}^2 = transport(\mathbf{W}^1, -\Delta t)$. As an alternative, two transport steps can be made instead, starting from the center of each cell face and sampling for the component of velocity found at that location. Note that the averaging step would not be required in this case. We chose to backtrace from the center of each grid cell to avoid doing two backtrace steps.

If the fluid is viscous (has $\eta > 0$) a diffusion term is required to account for viscosity forces. Because this second order term adds stiffness to the system, to avoid instability we use an implicit integration scheme $\mathbf{W}^3 = \mathbf{W}^2 + \frac{\Delta t \nu}{\rho}\nabla^2\mathbf{W}^3$ rather than an explicit Euler method. This forms the implicit equation, $(\mathbf{I} - \frac{\Delta t \nu}{\rho}\nabla^2)\mathbf{W}^3 = \mathbf{W}^2$, with equivalent matrix form which can be inverted using a sparse linear solver [22] to solve for $\mathbf{W}^3$. This implicit formulation allows very high viscosities without the need to reduce the timestep.

The final step in the stable fluid method is used to satisfy the continuity equation, i.e. to guarantee a divergence free velocity field. The idea is to decompose the velocity field $\mathbf{W}^3$ into a divergence free velocity field and the gradient of a scalar field q. The scalar field q is related to the fluid pressure using the pressure correction equation:

$$\mathbf{W}^4 = \mathbf{W}^3 - \frac{\Delta t}{\rho}\nabla p = \mathbf{W}^3 - \nabla q$$

If the gradient operator ∇ is applied to both sides to find divergence and because we require that $\nabla \cdot \mathbf{W}^4 = 0$, we can formulate a system of linear equations to solve for the fluid pressure:

$$\begin{aligned} \nabla \cdot \mathbf{W}^4 &= 0 \\ &= \nabla \cdot (\mathbf{W}^3 - \frac{\Delta t}{\rho}\nabla p) \\ &\Rightarrow \nabla \cdot \mathbf{W}^3 = \frac{\Delta t}{\rho}\nabla^2 p \end{aligned}$$

This system produces a sparse diagonal matrix which is inverted using a bi-conjugate gradient method to determine the pressures [22].

In summary, the stable fluid method for viscous fluids is established by solving the

following steps for each timestep Δt over the duration of the simulation:

$$\begin{aligned}
\mathbf{W}^0 &= \mathbf{U}_n \\
\mathbf{W}^1 &= \mathbf{W}^0 + \Delta t \mathbf{f} \\
\mathbf{W}^2 &= transport(\mathbf{W}^1, -\Delta t) \\
(\mathbf{I} - \frac{\Delta t \nu}{\rho} \nabla^2) \mathbf{W}^3 &= \mathbf{W}^2 \\
\frac{\Delta t}{\rho} \nabla^2 p &= \nabla \cdot \mathbf{W}^3 \\
\mathbf{W}^4 &= \mathbf{W}^3 - \frac{\Delta t}{\rho} \nabla p \\
\mathbf{U}_{n+1} &= \mathbf{W}^4
\end{aligned}$$

III.2.2. Divergence Calculation

Computing the divergence of the velocity field in view dependent coordinates requires special treatment. We compute divergence similar to [13] which referenced the second vector form of Green's theorem to compute divergence proportional to the volume of the cell as the sum of flux through each face. This can be conveniently written as:

$$V_{cell} \nabla \cdot \mathbf{u} = \sum_{faces} (\mathbf{u}_{face} \cdot \mathbf{n}) A_{face}$$

Figure 5 illustrates the concept using the coordinate normals centered on each face and the respective velocity components. The minus signs denote outward facing normals.

III.2.3. Pressure Computation and Velocity Correction

As mentioned in the stable fluid method, computing the fluid pressures at each timestep allows us to correct the velocity field to guarantee zero divergence. The sparse system of linear equations used to solve for the pressures is constructed by taking the Laplacian of

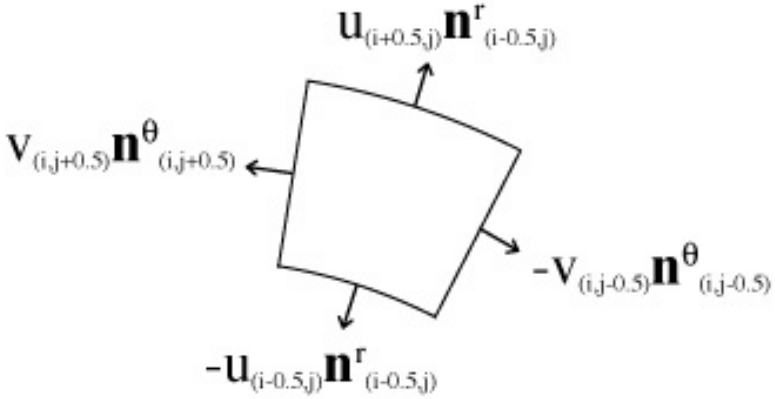

Fig. 5. Velocity components and cell face normals used to calculate divergence.

the pressure field. The Laplacian operator ∇^2 can be thought of as "the divergence of a gradient," and therefore we compute the elements of the matrix as if we were computing the divergence of the pressure gradient [13]. Only the current cell and adjacent cells are required for each row of this system, leading to a sparse matrix with most non-zero terms near the diagonal.

III.2.4. Boundary Conditions

Fluid simulations require correct boundary conditions for simulation cells located along walls or other domain features. Two types of conditions are treated in this thesis. The first are *wall-fluid* boundaries and the second are *infinite-fluid*, meaning fluid simulated in the grid can splash outside or flow into the simulation domain.

Simple *wall-fluid* conditions are described in [7] and prevent fluid flow through walls by modifying the pressure Laplacian matrix required in the Stable-fluid method. By assuming the flux through wall cells is zero, the pressure gradient across these cells is zero and the corresponding terms can be removed from the linear system. If after every velocity

update the velocities along the wall boundaries are restricted to zero the divergence calculation proceeds as usual. This approach results in fluid that flows around obstacles and the simulation boundaries.

An added *infinite-fluid* boundary condition can be imposed on the simulation boundaries to emulate continuous fluid. Rather than constraining velocities along the boundaries to be zero, the pressures are set either statically or dynamically. In the static case, for example, the pressures can be computed by integrating down vertical shafts of fluid starting at the fluid surface with ambient (air) pressure. Letting the velocities change dynamically during simulation but constraining the pressures allows fluid to flow freely into or out of the simulation while maintaining a constant fluid height. Dynamically computing pressures using wave profiles, along with velocity constraints, could be used to generate waves or other interesting effects.

The *infinite-fluid* boundaries are used, for instance, to simulate un-contained fluid. Interesting flow inside the simulation domain is not interrupted when it crosses the boundary. Figure 6 illustrates this idea. Notice how the waves caused by the drop do not appear to be inside a box or container such as a glass.

III.3. Fluid Surface Calculation

In order to fully simulate fluid effects using the stable fluid method, a way of tracking the surface of the fluid is required. This allows us to treat the fluid-air boundary conditions correctly as well as provide a convenient surface to render or display. The *Particle Level Set* method was developed to be a solution to the problem [6, 5].

The remainder of this section describes the Particle Level Set (PLS) method as it applies to a water surface. The second and third sub-sections detail the changes to the PLS required so it functions on a view dependent simulation grid.

