Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid

Nuttapong Chentanez Matthias Müller
NVIDIA PhysX Research

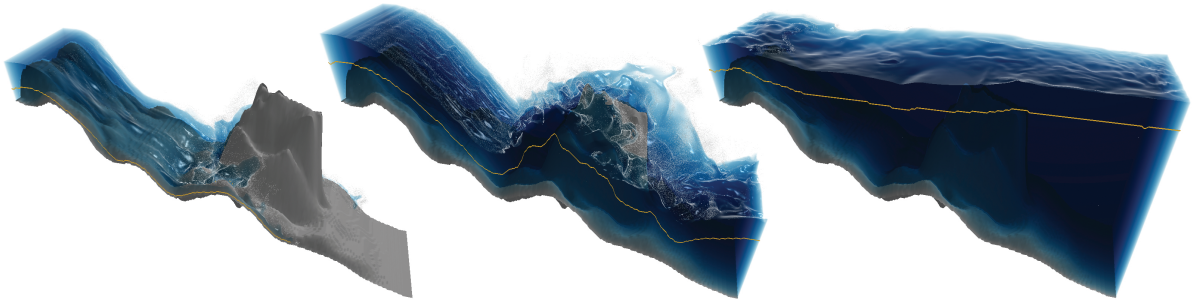

Figure 1: *Simulation of a flood at 30 frames per second including physics and rendering. Water flows from the left into an uneven terrain. The tall cells (below the orange line) represent the major part of the water volume while the computation is focused to the surface area represented by cubic cells (above the orange line). Particles are used to add visual richness to the scene.*

Abstract

We present a new Eulerian fluid simulation method, which allows real-time simulations of large scale three dimensional liquids. Such scenarios have hitherto been restricted to the domain of off-line computation. To reduce computation time we use a hybrid grid representation composed of regular cubic cells on top of a layer of tall cells. With this layout water above an arbitrary terrain can be represented without consuming an excessive amount of memory and compute power, while focusing effort on the area near the surface where it most matters. Additionally, we optimized the grid representation for a GPU implementation of the fluid solver. To further accelerate the simulation, we introduce a specialized multigrid algorithm for solving the Poisson equation and propose solver modifications to keep the simulation stable for large time steps. We demonstrate the efficiency of our approach in several real-world scenarios, all running above 30 frames per second on a modern GPU. Some scenes include additional features such as two-way rigid body coupling as well as particle representations of sub-grid detail.

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

Keywords: fluid simulation, multigrid, tall cell grid, real time

1 Introduction

Fluid simulation has a long history in computer graphics and has attracted hundreds of researchers in the past three decades. One of the main reasons for the fascination with fluids is the rich and complex behavior of liquids and gases. Due to the computational expense of capturing this complexity, fluid simulations are typically executed off-line. The computational load has so far made it hard to reproduce realistic scenarios in real time.

There are two basic approaches to solving the fluid equations: the grid-based (Eulerian) and the particle-based (Lagrangian) approach. Both have been successfully used as off-line methods to create impressive effects in feature films and commercials. One way to make such methods fast enough for real-time applications, such as computer games, is to reduce the grid resolution or the number of particles from the millions to the thousands. In the grid-based case, another way to accelerate the simulation is to reduce the dimensionality of the problem, most often from a 3 dimensional grid to a 2.5 dimensional height field representation. This reduction comes at a price: interesting features of a full 3D simulation such as splashes and overturning waves get lost because the height field representation cannot capture them.

In this paper we propose a new grid-based method that is fast enough to simulate fully three dimensional large scale scenes in real time. The main idea is to combine a generalized height field representation with a three dimensional grid on top of it. In contrast to a traditional height field simulation, we simultaneously solve the three dimensional Euler equations on both the height field columns and the regular cubic grid cells.

Our method is an adaptation of the approach proposed by [Irving et al. 2006]. In their paper, the authors discretize the fluid domain using a generalized grid, which contains both regular cubic cells and tall cells. The tall cells represent an arbitrary number of consecutive cubic cells in the up direction. With this generality the data structures as well as the computations become quite complex. For instance, there is a variable number of face velocities that need to be stored per tall grid cell, depending on the heights of adjacent tall cells. Our goal was to reduce the complexity of the general approach, while retaining enough flexibility to capture the important configurations of a three dimensional liquid. To this end, we introduce three restrictions/modifications:

- Each water column contains exactly one tall cell
- The tall cell is located at the bottom of the water column
- Velocities are stored at the cell center for regular cells and at the top and bottom of tall cells

These modifications greatly simplify data structures and algorithms as well as making the method GPU friendly. In addition, due to the simplified layout, we were able to formulate and implement a fast, parallel, multigrid Poisson solver for the tall cell grid. To accelerate our method further, we modified the level set and velocity advection schemes to ensure stability for the large time steps used in real time applications and to allow efficient implementation on GPUs.

To summarize, the main contributions of this work are:

1. A tall cell grid data structure that allows for efficient liquid simulation within a wide variety of scenarios.
2. An efficient multigrid Poisson solver for the tall cell grid. Our solver can also be used to accelerate fluid simulations on the commonly employed staggered regular grid.
3. Several modifications in the level set and velocity advection schemes that allow both larger time steps to be used and an efficient GPU implementation.

2 Related Work

Early work in the field of Eulerian fluid simulation in computer graphics include [Foster and Metaxas 1996] who used finite differences to solve the Navier-Stokes equations, [Stam 1999] who introduced the semi-Lagrangian method for advection and [Foster and Fedkiw 2001] who combined Lagrangian particles with the level set method to track the free surface of liquids. Since then, a wide variety of methods have been proposed to accelerate fluid simulations in order to cope with large scenes. One solution is to use adaptive grids in order to focus the computational effort to important regions such octrees [Losasso et al. 2004], tetrahedral grids [Feldman et al. 2005], [Klingner et al. 2006], [Chentanez et al. 2007], [Batty et al. 2010], Voronoi cells grids [Sin et al. 2009], [Brochu et al. 2010] or tall cell grids [Irving et al. 2006]. Using an adaptive grid is one of the main components of our method to reach real-time performance.

The simulation of a fluid can be split into three main steps: advection, surface tracking and pressure projection. Each of these steps has been the focus of many research papers. For instance, [Kim et al. 2008] proposed to advect derivatives along with physical quantities to improve the quality of the simulation. [Selle et al. 2008] used a modified MacCormack scheme to lift the semi-Lagrangian step to second order accuracy. To reduce volume-loss, [Enright et al. 2002] added particles on both sides of the liquid surface to correct the level set. Apart from level sets, other representations of the liquid surface have been proposed such as particles only [Zhu and Bridson 2005], [Adams et al. 2007], [Yu and Turk 2010] or explicit triangle meshes [Bargteil et al. 2005], [Müller 2009], [Brochu and Bridson 2009], [Wojtan et al. 2010] requiring various topological fixes. [Enright et al. 2003] used the ghost fluid method to improve the accuracy of pressure projection near the free surface. In many cases, the pressure projection step is the slowest part in liquid simulations because it involves solving a large linear system at each time step. The preconditioned conjugate gradients (PCG) method is commonly used [Foster and Fedkiw 2001], [Bridson 2008] for solving this system efficiently. The regularity of Eulerian grids makes the multigrid approach an effective alternative to PCG [Molemaker et al. 2008]. [McAdams et al. 2010] combined both approaches and used one multigrid V-Cycle as a conjugate gradients's pre-conditioner. To accelerate the multigrid approach [Lentine et al. 2010] modified the restriction and prolongation stencils to only consider velocities on the faces of coarser cells.