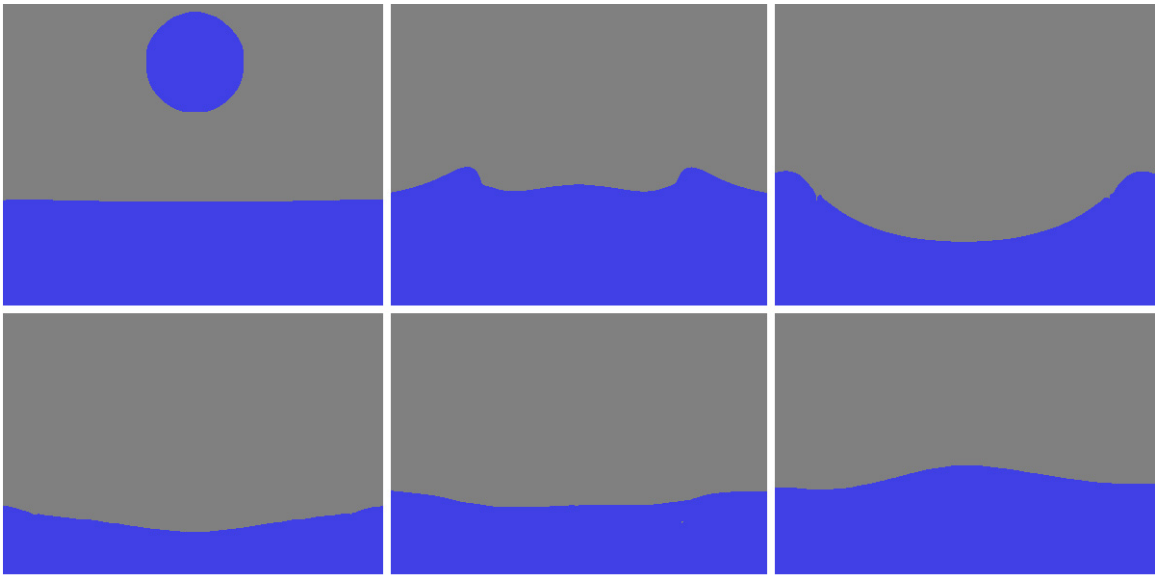

Fig. 6. An example of a fluid drop with infinite-fluid boundaries.

III.3.1. Particle Level Set Method

The PLS method is a ”thickened” front tracking method that wraps the fluid volume in an implicit surface called a *level set*. The level set is the zero set of a signed distance function ϕ, where negative values of ϕ are considered fluid and positive values considered air. The level set equation is

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = 0,$$

where $\frac{\partial \phi}{\partial t}$ is the time derivative, and $\mathbf{u} \cdot \nabla \phi$ being the component of fluid velocity in the direction along the gradient of the ϕ. This equation is integrated at each time step when new fluid velocities are computed to update ϕ.

Because numerical errors may smear out interesting details, massless fluid particles are positioned in a thin layer just inside and outside of the surface $\phi = 0$ and are used to correct the level set through the duration of the simulation. Particles placed inside the level set carry a negative sign and particles outside carry a positive sign. These particles

are initialized randomly around the surface within a radius r_p determined by the following switch

$$r_p = \left\{ \begin{array}{ccc} r_{max} & if & s_p\phi(\mathbf{x}_p) > r_{max} \\ s_p\phi(\mathbf{x}_p) & if & r_{min} \leq s_p\phi(\mathbf{x}_p) \leq r_{max} \\ r_{min} & if & s_p\phi(\mathbf{x}_p) < r_{min} \end{array} \right\}$$

with s_p being the sign of the particle, $\phi(\mathbf{x}_p)$ the signed distance to the surface, and r_{min} and r_{max} the minimum and maximum particle radius. These radii are usually 0.1 and 0.5 times the size of the grid cell dimension.

Once the level set has been advected, the particle positions are updated using the fluid velocity and are then used to check the level set for errors. If a particle is found to be across the interface (outside for negative particles and inside for positive particles) by a distance more than its radius, it is said to have *escaped*. A possible Level Set correction value for escaped particles is found using

$$\phi_p(\mathbf{x}) = s_p(r_p - |\mathbf{x} - \mathbf{x}_p|)$$

where $\mathbf{x}_p$ is the position of the particle and $\mathbf{x}$ is the position on the grid of the Level Set sample value. After all escaped particles and correction values are found, the minimum error correction value for the grid sample is used to replace the Level Set value. After the correction, the particle radii are adjusted using the switch above. Every 20 or 30 frames, areas of the surface with too many or too few particles are cleaned up, removing or adding new particles as the folding and tearing of the surface during fluid motion disturbs the distribution.

After the Level Set has been updated and corrected using the particles, it is re-initialized to a signed distance function and smoothed for rendering. Re-initialization is described in

the next section. To smooth the surface, a smoothed sign term

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + \Delta x^2}}$$

is first computed where Δx is the minimum grid cell dimension and computed for each cell individually. This term is used as a weight in the smoothing equation and approaches 1 or -1 further away from the surface. The smoothing equation

$$\frac{\partial \phi_{\tau+1}}{\partial t} = -S(\phi_{\tau=0})(|\nabla \phi_\tau| - 1)$$

is an iterative update in fictitious time used to even out local areas of the Level Set and reduce noise in the surface. Special treatment must be given to this process and it is suggested to use Godunov's method for computation [18].

III.3.2. Reinitialization

As the Level Set is updated using the Level Set equation, the signed distance property of ϕ may be destroyed. Once this happens, the methods above do not function properly and will introduce more errors. Therefore, after each timestep ϕ needs to be recomputed to have the signed distance property. The *Fast Marching Method* [2, 18] is a way to recompute the distance field both inside and outside the fluid.

The Fast Marching Method (FMM) begins at the fluid surface where interpolation is used to find approximately correct distance values for surface cells. These cells are labeled *Known* cells. Every neighbor of an *Known* cell is labeled as *Tentative*, and all other cells are labeled *Unknown*. The FMM loops as follows:

1. Let *Trial* be the position in *Tentative* with the smallest ϕ value. Label every neighbor of *Trial* that is *Unknown* as *Tentative*.

2. Recompute the values of ϕ for every neighbor of *Trial* that are *Tentative* by solving

the quadratic equation

$$\left(\frac{\phi_{i,j,k}-\phi_1}{\Delta x_i}\right)^2+\left(\frac{\phi_{i,j,k}-\phi_2}{\Delta x_j}\right)^2+\left(\frac{\phi_{i,j,k}-\phi_3}{\Delta x_k}\right)^2=1$$

where ϕ_1, ϕ_2, and ϕ_3 are the maximum known neighbor values in each dimension, $\Delta x_i, \Delta x_j$, and Δx_k are distances from the current cell and the neighbor value, and $\phi_{i,j,k}$ is the unknown value at the current cell. If a neighbor cell is not *Known*, its corresponding term must be removed from the equation as it cannot be used to determine the current ϕ value.

3. Change *Trial*'s label to *Known* and return to 1).

The FMM is considered a fast method because it's algorithmic complexity is *O(Nlog N)*. This is because every cell in the grid is touched once and if a heap is used to maintain a list of cells sorted on ϕ, this list can be updated in *O(log N)* time.

III.3.3. Extension Velocities

Because fluid particles are located outside the fluid surface, where physically acceptable velocity values are unknown, there needs to be a way of estimating how they will move with the surface of the fluid during the simulation. Rather than simulating both air and water, which involves a difficult two-phase fluid simulation involving more complicated fluid-air boundary conditions, the fluid velocity at the surface is used to derive acceptable values for the particles. These *extension velocities* can be computed in a similar fashion as the reinitialization algorithm by solving an upwind approximation to satisfy

$$\nabla F_{ext} \cdot \nabla \phi_{temp} = 0$$

where F_{ext} are the extension velocities and ϕ_{temp} is a temporary signed distance value created as in the FMM. This method is described in [2] but was not used here.

Instead, the signed distance field was first updated using the FMM. Then, velocities

were converted to physical space coordinates and placed at each node in the grid. Next, in a similar fashion to the FMM, cells were labeled and sorted using heapsort. Rather than calculating tentative distance values as in the FMM, weighted averages of *Known* cell velocities were used to update the extension velocity for the current *Tentative* cell. In this thesis, the signed distance field and extension velocities were calculated in two passes but could also be calculated together as the front advances.

III.4. Rendering and Shading

An important part of creating a believable fluid simulation for computer graphics includes how it is visually displayed. Many methods exist to render fluid-like surfaces. A common method is to extract the level set surface using an iso-surface extraction algorithm, such as the *Marching Cubes* algorithm, to generate a polygonal mesh [12]. This mesh is shaded and rendered similarly to other polygonal objects and also allows the surface at each frame to be saved to disk cheaply. One drawback, however, is that the surface polygons are not necessarily coherent between frames, and unless the mesh is highly tessellated, popping and snapping are visible.

Rather than extracting the surface as a mesh, the surface can be ray traced directly. This eliminates any tessellation issues and guarantees a clean render of the smoothed fluid surface. Refractivity and reflection are easily combined in the ray tracer to render transparent fluids as well. Ray tracing also allows more advanced rendering methods, such as photon mapping and environment lighting, to make the image more realistic. Photon mapping traces packets of light energy through the scene to simulate *global illumination* and *caustics* [11].

III.4.1. Ray Tracing the Level Set Surface

Integrating a level set object into a ray tracer involves extending the bounding object, commonly found in ray tracers to speed up complex model ray-collision tests, and implementing a root-finding algorithm. The bounding object for a regular, Cartesian grid based level set would, for example, be comprised of a rectangular axis-aligned box. For a view dependent Level Set, the bounding geometry might look like a slice of pie with a hole in the middle. Figure 7 illustrates this cylindrical bounding geometry setup. The ray collision table for this box includes 4 ray-plane and 2 ray-cylinder collision tests.

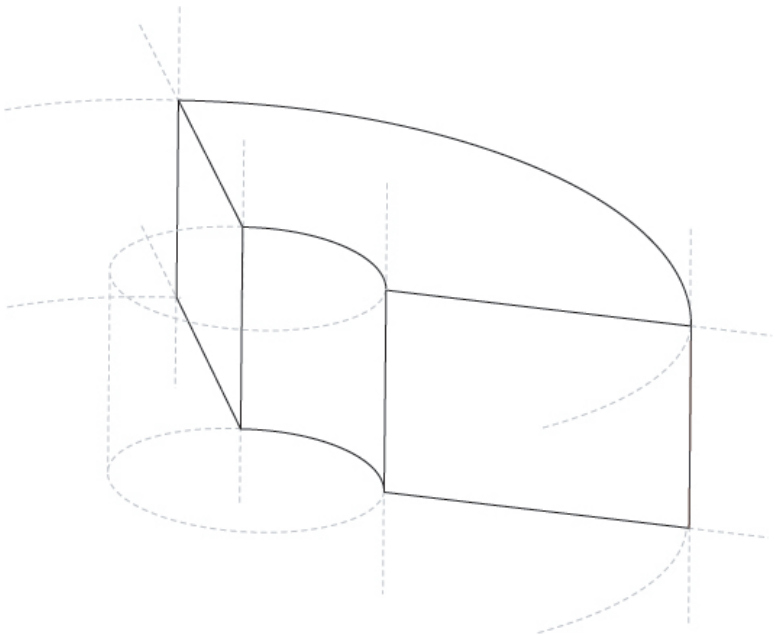

Fig. 7. Cylindrical bounding box setup.

Once a ray has been determined to hit the bounding geometry, the exact intersection point is determined. This is used as a starting position for an iterative, root finding search through the level set field. Remember that the level set field provides a signed distance value at every grid node and that this value is the minimum distance to the surface. Instead

of integrating the ray with a constant timestep, the magnitude of the distance value is used. Alternatively, the magnitude of $\phi(\nabla\phi \cdot \mathbf{r})$, where $\mathbf{r}$ is the ray direction, might be used as a more conservative timestep. If the sign of this distance value changes while integrating through the Level Set, then an intersection has occurred. At this point the ray is reset to the previous position and the timestep is halved. After a constant number of iterations in this manner, an acceptable intersection point at the surface has been found and the routine returns this position and a normal to the surface. Note that the normal is computed as the gradient of ϕ. If the ray is integrated outside of the bounding geometry, then there was no collision and the root finding routine returns no intersection.

If a ray hits a transparent or refractive surface (such as water) two recursive ray tracing steps proceed. The first ray is refracted through the surface and continues to trace through the Level Set, returning a color value when it finally hits an opaque object. A second ray is also calculated and shot to simulate the *Fresnel* effect. Some dielectric materials won't allow rays to pass through the surface due to a difference in the index of refraction of the two mediums. Therefore, at glancing angles the ray is reflected off the surface, similar to a mirror. Combining the colors of refracted and reflected rays gives believable results for water surfaces.

III.4.2. Rendering Using Photon Mapping

Photon Mapping [11] is a method that is used to simulate a type of realistic rendering known as *global illumination* with *caustics*. Global illumination is different from regular ray tracing in that it includes light reflecting from surrounding objects in addition to light coming directly from the light sources. The effect is quite pronounced. Figure 8 shows a scene that has been ray traced using simple forward ray tracing and the same scene ray traced including photon mapping. Notice how the colors from the wall bleed onto the

sphere with photon mapping enabled. Caustics, the beautiful patterns of light projected onto walls or the celiing, are caused by light refracting through ripples or waves on the water surface. Photon mapping simulates caustics as well because photons, when refracted through the surface, are accumulated in small areas and produce concentrated areas of light. Figure 9 illustrates this effect.

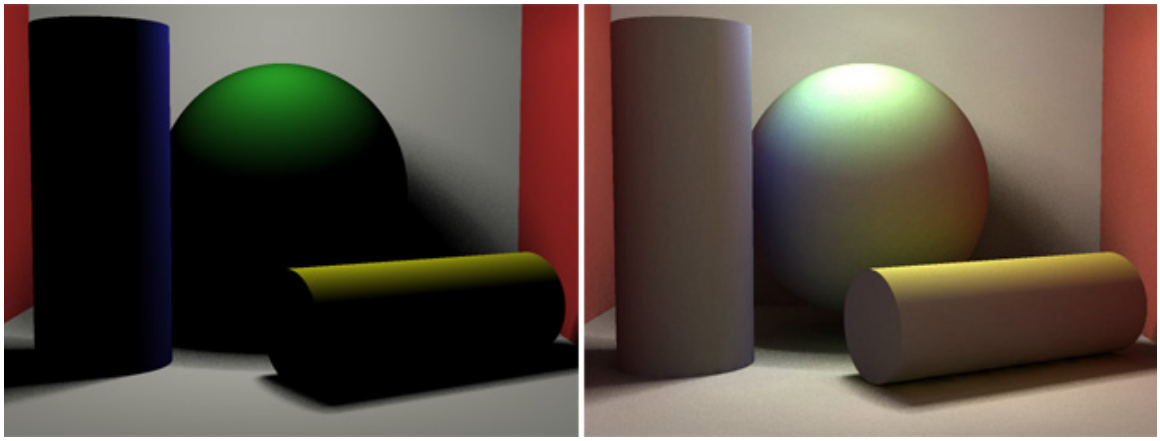

Fig. 8. A scene ray traced without and with global illumination.