The complex and interesting motion of a fluid is typically caused by its interaction with the solid environment. Therefore, handling solid boundary conditions correctly has been a further active research area. [Takahashi et al. 2002] proposed to use a volume of fluid fraction method to handle two-way rigid body coupling accurately. [Carlson et al. 2004] included fluid cells as well as cells occupied by rigid bodies in one pressure solve. Similarly, [Klingner et al. 2006] combined fluid motion and rigid body momentum into a single linear system and solved it simultaneously. Later, the method was extended to include soft body-fluid coupling [Chentanez et al. 2006] and then re-formulated to conserve momentum yielding a symmetric system matrix in [Robinson-Mosher et al. 2008]. Two-way coupling of fluids with cloth and thin shells was studied by [Guendelman et al. 2005]. Using a variational formulation [Batty et al. 2007] were able to handle fluid-solid interactions with sub-grid accuracy.

Other approaches to simulate liquids include particles-based methods such as [Müller et al. 2003], [Premoze et al. 2003], [Adams et al. 2007], [Solenthaler and Pajarola 2009] and lattice-Boltzmann models [Thürey and Rüde 2004], [Thürey and Rüde 2009]. Real-time performance has been achieved by using the pipe model [Šťava et al. 2008], the 2D wave equation [Holmberg and Wünsche 2004] and the shallow water equations [Thurey et al. 2007], [Chentanez and Müller-Fischer 2010] to name a few. Height field methods cannot capture the 3D phenomena faithfully though. So far, only a few researchers have shown 3D Eulerian liquid simulation at interactive rates. To achieve real-time performance [Crane et al. 2007] confined the liquid to a relatively small rectangular domain without general fluid-solid interaction, while [Long and Reinhard 2009] leveraged the discrete cosine transform to speed up their simulation.

3 Methods

We simulate liquids by solving the inviscid Euler Equations,

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \frac{\mathbf{f}}{\rho} - \frac{\nabla p}{\rho}, \quad (1)$$

subject to the incompressibility constraint

$$\nabla\cdot\mathbf{u} = 0, \quad (2)$$

where $\mathbf{u} = [u, v, w]^{\mathrm{T}}$ is the fluid velocity field, p is the pressure, t is time, ρ the fluid density and $\mathbf{f}$ is a field of external forces. The equations are solved in the domain specified by a scalar level-set field ϕ in the region where $\phi < 0$. ϕ itself is evolved by

$$\frac{\partial \phi}{\partial t} = -\mathbf{u}\cdot\nabla\phi. \quad (3)$$

[Foster and Fedkiw 2001]. Dirichlet and Neumann boundary conditions must be taken into account as well when solving these equations.

3.1 Discretization

As a discretization of the simulation domain we use the specialized tall cell grid discussed in Section 1. The diagram on the left of Figure 2 shows the tall cell grid in 2D. From bottom to top, each column consists of terrain, one tall cell and a fixed number of regular cells. The terrain height and the height of the tall cell are discretized to be a multiple of the grid spacing Δx. These height values are stored in two arrays. For regular cells, all the physical quantities like velocity, level set value and pressure are stored at the

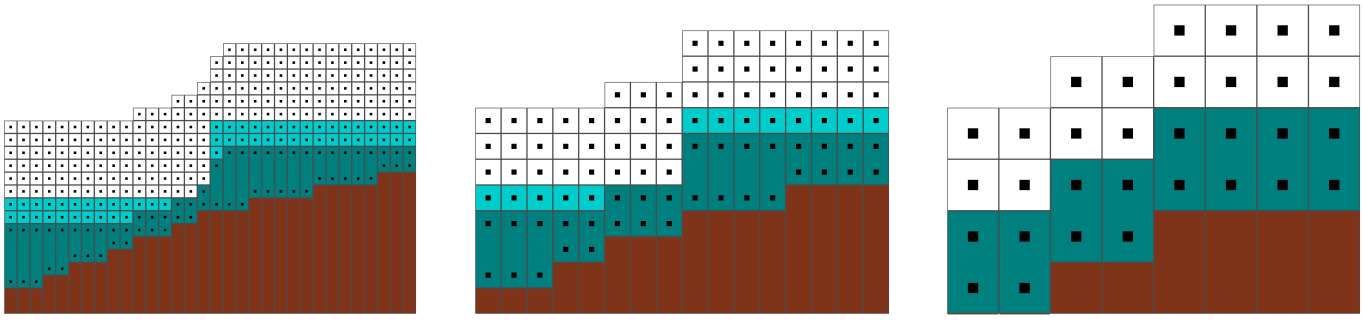

Figure 2: *Left: 2D cross section of the tall cell grid. Each column stores the terrain height, one tall cell and a constant number of regular cubic cells. Physical quantities are stored at the center of regular cells and at the top and the bottom of tall cells. Middle and Right: The next two coarser levels in the hierarchy of grids used for velocity extrapolation and the multigrid solver.*

cell center. For tall cells, these quantities are stored at the center of the topmost and the bottommost subcells. In terms of implementation, a quantity q is stored in a compressed 3D array, $q_{i,j,k}$, of size $(B_x, B_y + 2, B_z)$ where B_x and B_z are the number of cells along the x and z axis respectively, B_y is the constant number of regular cells along the y-axis per column and the $+2$ comes from the top and the bottom values stored in per tall cell. In addition, we store the terrain height $H_{i,k}$ and the tall cell height $h_{i,k}$ in two 2D arrays of size (B_x, B_z). The y-coordinate of the uncompressed position of an array element $q_{i,j,k}$ is given by

$$y_{i,j,k} = \begin{cases} H_{i,k} + 1 & \text{if } j = 1 \text{ (tall cell bottom)} \\ H_{i,k} + h_{i,k} & \text{if } j = 2 \text{ (tall cell top)} \\ H_{i,j} + h_{i,k} + j - 2 & \text{if } j \geq 3 \text{ (regular cell).} \end{cases} \quad (4)$$

We denote a quantity stored in the compressed array at position i, j, k with $q_{i,j,k}$ without parentheses, and a quantity at the uncompressed world location $(x\Delta x, y\Delta x, z\Delta x)$ as $q_{(x,y,z)}$ with parentheses. Depending on the y-coordinate, there are four cases for evaluating $q_{(x,y,z)}$ based on the values stored in the compressed array.

- If $y \leq H_{x,z}$ the value of $q_{(x,y,z)}$ is the value below the terrain.
- If $H_{x,z} < y \leq H_{x,z} + h_{x,z}$ the requested quantity lies within the tall cell. In this case, we linearly interpolate from the top and the bottom sub-cells of the tall cell:

$$q_{(x,y,z)} = \frac{y - H_{x,z}}{h_{x,z}} q_{x,2,z} + (1 - \frac{y - H_{x,z}}{h_{x,z}}) q_{x,1,z} \quad (5)$$

- If $H_{x,z} + h_{x,z} < y < H_{x,z} + h_{x,z} + B_y$ we have to look up the quantity from the regular cells in the compressed array as

$$q_{(x,y,z)} = q_{x,(y - H_{x,z} - h_{x,z} - 2),z} \quad (6)$$

- Otherwise $q_{(x,y,z)}$ gets the value above air.

This definition of $q_{(x,y,z)}$ hides the tall cell structure of the grid. Once implemented, the grid can be accesses as if it was a regular grid composed of cubical cells only, which simplifies what follows significantly. A quantity at an arbitrary point in space can be computed using tri-linear interpolation of the nearest $q_{(x,y,z)}$'s.

There are a few properties that distinguish our tall cell formulation from [Irving et al. 2006].

1. Our tall cell grid has a constant data size for all quantities p, $\mathbf{u}$, ϕ. This allows for an efficient GPU implementation. In contrast, [Irving et al. 2006] store a variable number of values of p and ϕ depend on the number of tall cells used per water column. Moreover, each tall cell stores one velocity component per touching tall cell neighbor. The fact that the number of tall cells and the number of touching neighbors vary during the simulation complicates the data storage and implementation of [Irving et al. 2006].
2. We use a collocated grid, which reduces the number of rays that need to be traced in the semi-lagrangian step. This part of the computation is significant, especially if the resolution used for surface tracking is higher than the one for simulation.
3. In contrast to the general case, the stencil of the discrere Laplacian operator on our simplified tall cell grid only includes a constant number of neighbors (see Section 3.7).

3.2 Time integration

Our time integration scheme is summarized in Algorithm 1. With the exception of the remeshing step, it follows standard Eulerian liquid simulation [Enright et al. 2002]. First, we extrapolate the velocity field into the air region. Then, after reinitializing the signed distance field, we advect the level set and the velocity field and take external forces into account. The next step is to recompute the height of the tall cells and transfer the physical quantities to the new grid. Finally, we enforce incompressibility by making the velocity field divergence free.

Algorithm 1 Time step

1: Velocity extrapolation
2: Level set reinitialization
3: Advection and external force integration
4: Remeshing
5: Incompressibility enforcement

3.3 Velocity Extrapolation

The x-component of the velocity field u can be extrapolated into the air region, where $\phi > 0$, by solving the equation

$$\frac{\partial u}{\partial \tau} = -\frac{\nabla \phi}{|\nabla \phi|} \cdot \nabla u, \quad (7)$$

where τ is fictitious time [Enright et al. 2002]. Similar equations are used for v and w. For a CPU implementation, an $O(n \log n)$ algorithm exists for solving this equation efficiently [Adalsteinsson and Sethian 1997]. To solve the equation efficiently on GPUs [Jeong et al. 2007] proposed to use a variation of an Eikonal solver.

If the time step is not too large, the velocity is only needed within a narrow band of air cells near the liquid surface [Enright et al. 2002].

In this case, the two algorithms mentioned above are efficient because they can be terminated early. However, in our examples, the velocity is relatively large and the time step we use ($\frac{1}{30}$ s) is much larger than is typically used in water simulations. Therefore, water can cross several grid cells in a single time step. To make this possible, we need velocity information far away from the liquid surface.

We observed that we only need an accurate velocity field close to the surface, while far away from the liquid a crude estimate is sufficient. Therefore, we apply the algorithm proposed in [Jeong et al. 2007] only in a narrow band of two cells. Outside this region we use a hierarchical grid for extrapolating the velocity field. An example of a hierarchy of grids is shown in Figure 2. All velocity components can be extrapolated at the same time because we use a collocated grid.

3.3.1 Hierarchical Grid for Velocity Extrapolation

The number of levels of the hierarchical grid is determined by $L = \log_2 \min(B_x, B_y, B_z)$. The finest level of the grid corresponds to the simulation grid with $\Delta x^L = \Delta x$, $\mathbf{u}^L_{i,j,k} = \mathbf{u}_{i,j,k}$, $H^L_{i,k} = H_{i,k}$ and $h^L_{i,k} = h_{i,k}$. On coarser levels $l, L > l \geq 1$, the quantities H^{l+1} and h^{l+1} are defined via down sampling as

$$H^l_{i,k} = \left\lfloor \frac{\min_{i'=2i,k'=2k}^{2i+1,2k+1} H^{l+1}_{i',k'}}{2} \right\rfloor, \quad (8)$$

$$h^l_{i,k} = \left\lceil \frac{\max_{i'=2i,k'=2k}^{2i+1,2k+1} H^{l+1}_{i',k'} + h^{l+1}_{i',k'}}{2} \right\rceil - H^l_{i,k}, \quad (9)$$

and $\Delta x^l = 2\Delta x^{l+1}$, $B^l_x = \frac{B^{l+1}_x}{2}$, $B^l_y = \frac{B^{l+1}_y}{2}$, and $B^l_z = \frac{B^{l+1}_z}{2}$.

With this definition, coarser grids are tall cell grids and are guaranteed to cover all cells in the finer grid, as the diagrams in the middle and on the right of Figure 2 show. The velocities in the hierarchy of grids are evaluated by sweeping down then sweeping up the hierarchy. On the finest level L, we declare the velocity of a cell to be known if the cell is a liquid cell or if the velocity is already extrapolated. We then go through the levels from finest to coarsest and obtain velocities by tri-linear interpolation of the velocities of the previous level using only known velocities and renormalizing the interpolation weights accordingly. The velocity of a coarse cell is declared to be known if at least one corresponding finer cell velocity is known. We then traverse the hierarchy in the reverse order from coarsest to finest and evaluate velocities on finer levels by tri-linearly interpolating values from coarser grids. After these two passes every cell of the finest grid has a known velocity.

3.4 Level Set Reinitialization

Advecting ϕ destroys its property of being a signed distance field. Therefore, ϕ needs to be reinitialized periodically to be accurate at least for two to three cells away from the liquid surface. We use the method of [Jeong et al. 2007] for this step. Since we use a higher resolution grid for surface tracking than for the simulation in most of our examples, this step can be quite costly. In practice, we found that we can simplify the process significantly, while still getting satisfactory results. First, we run the reinitialization step only every ten frames. Second, during reinitialization, we do not modify ϕ values of grid points next to the surface in order to avoid moving it. Third, in every frame we clamp the value of ϕ next to the liquid surface to not exceed the grid spacing Δx. Without clamping, incorrect values get advected near the surface and cause surface bumpiness. To stabilize the process further we clamp all ϕ values to have magnitude less than $5\Delta x$. We have not seen significant problems or artifacts due to these stabilizations.