Regular ray tracing is also known as *forward ray tracing* because rays are shot starting from the eye or camera position into the scene. When they intersect with geometry, the intersection point is lit by shooting more rays towards the lights and if visible, the material's lighting equation is solved with the current light's properties. Summing up all light equations gives the approximate color at that position. This differs from Photon mapping which uses a process known as *backward ray tracing* that shoots rays from light sources into the scene. The contributions of each light are stored in a data structure describing the scene and can be referenced when making the final image.

Photon mapping is implemented as a two step process. First, photons are traced from each light source into the scene. Every time they intersect an object they are stored into a *photon map*, and afterwards they are reflected and shot back into the scene. Again they are

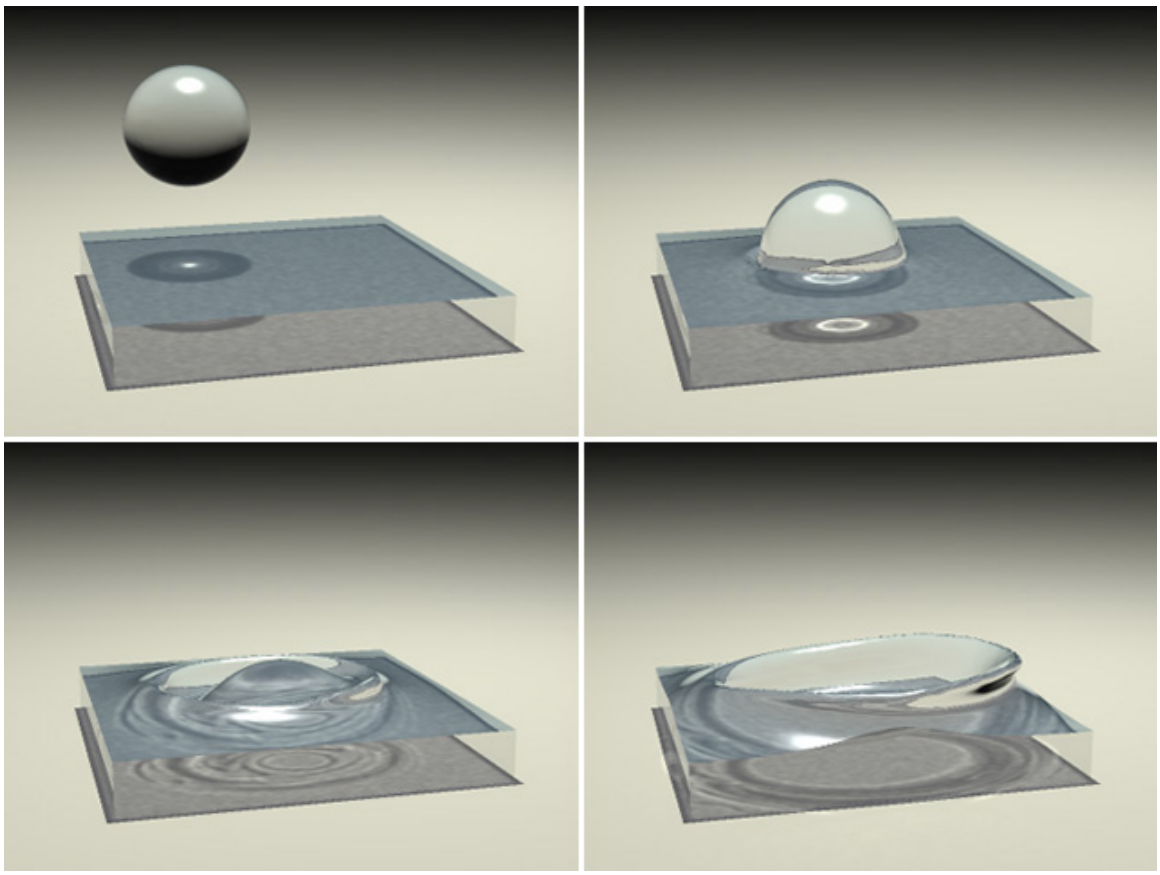

Fig. 9. A water simulation rendered with caustics using photon mapping.

stored in the photon map if they hit an object. This process repeats until either the photon has no more power or it fails to hit an object.

The next pass is regular forward ray tracing and is where the final image is created. When intersection points are being shaded the light equation includes an additional two terms that sample the photon map: one for global illumination and one for caustics. Caustic sampling is straightforward: search the photon map for photons within a radius of the intersection point and sum up the photon power. Filtering, for instance a tent or Gaussian filter, can alternatively be used to smooth the caustic samples but is not necessary.

The second term computes global illumination in a process known as *final gathering*. Instead of sampling the global photon map directly, a hemisphere of rays are shot from the point to be shaded and when an intersection occurs, the global photon map is sampled from that position. Computing a weighted average of these samples produces an estimate of the incoming radiance which is added to the lighting equation. Because this step is computationally expensive and due to the fact that it does not contribute much to the effects of water it can be ignored.

CHAPTER IV

IMPLEMENTATION AND RESULTS

IV.1. Implementation

The methods described in this thesis were implemented as a stand alone, Object Oriented C++ program. Because of the complexity of this project, a number of parent interfaces and child classes were used to simplify development, re-usability and testing. The program structure implements the standard stable-fluid method to simulate fluids such as gas and smoke as well as liquids such as water, in both Cartesian and view dependent coordinates. In addition, the structure exposes the functionality at multiple levels so it can be extended to support features required by other stable-fluid based methods such as explosions [4] and visco-elastic fluids [9].

IV.1.1. Class Hierarchy

The following important parent classes implement functionality that is shared across the program:

- *Grid* - stores required fluid components such as velocity, pressure and density in a standard 3D grid
- *Differencer* - executes numerical methods on *Grid* objects such as finite differencing, normal calculations and conversion between physical and computational space
- *FluidSolver* - implements the stable-fluid method and ability to track density along the flow

- *LevelSet::Grid* - an extension of the *Grid* to support Level Set methods such as reinitialization, smoothing and extension velocities
- *RayTracer* - a multi-threaded ray tracer used to render out the scene and fluid objects
- *Object* - ray-traceable object class as described in [19]

These parent classes were extended to support more specific functionality or methods. The child classes contain more specific code, for example, to implement differences in numerical methods, program flow or rendering options:

- *SmokeSolver::FluidSolver* - implements the smoke solver described in [8]
- *LiquidSolver::FluidSolver* - implements the liquid solver described in [5]
- *CartesianDifferencer::Differencer* - numerical methods on Cartesian based grids
- *PolarDifferencer::Differencer* - numerical methods on view dependent (more generally, polar based) grids
- *ParticleLevelSet::LevelSet* - an implementation of the *Particle Level Set Method* described in [5]
- *LevelSetObject::Object* - implements the root-finding algorithm to intersect a *Ray* (see below) with the surface
- *CylindricalLevelSetObject::LevelSetObject* - extension to the renderable *LevelSetObject* for view-dependent rendering

In addition to the above classes, the following utility classes were implemented to facilitate efficient development and compile-time optimizations:

- *Vector* - template 3D vector class with inline operators

- *Ray* - position and direction information used during ray tracing
- *PhotonMap* - an octree-based container used to store and search photons during rendering

IV.1.2. Program Flow

The final program flow for liquid simulations is listed below. Note that these pseudo-code listings are "flattened" versions from the Object Oriented structure in the actual implementation but still maintain the correct order.