3.5 Advection and external force integration

To advect $\mathbf{u}$ we use the modified MacCormack scheme proposed by [Selle et al. 2008] and revert to simple Semi-Lagrangian advection if the new velocity component lies outside the bound of the values used for interpolation. To update ϕ we use Semi-lagrangian advection because we found that MacCormack causes noisier surfaces even if care is taken near the interface. Due to the collocated grid we only need to trace the Semi-Lagrangian ray once for all quantities reusing the same interpolation weights. After that, we integrate external forces such as gravity using forward-Euler.

3.6 Remeshing

After advection, we identify liquid cells as those where $\phi \leq 0$. At this point we need to define new values $h_{i,k}$, i.e. decide how many cells above the terrain should be grouped into one tall cell for each column (i, k). There are a few desirable constraints that may conflict each other:

1. There must be at least G_L regular cells below the bottom most liquid surface to capture the 3D dynamics of the liquid.
2. There must be at least G_A regular cells above the top most liquid surface, to allow water to slosh into the air in the next time steps.
3. The heights of adjacent tall cells must not differ by more than D units to reduce the volume gain artifacts as will be discussed in Section 5.

We first iterate through each pair (i, k) and compute the maximum and minimum y-coordinate of the top of the tall cell that satisfy constraints (1) and (2), respectively. Next we initialize the temporary variable $y^{\text{tmp}}_{i,k}$ to be the average of the two extrema. To reduce the differences in height of adjacent tall cells we then run several smoothing passes on $y^{\text{tmp}}_{i,k}$. During the smoothing we clamp $y^{\text{tmp}}_{i,k}$ so that it always satisfies conditions (1) and (2), giving preference to condition (2) by enforcing it after condition (1). Finally, we iterate through (i, k) again and enforce condition (3) in a Jacobi-type fashion using

$$y^{\text{tmp}'}_{i,k} = \min(y^{\text{tmp}}_{i,k}, \max_{|i'-i|+|k'-k|=1} y^{\text{tmp}}_{i',k'} + D) \quad (10)$$

In our examples we used $8 \leq G_L \leq 32, G_A = 8, 3 \leq D \leq 6$ and between one and two Jacobi iterations. Finally we set $h^{\text{new}}_{i,k} = y^{\text{tmp}}_{i,k} - H_{i,k}$. The algorithm attempts to make compromise among the constraints but may not satisfy all of them. Once we know the new heights of the tall cells, we transfer all the physical quantities to the new grid. For regular cells, we simply copy the values at the corresponding locations from the old grid or interpolate linearly if the location was occupied by a tall cell in the previous time step. For tall cells, we do a least square fit to obtain the values at the bottom and the top of the cell, similar to [Irving et al. 2006].

3.7 Enforcing Incompressibility

Suppose the velocity field after the advection and the remeshing step is $\mathbf{u}^*$. We want to find the pressure field p such that

$$\nabla \cdot (\mathbf{u}^* - \frac{\Delta t}{\rho} \nabla p) = 0. \quad (11)$$

Assuming a constant ρ, we have a Poisson equation

$$\nabla^2 p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^*. \quad (12)$$

To discretize this equation, we need to define the divergence, gradient and Laplacian operators on our restricted tall cell grid. We use

the following divergence operator

$$(\nabla \cdot \mathbf{u})_{i,j,k} = (\frac{\partial u}{\partial x})_{i,j,k} + (\frac{\partial v}{\partial y})_{i,j,k} + (\frac{\partial w}{\partial z})_{i,j,k}, \quad (13)$$

where $(\frac{\partial u}{\partial x})_{i,j,k} = \frac{u^+_{i,j,k} - u^-_{i,j,k}}{\Delta x}$ and

$$u^+_{i,j,k} = \begin{cases} \frac{u_{i,j,k} + u_{(i+1,y,k)}}{2} & \text{if the cell } (i+1,y,k) \text{ is not solid} \\ u_{\text{solid}} & \text{otherwise.} \end{cases} \quad (14)$$

$u^-_{i,j,k}$ is defined similarly and so are the terms $(\frac{\partial v}{\partial y})_{i,j,k}$ and $(\frac{\partial w}{\partial z})_{i,j,k}$.

For the Laplacian we use

$$(\nabla^2 p)_{i,j,k} = (\frac{\partial^2 p}{\partial x^2})_{i,j,k} + (\frac{\partial^2 p}{\partial y^2})_{i,j,k} + (\frac{\partial^2 p}{\partial z^2})_{i,j,k}, \quad (15)$$

where $(\frac{\partial^2 p}{\partial x^2})_{i,j,k} = \frac{p^{x+}_{i,j,k} - 2p_{i,j,k} + p^{x-}_{i,j,k}}{\Delta x^2}$ and

$$p^{x+}_{i,j,k} = \begin{cases} p_{i,j,k} \frac{\phi_{(i+1,y,k)}}{\phi_{i,j,k}} & \text{if cell } (i+1,y,k) \text{ is air,} \\ s_{(i+1,y,k)} p_{i,j,k} + & \\ \quad (1 - s_{(i+1,y,k)}) p_{(i+1,y,k)} & \text{otherwise,} \end{cases} \quad (16)$$

where $s_{i,j,k}$ is the fraction of solid in a cell. $p^{x-}_{i,j,k}$ is defined similarly and so are the terms $(\frac{\partial^2 p}{\partial y^2})_{i,j,k}$ and $(\frac{\partial^2 p}{\partial z^2})_{i,j,k}$. Equation 16 incorporates two important methods. First, for air cells we use the ghost-fluid method [Enright and Fedkiw 2002] to get more accurate free-surface boundary conditions by assigning negative pressures to air cells such that $p = 0$ exactly on the liquid surface, i.e. where $\phi = 0$ and not at the center of the air cell. The second line of Equation 16 utilizes solid fraction [Batty et al. 2007]. It is not only valid for $s = 0$ and $s = 1$ but for any value in between so cells that are only partially occupied by solids can be handled correctly. This is an important feature in the case of a hierarchical grid where coarser cells cover both, solid and fluid cells of finer levels.