SIMULATION$(SimulationTime, TimeStep)$

$\mathbf{U}_0, \phi_0, \mathbf{P}_0 \leftarrow$ INITIALIZE()

$n \leftarrow 0$

while $ElapsedTime < SimulationTime$

do $n = n + 1$

▷ Simulate

$\mathbf{U}_n \leftarrow$ UPDATEVELOCITY$(\mathbf{U}_{n-1}, TimeStep)$

$\phi_n, \mathbf{P}_n \leftarrow$ UPDATELEVELSET$(\mathbf{U}_n, \phi_{n-1}, \mathbf{P}_{n-1}, TimeStep)$

▷ Render the current frame

SHOOTPHOTONS$(Scene, \phi_n)$

RAYTRACE$(Scene, \phi_n)$

UPDATEVELOCITY($\mathbf{U}_{n-1}, \phi_{n-1}, TimeStep$)

▷ Use the Stable-Fluid method to update fluid velocity

$\mathbf{W}^0 \leftarrow \mathbf{U}_{n-1}$

$\mathbf{W}^1 \leftarrow$ FORCES($\mathbf{W}^0, TimeStep$)

$\mathbf{W}^2 \leftarrow$ TRANSPORT($\mathbf{W}^1, TimeStep$)

$\mathbf{W}^3 \leftarrow$ DIFFUSE($\mathbf{W}^2, Viscosity, TimeStep$)

$\mathbf{W}^4 \leftarrow$ PROJECT($\mathbf{W}^3, TimeStep$)

▷ Use the Level Set to provide reasonable velocities outside the fluid

$\mathbf{U}_n \leftarrow$ EXTENDVELOCITIES($\mathbf{W}^4, \phi_{n-1}$)

UPDATELEVELSET($\mathbf{U}_n, \phi_{n-1}, \mathbf{P}_{n-1}, TimeStep$)

▷ Advance and correct the Particle Level Set

$\phi_n, \mathbf{P}_n \leftarrow$ ADVANCE($\mathbf{U}_n, \phi_{n-1}, \mathbf{P}_{n-1}, TimeStep$)

$\phi_n \leftarrow$ CORRECT($\phi_n, \mathbf{P}_n$)

$\phi_n \leftarrow$ SMOOTH(ϕ_n)

$\phi_n \leftarrow$ REINITIALIZE(ϕ_n)

▷ Every 20 frames redistribute surface particles

if ($n \bmod 20$) $\mathbf{P}_n \leftarrow$ RESEEDPARTICLES($\mathbf{P}_n$)

IV.2. Results

The methods above were successfully implemented with mixed results. To explain the results the original goals of the thesis are reiterated and compared. The first goal was to simulate large areas of open water efficiently while the second goal was to decouple the simulation from fixed wall boundaries. Execution times were not a goal of this thesis. The main idea is to simulate more volume per grid cell and not to optimize the runtime code.

IV.2.1. Large Water Simulation

This thesis approached the problem of large water simulation by decreasing detail with distance from the camera (i.e. increasing fluid volume per simulated area). This enabled grid setups that encompass more volume per simulated unit than a uniform Cartesian grid. While Cartesian grids can be expanded to fill the same space, view dependent grids locate detail closer to the viewer.

To achieve this level-of-detail approach the simulation was ran on a modified polar coordinate grid with non-uniform cell sizes. An inverse coordinate transformation was then used to help modify the numerical methods in the stable fluid method so they may function on this new grid setup. Finally the particle level set method was adapted to support the new grid.

While simulations run in this new setup were stable (e.g. the solution converges and does not "explode"), noticeable artifacts appeared in the fluid motion. Figures 10 and 11 show how fluid appears to flow in the angular direction and is seen as concentric rings.

Upon inspection, these artifacts are caused by the final step in the stable fluid method – projection. During this step the fluid pressures are found by solving a system of linear equations relating the divergence of the pressure gradient to the fluid divergence. After the pressures are found, their gradient is subtracted from the fluid velocity to remove any divergent flow. While this step satisfies the incompressibility constraint, $\nabla \cdot \mathbf{u} = 0$, and therefore maintains fluid volume, it is not preserving momentum across cells with varying tangent spaces.

Alternative approaches to defeating this problem were unsuccessful. The first approach was to simulate fluid using derivatives determined by coordinate transformation on physical space vectors, rather than using the stable fluid method in computational space. This resulted in exactly the same results. The second approach was to apply a non-uniform

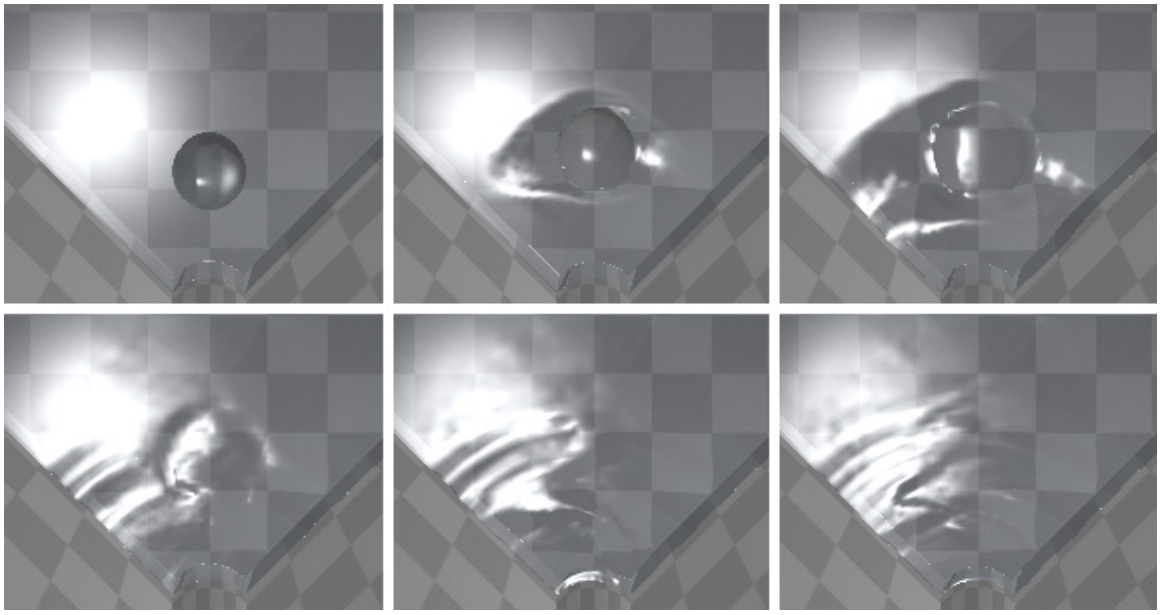

Fig. 10. Artifacts appear in the view dependent surface as seen from above.

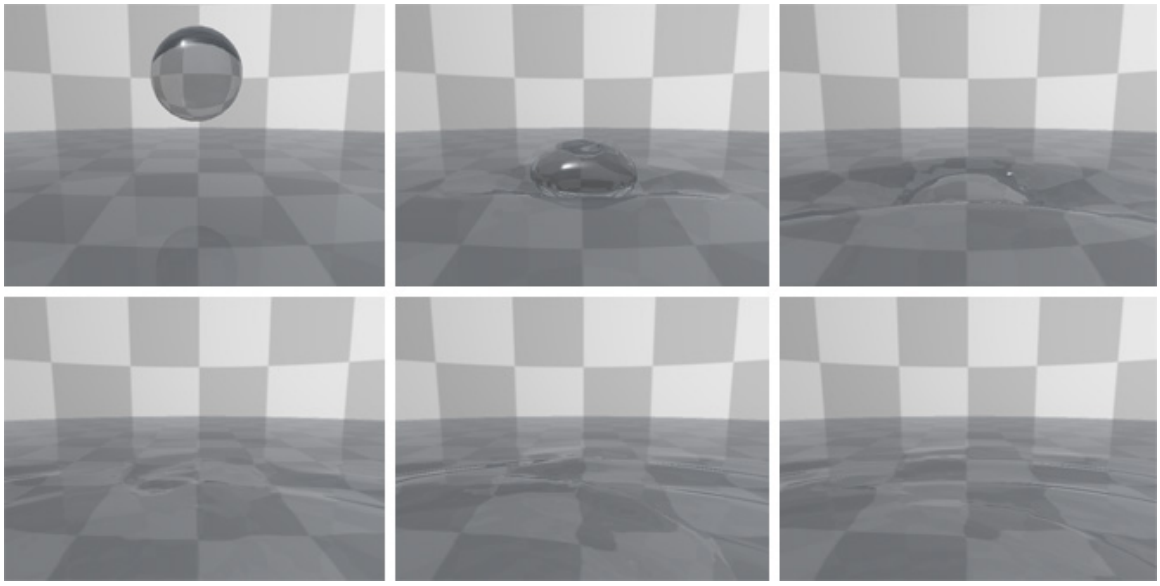

Fig. 11. Artifacts appear in the view dependent surface as seen from the viewer.

rotation to gradients used in the pressure matrix to solve for pressures in the same vector space. This approach seemed to work but was unstable and caused the solution to diverge ("explode"). Due to time constraints a stable solution to this problem was not found.