Discretizing Equation 12 by applying the operators defined above to all the regular cells and the bottom and the top of tall cells yields a linear system for the unknown pressure field p. After solving for p, we compute its gradient using

$$(\nabla p)_{i,j,k} = [(\frac{\partial p}{\partial x})_{i,j,k}, (\frac{\partial p}{\partial y})_{i,j,k}, (\frac{\partial p}{\partial z})_{i,j,k}]^{\mathrm{T}}, \quad (17)$$

where $(\frac{\partial p}{\partial x})_{i,j,k} = \frac{p^{x+}_{i,j,k} - p^{x-}_{i,j,k}}{\Delta x}$. $(\frac{\partial p}{\partial y})_{i,j,k}$ and $(\frac{\partial p}{\partial z})_{i,j,k}$ are defined similarly. The velocity can then be corrected using

$$\mathbf{u}_{i,j,k} -= \frac{\Delta t}{\rho} (\nabla p)_{i,j,k} \quad (18)$$

Solving the linear system for p is usually the most time consuming step in fluid simulations. Without tall cells, the matrix of our system is identical to the one appearing in standard Eulerian regular grid liquid simulation used by many authors [Foster and Fedkiw 2001], [Enright et al. 2002], [Rasmussen et al. 2004], [Guendelman et al. 2005], [Batty et al. 2007], [Kim et al. 2008] and can be solved efficiently using the incomplete Cholesky preconditioned Conjugate Gradients method. In the presence of tall cells though, the resulting linear system is non-symmetric and the Conjugate Gradients method cannot be used. On the other hand, even though non-symmetric, the system is still much simpler than the one emerging from the general case of [Irving et al. 2006] because we have a constant number of coefficients that need to be stored per cell. This property makes the problem well suited for a data parallel architecture such as a GPU and for a multigrid approach. We therefore decided to write a parallel multigrid solver using CUDA[Sanders and Kandrot 2010].

3.7.1 Multigrid Overview

Algorithm 2 summarizes our multigrid pressure solver.

Algorithm 2 Multigrid

```
Compute matrix A^L for level L
for l = L − 1 down to 1 do
    Down sample φ^{l+1} → φ^l and s^{l+1} → s^l
    Compute matrix A^l for level l
end for
b^L = −(Δt/ρ)(∇ · u)
p^L = 0
for i = 1 to num_Full_Cycles do
    Full_Cycle()
end for
for i = 1 to num_V_Cycles do
    V_Cycle(L)
end for
```

The hierarchy of grids we use is the same as the one described in Section 3.3. On each level, a linear system of the form $\mathbf{A}^l p^l = b^l$ has to be solved. To down sample s^{l+1} to s^l, we do an 8-to-1 average for regular cells and a least square fit of the 8-to-1 averages of the sub cells for the tall cells. For down sampling ϕ^{l+1} to ϕ^l we distinguish the following two cases:

1. if the 8 ϕ-values all have the same sign or $l < L - C$ we use the 8-to-1 average,
2. otherwise we use the average of the positive ϕ-values.

The key idea is to ensure that air bubbles persist in the C finest levels. In those levels, bubbles have a significant influence on the resulting pressure values. On the other hand, letting air bubbles disappear in coarser levels is not problematic because only a general pressure profile is needed there in order to get accurate pressure values in the original grid. Tracking bubbles on coarser levels is not only unnecessary but we found that keeping them yields incorrect profiles because their influence gets exaggerated. We use $C = 2$ in all simulations.

We then compute the coefficients of the $\mathbf{A}^l$ for each level using Equation 16. Unlike [McAdams et al. 2010], our solver handles sub-grid features correctly through the ghost fluid and solid fraction methods on all the levels of the hierarchy. So in contrast to [McAdams et al. 2010], our solver converges even in the presence of irregular free-surface and solid boundaries. Handling sub-grid features correctly is crucial to obtain meaningful pressures fields on coarse levels. For example, in the hydrostatic case we can enforce free surface boundary conditions at the correct location up to first order to get a correct linear pressure profile on all levels of the hierarchy. Without using sub-grid resolution, slightly different problems would be solved on the coarse grids.

For smoothing, we use the Red-Black Gauss-Seidel(RBGS) method and solve the system in two parallel passes. The restriction operator tri-linearly interpolates r, where $r_{(x,y,z)}$ is specially computed as

$$r_{(x,y,z)} = \begin{cases} r_{x,1,z} & \text{if } y = H_{x,z} + 1 \\ r_{x,2,z} & \text{if } y = H_{x,z} + h_{x,z} \\ r_{x,y-H_{x,z}-h_{x,z}-2,z} & \text{if } H_{x,z} + h_{x,z} \le y \\ & \quad < H_{x,z} + h_{x,z} + B_y \\ 0 & \text{otherwise.} \end{cases} \quad (19)$$

Note that $r_{(x,y,z)}$ is zero everywhere inside a tall cell except at the top and bottom, because divergence is measured only at the top and bottom sub-cells. Using a wider stencil for restriction as in

[McAdams et al. 2010] is more expensive and does not yield a faster convergence rate in our tests. For prolongation we also use tri-linear interpolation. On the boundary, if we find that a pressure value outside the grid is needed for interpolation, then we simply ignore it and renormalize the interpolation weights. If all values are outside the grid the pressure is set to zero.

There are three critical steps to making our multigrid algorithm converge:

1. The use of full-cycles.
2. Preserving air bubbles in the finest levels.
3. Using the ghost fluid and solid fraction methods.

Not considering any one of these leads to either stagnation or even divergence of the solver as reported in [McAdams et al. 2010].

Algorithm 3 V_Cycle(l)

if $l == 1$ **then**
  Solve the linear system, $\mathbf{A}^1 p^1 = b^1$
else
  for $i = 1$ to num_Pre_Sweep **do**
    Smooth(p^l)
  end for
  $r^l = b^l - \mathbf{A}p^l$
  $b^{l-1} = \text{Restrict}(r^l)$
  $p^{l-1} = 0$
  V_Cycle($l-1$)
  $p^l = p^l + \text{Prolong}(p^{l-1})$
  for $i = 1$ to num_Post_Sweep **do**
    Smooth (p^l)
  end for
end if

Algorithm 4 Full_Cycle()

$p^{\text{tmp}} = p^L$
$r^L = b^L - \mathbf{A}p^L$
for $l = L-1$ down to 1 **do**
  $r^l = \text{Restrict}(r^{l+1})$
end for
$b^1 = r^1$
Solve the linear system, $\mathbf{A}^1 p^1 = b^1$
for $l = 2$ to L **do**
  $p^l = \text{Prolong}(p^{l-1})$
  $b^l = r^l$
  V_Cycle(l)
end for
$p^L = p^{\text{tmp}} + p^L$

3.8 Optimizations

We optimized our method in several ways to increase its performance.

- For all tri-linear interpolations, we first interpolate along the y-axis. This step always requires exactly 2 consecutive grid point values independent of whether the entry is part of a tall or a regular cell. In this way, only 8 memory access are necessary instead of up to 16 when using Equation 5 naively.
- In the Gauss Seidel step, to get the pressure below the top pressure value of a tall cell, we access $p_{i,j-1,k}$ in the compressed grid and do the interpolation implicitly via modifying the Laplace stencil instead of querying $p_{(i,y-1,k)}$ through the mapping function.
- We clamp the grid hierarchy at the level that completely fits in the GPU's shared memory. This top level can then be solved efficiently to high precision by executing multiple Gauss Seidel iterations using a single kernel (see [Cohen et al. 2010]).
- We only build the hierarchical grid once per simulation frame at the incompressibility solve step. The same hierarchy can be re-used for velocity extrapolation in the next time step because remeshing happens after velocity extrapolation.

3.9 Extensions

In this section we describe a few additional methods to complement the core grid-based fluid solver.

3.9.1 Rigid Body Coupling

To handle rigid body coupling, we use a variation of the Volume of Solid Method (VOS) [Takahashi et al. 2002] and alternately run the water and the rigid body solver. Although more accurate techniques have been proposed for fluid-rigid body coupling [Carlson et al. 2004], [Chentanez et al. 2006], [Batty et al. 2007], [Robinson-Mosher et al. 2008], we use this simple method because it requires only minimal changes of the water simulator that do not affect its GPU optimized structure. For rigid to water coupling, we voxelize the rigid bodies into the water simulation grid by modifying the solid fraction s and blend the fluid and solid velocities based on this fraction. The divergence calculation treats a cell as solid if $s > 0.9$.

Special care has to be taken regarding the level set function ϕ inside rigid bodies because the ϕ resulting from the Semi-Lagrangian advection step is not correct there. We therefore define a second field ϕ^s defined inside rigid bodies only. Ideally, ϕ^s would be the extrapolation of ϕ outside the body. A correct evaluation of this function would, however, require a fast marching step. We use a simpler approach which lets ϕ diffuse into the solid over several time steps using

$$\phi^s_{i,j,k} = \frac{1}{S} \sum_{|i'-i|+|y'-y|+|k'-k|=1} (1 - s_{(i',y',k')})\phi_{(i',y',k')}$$

if $S > 0$ and

$$\phi^s_{i,j,k} = \frac{1}{6} \sum_{|i'-i|+|y'-y|+|k'-k|=1} \phi_{(i',y',k')}$$

otherwise, where $S = \sum_{|i'-i|+|y'-y|+|k'-k|=1}(1 - s_{(i',y',k')})$. For mixed cells, the two level set values are blended as $s\phi^s + (1-s)\phi$. This estimation is not strictly correct, but it is sufficient in all of our examples to generate plausible behavior.

For water to rigid coupling, we visit all the voxels that contain both rigid bodies and water and sum up the forces and torques resulting from the interaction. We consider buoyancy and drag. The buoyancy force is computed using s and the relative density of the solid w.r.t. the liquid. We use a drag force proportional to s and the relative velocity between the fluid and the solid. Again, this force is only an approximation of the real drag force but it yields plausible results in our examples.

3.9.2 Particle-Based Thickening

To reduce volume loss due to the use of large time steps we apply a variation of the particle thickening method presented in [Chentanez et al. 2007]. The method identifies thin parts of the water domain and seed particles there. These particles are moved forward in time and then the signed distance function of each particle is united with the advected ϕ. A grid location (x, y, z) is considered thin if

1. $\phi^{\text{thin}} \leq \phi \leq 0$ and

2. $\phi^l = \phi$ at $(x, y, z)\Delta x + 2\phi^{\text{thin}} \frac{\nabla\phi_{(x,y,z)}}{|\nabla\phi_{(x,y,z)}|}$ is positive and

3. $\phi^r = \phi$ at $(x, y, z)\Delta x - 2\phi^{\text{thin}} \frac{\nabla\phi_{(x,y,z)}}{|\nabla\phi_{(x,y,z)}|}$ is positive.

When a thin cell is identified, 16 particles are seeded on the disk of radius $\frac{1}{2}\Delta x$ centered at $(\frac{\phi_{(x,y,z)}}{\phi^l - \phi_{(x,y,z)}} - \frac{\phi_{(x,y,z)}}{\phi^r - \phi_{(x,y,z)}})(-\phi^{\text{thin}})\frac{\nabla\phi_{(x,y,z)}}{|\nabla\phi_{(x,y,z)}|}$ whose normal is $\frac{\nabla\phi_{(x,y,z)}}{|\nabla\phi_{(x,y,z)}|}$. The center of the disk is an estimation of the mid-point between the two water surfaces above and below the thin region. The radius of a particle is taken to be $-\phi$ at its location. Its velocity is computed via tri-linear interpolation of the velocity field. Particles whose radius is negative are ignored. In our examples we used $\phi^{\text{thin}} = -1.5\Delta x$.

3.9.3 Particles Generation

For rendering purposes, we automatically generate particles that represent spray and small droplets. At each time step cells whose ϕ-value satisfies $\phi^{\text{gen}} \leq \phi \leq 0$ are sampled with trial particles. Again, the radius of a particle is taken to be $-\phi$ at its location and its velocity is computed via tri-linear interpolation of the velocity field. After being moved forward in time we check whether the particle arrived at a location where ϕ is greater than twice its radius. If so, we seed a number of escape particles there with the same velocity plus some additional noise. These particles are rendered as spray in the flood and the lighthouse examples in Figures 1 and 5. In the lighthouse example, a subset of the spray particles is converted into mist particles. In addition, whenever spray particles fall into the main body of water they are converted to foam particles with some probability. Spray and mist particles move ballistically, the latter experiencing more drag. Foam particles stay on the surface and are advected passively with the velocity of the water.

Case	Total	VE	LA	VA	RM	PP
Manip	29.06	1.30	2.35	0.57	0.56	8.56
Tank	27.29	1.10	3.26	0.67	0.56	8.44
Flood	32.33	2.35	0.59	1.14	0.85	13.49
LightH	33.09	2.05	0.61	0.67	0.95	9.77

Table 1: *Timing for the examples scenes in milliseconds. Total stands for the frame time including rendering, VE for velocity extrapolation, LA for level set advection, VA for velocity advection, RM for remeshing and PP for pressure projection.*

Case	Sim	Surf
Manip	64x(64+2)x64	128x(128+2)x128
Tank	64x(64+2)x64	128x(128+2)x128
Flood	64x(32+2)x256	64x(32+2)x256
LightH	128x(32+2)x128	128x(32+2)x128

Table 2: *Simulation and surface tracking grid sizes used in our examples.*

4 Results

We demonstrate the features and performance of our simulation algorithm in several scenarios. The timing data for each example is listed in Table 1. All examples run in real time at more than 30 frames per second on a single NVIDIA GTX480 graphics card. The simulation time step is $\frac{1}{30}$ second in all cases. We found that executing two V-cycles and one full multigrid in the pressure solver is sufficient to get visually pleasing results. The water level in our examples does not decrease significantly over time because the multigrid solver is able to reduce the low-frequency error quickly even with only a few cycles. This is in contrast to a Jacobi type method as used by [Crane et al. 2007] who reported water loss to be a significant problem.