IV.2.2. Boundary Free Simulation

This thesis successfully implemented the *infinite-fluid* boundary conditions. These boundary conditions allow fluid to move freely into or out of the simulation domain without reflections. Moreover, they help maintain a constant level of fluid, such as deep ocean. Figures 12 and 13 help distinguish between *infinite-fluid* and *wall-fluid* boundaries. Notice that when the simulation is run with strict wall boundaries the fluid runs high up against the walls. While with infinite fluid boundaries, the fluid flows smoothly through the simulation boundary.

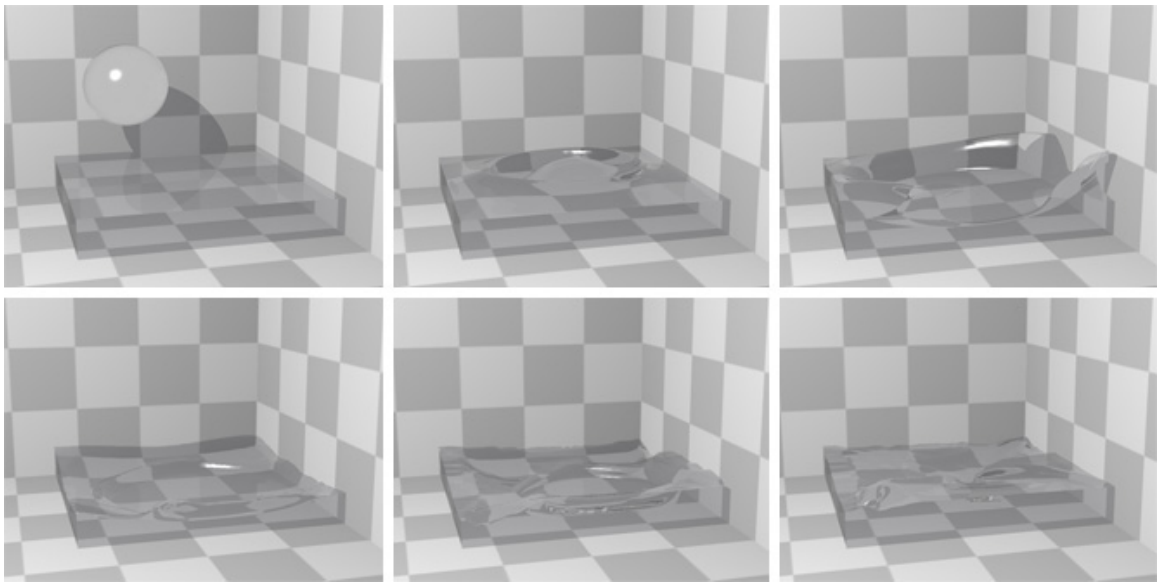

Fig. 12. A drop of water with *infinite-fluid* boundaries.

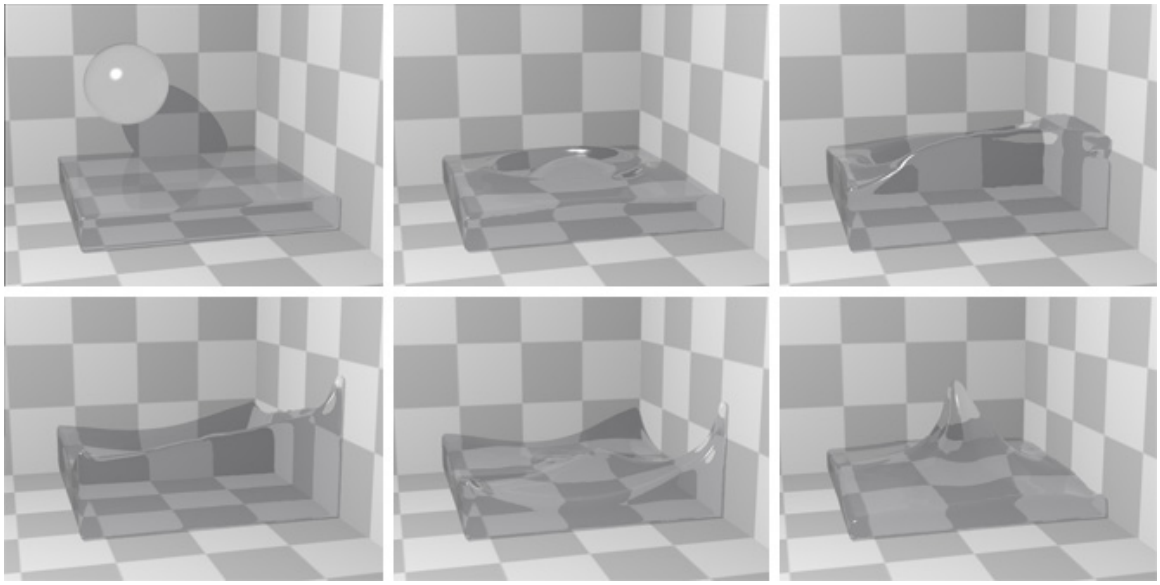

Fig. 13. A drop of water with *wall-fluid* boundaries.

IV.2.3. Final Simulations

Figures 14 and 15 show sequences of two animations produced by our methods. These were Cartesian grid-based simulations rendered with caustics using photon mapping. The simulation grid sizes were 80x60x80 cells and took approximately 24 hours to simulate and render. Rendering took the majority of the time as 3x3 super-sampling was enabled and the number of photons was about 500,000. With regards to running times, note that no hardware specific optimizations (such as maintaining cache coherence or instruction vectorizing) were implemented as these are beyond the scope of this thesis.

The next example simulation was done using the view dependent framework and is similar in setup to those above, with the exception being rendering. The eye and top views in Figure 16 show three fluid sources pouring into a wedge-shaped container.