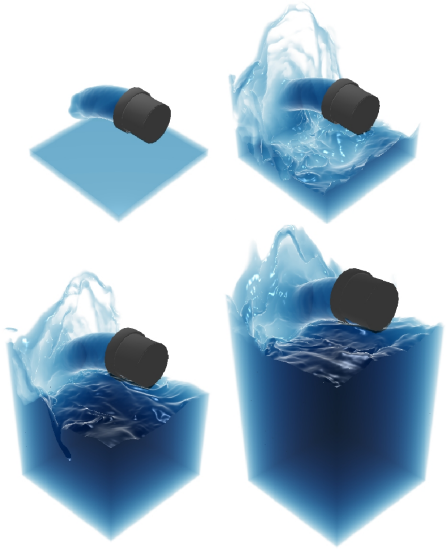

Figure 3: *Water flows from a magic inexhaustible bucket into a tank filling it up to an arbitrary level without increasing computational load.*

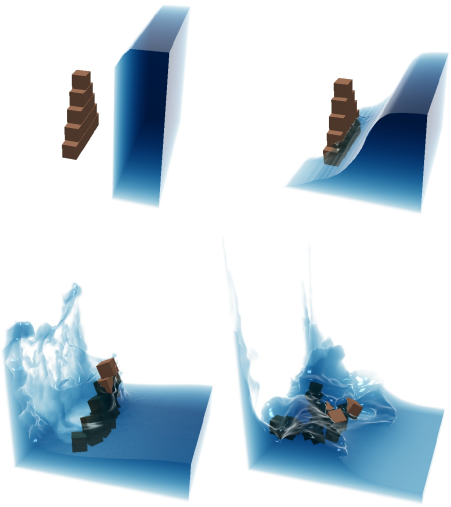

Figure 4: *This dam breaking scene demonstrates two-way interaction of water with rigid bodies and user intervention.*

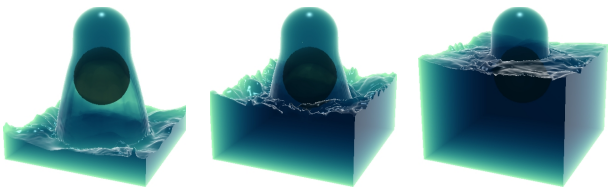

Figure 6: *Water flows past a sphere into a tank. This scene was used for comparing IC(0) PCG with our multigrid solver.*

Figure 5: *Our method allows real-time simulation of large scale scenarios. To increase realism we enriched the scene by adding various additional features. Spray, mist and foam effects are created with thousands of particles. We overlayed the beach with a simulated wet map and added an evolving foam map to the water surface. The high frequency waves are created by adding a wave texture and advecting it with the velocity field of the water. To demonstrate the interactivity of the scene, we let the user add water and interact with the rigid bodies during the simulation.*

Figure 1 shows a flooding scene. Water is injected on the left side with an increasing flow rate for 30 seconds after which the flow is abruptly stopped. This scene demonstrates the efficiency of using a tall call grid when simulating a scenario with large variations in water depth. Notice also how water fills up the uneven terrain and settles down to a flat steady state. In the scene shown in Figure 3 a jet of water fills up a tank to an arbitrary level. Simulating this scenario using only regular cells would require increasing storage and computation time with rising water level. To demonstrate two-way interaction of a liquid with rigid bodies we put a stack of boxes into a standard dam break scene as shown in Figure 4.

Figure 5 shows a large scale simulation of waves crashing and breaking over a beach. Here we used particles to add small scale detail such as spray and mist. The foam map on the water surface is advected by the water's velocity field and seeded by spray particles that fall into the main body of water. Additionally, we overlaid the sandy area with a wet map which dries out over time and gets activated when the water level rises. To add more detail to the water surface, we superimposed it with a wave texture that is animated by an FFT-based simulation. Its coordinates are advected and blended using the algorithm presented in [Neyret 2003]. The entire scene is inspired by the results presented in [Losasso et al. 2008]. One of the main goals of our project was to show that it is possible to simulate such a complex scene in real time. Even though a direct comparison is not fair due to the fact that we used a coarser grid and faster hardware it is still worth mentioning that our simulation runs three to four orders of magnitude faster than what was reported in [Losasso et al. 2008].

To benchmark our solver we took the Incomplete Cholesky Preconditioned Conjugate Gradients method (IC(0) PCG) [Bridson 2008]) as a reference because it is the state of the art way to solve for incompressibility in fluid solvers. We ran our tests in double precision using a single CPU thread on an Intel Core i7 at 2.67 GHz with 4 GB of RAM. Since PCG is not applicable to non-symmetric systems, we did the comparison using only cubic cells grid without tall cells at three different resolutions, 64^3, 128^3, and 256^3. Our test scenario shown in Figure 6 is composed of a stream of water flowing past a solid sphere into a tank with three different water levels. We ran our tests with two different tolerances on the infinity norm of the residual: $10^{-4}\frac{1}{s}$ and $10^{-8}\frac{1}{s}$. The solver ran only full cycles with num_Pre_Sweep $=$ num_Post_Sweep $= 2$ (see Algorithm 3). The timings in seconds for various cases are shown in Table 3. Our

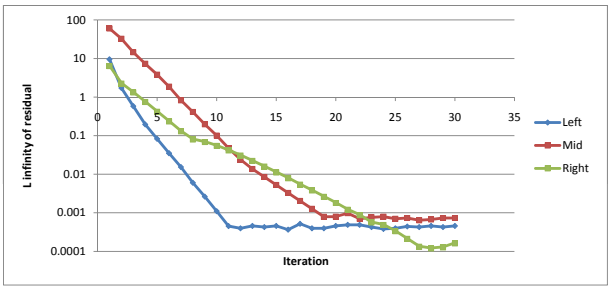

Figure 7: *Convergence rates for the three frames shown in Figure 1. The infinity norm of the residual is plotted on a logarithmic scale against the number of solver iterations. We use single precision floating point numbers on the GPU which explains why the error stops decreasing at some point.*

multigrid solver outperforms PCG in all scenarios except the one where the grid resolution and the water level are low, in which case they perform equally well. With increasing grid size, the speedup over PCG increases up to 14x at a resolution of 256^3 cells. The number of iterations required to reach a certain tolerance is almost constant regardless of the grid resolution, which is expected from the multigrid algorithm that has linear time complexity.

A fair comparison against [McAdams et al. 2010] is not feasible for practical cases because they do not handle the free liquid surface with sub-grid precision as we do. In other words, the two methods do not solve the same problem. In addition, in contrast to [McAdams et al. 2010], our solver runs stably on its own without an additional PCG loop. The latter requires global reduction with double precision floating point arithmetic [Bridson 2008], a step that would slow down a GPU implementation.

For a performance analysis of the general case with cubic and tall cells we used the three frames of the flood scene shown in Figure 1. Figure 7 shows the infinity norm of the residual on a logarithmic scale against the the number of multigrid iterations. The curves show that our solver reduces the error exponentially even for the asymmetric system derived from a tall cell grid. At some point, the error cannot be reduced any further and the curves reach a plateau. This is because we use single precision arithmetic with single precision floating point numbers on the GPU.