This simulation took approximately 48 hours to simulate – twice as long as the same setup in Cartesian coordinates. Although a portion of this time included executing more

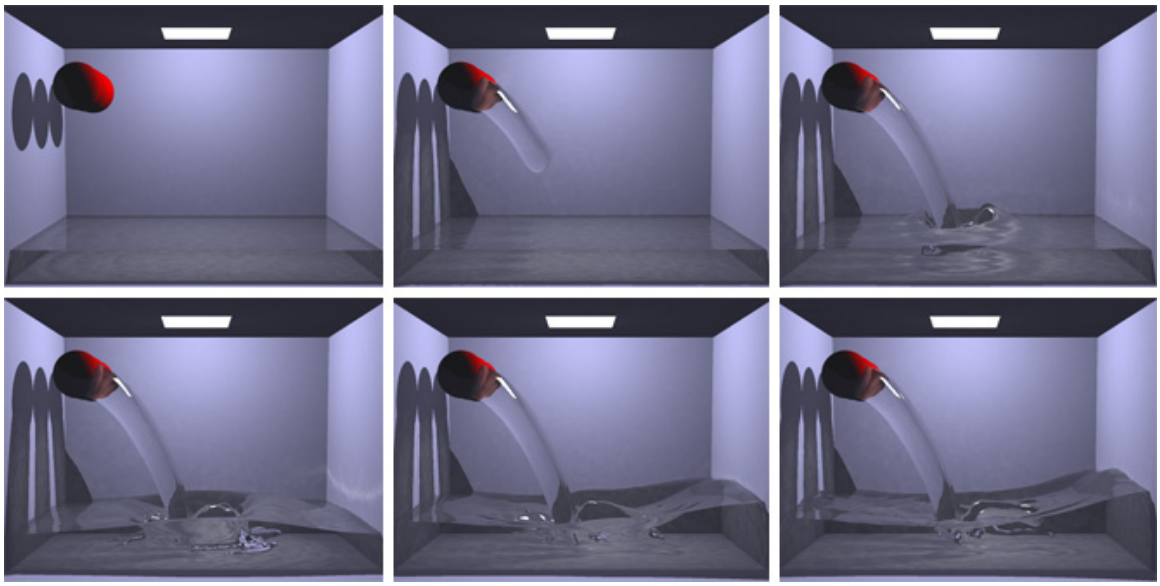

Fig. 14. A 80x60x80 water simulation rendered with caustics.

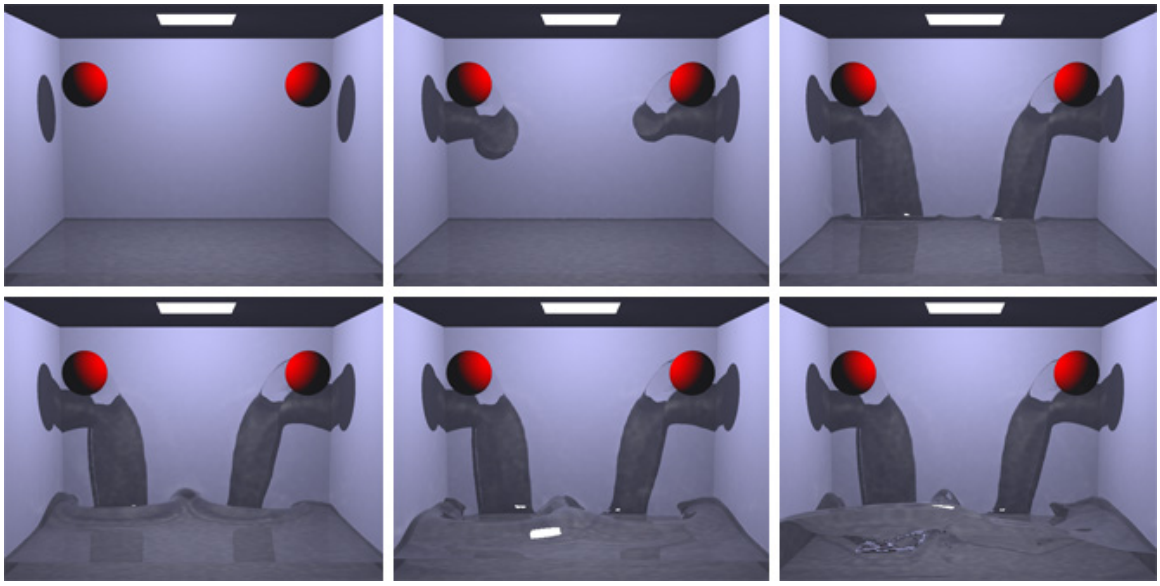

Fig. 15. Another 80x60x80 water simulation rendered with caustics.

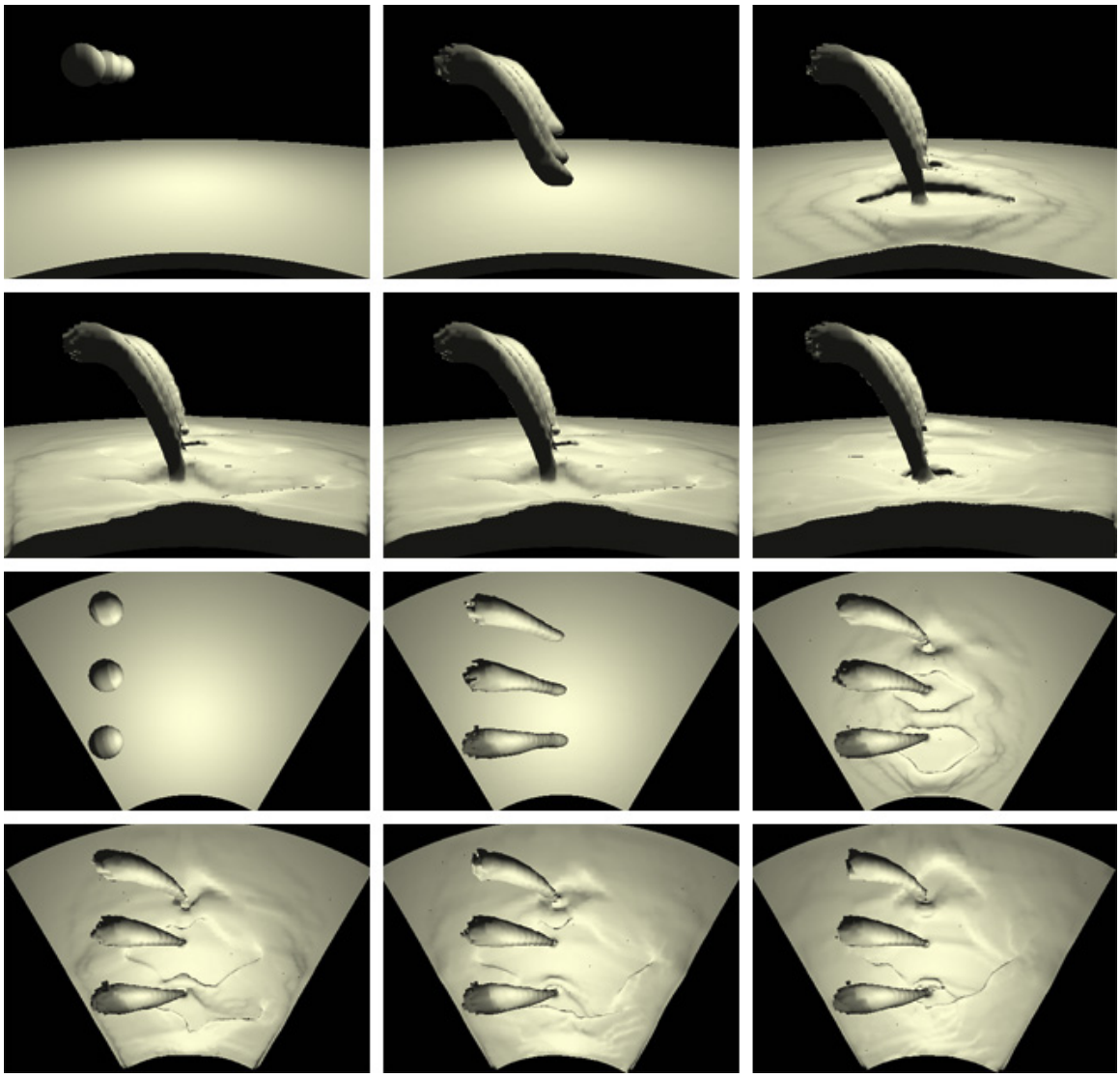

Fig. 16. A view dependent water simulation.

complicated conversion routines, the majority of the simulation time was dominated by a more strict CFL condition required by the level set methods. A view dependent setup allows much smaller simulated cell sizes near the camera and the simulation can only step through time as fast as the smallest CFL in the entire domain. This disadvantage is discussed further in the conclusions section.

CHAPTER V

CONCLUSIONS

V.1. Conclusions

To solve view dependent fluid dynamics this thesis relied on an inverse coordinate transformation of the Navier-Stokes equations as applied to the stable fluid method. It coupled this solution with a modified particle level set method to maintain a liquid surface. Simple, *infinite-fluid* boundary conditions were added using pressure constraints. Finally, the liquid surface was rendered using ray tracing and a global illumination "photon mapping" technique to render caustics.

Although the implementation of the stable fluid and level set methods were fairly straightforward, difficulties arose when trying to overcome the grid bias caused by the view dependent coordinate system. It was found that the velocity projection step in the stable fluid method, which is used to guarantee a divergence free velocity field, introduces these artifacts even after an inverse coordinate transformation is applied. The approach taken by this thesis does not preserve momentum correctly across the grid where vectors of different basis need to be compared to each other – it only computes flux across the cell boundaries and ignores any sense of direction.

Another drawback of the method is that the entire simulation can only be run at time steps satisfying the *minimum* CFL condition for stability in the level set method. As it's name implies, the stable fluid method can take large time steps and will be unconditionally stable. However, level set methods do not have such luxury and large time steps cause too much numerical dissipation and eventually the loss of fluid volume. In addition, the level set methods are much more expensive than the stable fluid method, and given that approximately 15-30 level set steps are required per frame under usual conditions, simulation times

are much larger than having a uniform, lower resolution grid.

Compared to newer, adaptive resolution fluid simulation methods such as the octree method used in [13], the view dependent method is overshadowed in its ability to support level-of-detail. To support view dependent level-of-detail, the octree method can be tailored to refine simulation cells with distance to the camera as well as to simulation features and boundaries.

One interesting method was found essentially after the development of the view dependent method that might be of help for future work. [20] transforms the stable fluid method to support 2D flows on subdivision surfaces (or any general, curvilinear coordinate system). In this method the surface parameterization is used to defeat deformations, potentially those seen in the view dependent method. After investigation, the method is essentially an inverse coordinate transformation for non-vector properties such as pressure, but provides special treatment to vector quantities such as velocity to account for changing vector bases.

V.2. Future Work

A simple approach to view dependent level-of-detail would be to implement a solution to the 2D shallow water equation as a height field on a 2D view dependent grid using either an inverse coordinate transformation or curvilinear coordinate solution described in [20]. This would allow 2.5D simulations, and if solved using general purpose graphics processors would allow very high resolution simulations in real-time.

Recently, the popularity of particle-based fluid methods has risen and a large set of interesting work has been developed. Refer to [3], [16] and [17] for details and some examples of this method. It would be interesting to see if the particle filtering used by these techniques could be adapted to a view dependent setup. Moreover, particles could be

merged or refined programmatically as they move with distance to the camera.

REFERENCES

[1] Anderson, J. D., *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill, Inc., New York, NY, 1995.

[2] Adalsteinsson D. and Sethian J., "The fast construction of extension velocities in level set methods," *Journal of Computational Physics,* Vol. 148 pp. 2-22, 1999.

[3] Clavet, S., Beaudoin, P. and Poulin, P., "Particle-based viscoelastic fluid simulation," *Proceedings of Symposium on Computer Animation '05,* pp. 219-228, 2005.

[4] Feldman, B. E., O'Brien, J.F. and Arikan, O., "Animating suspended particle explosions," *ACM Transactions on Graphics (Proc. SIGGRAPH '03)*, 2003.

[5] Enright D., Marschner S. and Fedkiw R., "Animation and rendering of complex water surfaces," *ACM Transactions on Graphics (Proc. SIGGRAPH '02)*, pp. 736-744, 2002.

[6] Foster N. and Fedkiw R., "Practical animation of liquids," *ACM Transactions on Graphics (Proc. SIGGRAPH '01)*, pp. 23-30, 2001.

[7] Foster N. and Metaxes D., "Realistic animation of liquids," *Graphical Models and Image Processing,* Vol. 58, pp. 204-212, 1996.

[8] Fedkiw R., Stam J. and Jensen H., "Visual simulation of smoke," *ACM Transactions on Graphics (Proc. SIGGRAPH '01)*, pp. 15-22, 2001.

[9] Goktekin T. G., Bargteil A. W. and O'Brien J. F., "A method for animating viscoelastic fluids," *ACM Transactions on Graphics (Proc. SIGGRAPH '04)*, pp. 463-468, 2004.

[10] Harlow F. and Welch J., "Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface," *The Physics of Fluids,* Vol.8, pp. 2182-2189, 1965.

[11] Jensen, H. W., *Realistic Image Synthesis Using Photon Mapping*, A K Peters, Ltd., Natick, MA, 2001

[12] Lorensen, W. and Cline, H.E., "Marching cubes: A high resolution 3D surface construction algorithm," *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Applications*, pp. 163-169, 1987.

[13] Losasso F., Gibou F. and Fedkiw R., "Simulating water and smoke with an octree data structure," *ACM Transactions on Graphics (Proc. SIGGRAPH '04)*, pp. 457-462, 2004.

[14] Mihalef V., Metaxas D. and Sussman M., "Animation and control of breaking waves," *Proceedings of Symposium on Computer Animation '04*, pp. 315-324, 2004.

[15] McNamara A., Treuille A., Popovic Z. and Stam J., "Fluid control using the adjoint method," *ACM Transactions on Graphics (Proc. SIGGRAPH '04)*, pp. 447-454, 2004.

[16] Muller, M., Solenthaler, B., Keiser, R. and Gross, M., "Particle-based fluid-fluid interaction," *Proceedings of Symposium on Computer Animation '05*, pp. 237-244, 2005.

[17] Muller, M., Charypar, D. and Gross, M., "Particle-based fluid simulation for interactive applicaitions," *Proceedings of Symposium on Computer Animation '03*, pp. 154-159, 2003.

[18] Osher S. and Fedkiw R., *Level Set Methods and Dynamic Implicit Surfaces*, Springer-Verlag New York, Inc., New York, NY, 2003.

[19] Shirley, P. and Morley, R.K., *Realistic Ray Tracing,* AK Peters, Natick, MA, 2003.

[20] Stam, J., “Flows on surfaces of arbitrary topology”, *ACM Transactions on Graphics (Proc. SIGGRAPH ’03)*, pp. 724-731, 2003.

[21] Stam, J., ”Stable fluids,” *ACM Transactions on Graphics (Proc. SIGGRAPH ’99)*, pp. 121-128, 1999.

[22] Teukolsky, W.H., Vetterling, W., T. and Flannery, B.P., *Numerical Recipes in C: The Art of Scientific Computing,* Cambridge University Press, Cambridge, UK, 1996.

[23] Thompson, J. F., Warsi, Z. U. A. and Mastin, C. W., *Numerical Grid Generation,* Elsevier Science Publishing Co., New York, NY, 1985.

VITA

Brian Arthur Barran
12715 New Kentucky Rd.
Cypress, TX 77429
bbarran@viz.tamu.edu

Education

M.S. in Visualization Sciences	Texas A&M University, May 2006
B.S. in Computer Science	Texas A&M University, May 2002

Employment

Software Engineer	Electronic Arts Canada, July 2005 - Continuing after graduation
Research Assistant	Texas A&M University, February 2004 - May 2005
Software Engineer	Hewlett-Packard, May 2000 - August 2003