Using a co-located grid is one of the main reasons why our incom-

pressibility solver is simple and fast enough for real-time applications. However, since the divergence is only measured at the top and the bottom of tall cells, in the center, the solver is only aware of water flow in adjacent cubic cells, not inside the tall cell, which results in slight water gain over time. Even though this problem is not present in the staggered formulation of [Irving et al. 2006], we chose speed over accuracy in this trade off. To mitigate the problem, we make sure that the heights of adjacent tall cells do not differ too much, using parameter D in the remeshing step described in Section 3.6. This step reduces the chance that water flows into tall cells through their middle faces because they are not exposed to the regular cells. Note that smoothing the interface between the tall and the cubic cell regions does not smooth out the visual water surface. Note also that our pressure projection operator is not idempotent because the Laplacian is not a composition of gradient and divergence and hence may not eliminate divergence completely. This is not a problem in our real-time application but it could be problematic if very small divergence is required such as in off-line simulation.

64^3	IC(0) PCG				Multi-grid			
	Tol $=10^{-4}$		Tol $=10^{-8}$		Tol $=10^{-4}$		Tol $=10^{-8}$	
Cases	Iteration	Time	Iteration	Time	Iteration Full-cycle	Time	Iteration Full-cycle	Time
Low	57	0.75	92	1.21	9	0.75	16	1.31
Mid	97	1.35	156	2.18	8	0.68	14	1.18
High	124	1.92	198	3.06	7	0.61	12	1.02

128^3	IC(0) PCG				Multi-grid			
	Tol $=10^{-4}$		Tol $=10^{-8}$		Tol $=10^{-4}$		Tol $=10^{-8}$	
Case	Iteration	Time	Iteration	Time	Iteration Full-cycle	Time	Iteration Full-cycle	Time
Low	102	10.95	162	17.32	9	6.32	16	11.10
Mid	211	25.31	327	39.14	8	5.64	13	9.03
High	251	32.39	435	55.96	8	5.74	13	9.16

256^3	IC(0) PCG				Multi-grid			
	Tol $=10^{-4}$		Tol $=10^{-8}$		Tol $=10^{-4}$		Tol $=10^{-8}$	
Case	Iteration	Time	Iteration	Time	Iteration Full-cycle	Time	Iteration Full-cycle	Time
Low	217	183.31	334	281.83	7	39.49	11	55.89
Mid	450	424.27	675	635.03	7	39.77	12	67.06
High	523	542.32	918	951.08	8	46.08	12	68.04

Table 3: *Performance comparison between IC(0) PCG and our multigrid solver based on the three frames shown in Figure 6. The simulations were executed in a single CPU thread using double precision floating point numbers.*

5 Conclusion and Future Work

We have presented a method that is capable of simulating complex water scenes in real time. There are three main factors that speedup the solver to reach real-time performance. First, we use a specialized tall cell grid to focus computation time on areas near the surface, where the motion of the liquid is most interesting. Second, we devised an efficient multigrid solver that can handle the asymmetric systems resulting from such a hybrid grid. Third, we laid out the data structures and the algorithms to most efficiently use the compute power of modern GPUs.

In the future, we plan to investigate how to couple our 3D solver with a 2D height field solver in order to simulate even larger domains in real time. So far we focused on real-time simulations only. A next step would be to drop the real-time constraint and substantially increase the grid resolution. This will require a re-design of our data layout.

Acknowledgements

We would like to thank the members of the NVIDIA PhysX and APEX teams for their support and helpful comments. We also thank Aleka McAdams and Joseph M. Teran for provinding us with the source code of [McAdams et al. 2010] through their website.

References

ADALSTEINSSON, D., AND SETHIAN, J. A. 1997. The fast construction of extension velocities in level set methods. *Journal of Computational Physics 148*, 2–22.

ADAMS, B., PAULY, M., KEISER, R., AND GUIBAS, L. J. 2007. Adaptively sampled particle fluids. In *Proc. SIGGRAPH*, 48.

BARGTEIL, A. W., GOKTEKIN, T. G., O'BRIEN, J. F., AND STRAIN, J. A. 2005. A semi-lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*.

BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. In *Proc. SIGGRAPH*, 100.

BATTY, C., XENOS, S., AND HOUSTON, B. 2010. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. In *Proc. Eurographics*.

BRIDSON, R. 2008. *Fluid Simulation for Computer Graphics*. A K Peters.

BROCHU, T., AND BRIDSON, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing 31*, 4, 2472–2493.

BROCHU, T., BATTY, C., AND BRIDSON, R. 2010. Matching fluid simulation elements to surface geometry and topology. In *Proc. SIGGRAPH*, 1–9.

CARLSON, M., MUCHA, P. J., AND TURK, G. 2004. Rigid fluid: animating the interplay between rigid bodies and fluid. In *Proc. SIGGRAPH*, 377–384.

CHENTANEZ, N., AND MÜLLER-FISCHER, M. 2010. Real-time simulation of large bodies of water with small scale details. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

CHENTANEZ, N., GOKTEKIN, T. G., FELDMAN, B. E., AND O'BRIEN, J. F. 2006. Simultaneous coupling of fluids and deformable bodies. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 83–89.

CHENTANEZ, N., FELDMAN, B. E., LABELLE, F., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2007. Liquid simulation on lattice-based tetrahedral meshes. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 219–228.

COHEN, J. M., TARIQ, S., AND GREEN, S. 2010. Interactive fluid-particle simulation using translating eulerian grids. In *Proc. ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 15–22.

CRANE, K., LLAMAS, I., AND TARIQ, S. 2007. Real-time simulation and rendering of 3d fluids. In *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley Professional, August, ch. 30.

ENRIGHT, D., AND FEDKIW, R. 2002. Robust treatment of interfaces for fluid flows and computer graphics. In *Computer Graphics, 9th Int. Conf. on Hyperbolic Problems Theory, Numerics, Applications*.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. In *Proc. SIGGRAPH*, 736–744.

ENRIGHT, D., NGUYEN, D., GIBOU, F., AND FEDKIW, R. 2003. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *In Proc. 4th ASME-JSME Joint Fluids Eng. Conf., number FEDSM200345144. ASME*, 2003–45144.

FELDMAN, B. E., O'BRIEN, J. F., AND KLINGNER, B. M. 2005. Animating gases with hybrid meshes. In *Proc. SIGGRAPH*, 904–909.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. SIGGRAPH*, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graph. Models Image Process. 58*, 5, 471–483.

GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shells. In *Proc. SIGGRAPH*, 973–981.

HOLMBERG, N., AND WÜNSCHE, B. C. 2004. Efficient modeling and rendering of turbulent water over natural terrain. In *Proc. GRAPHITE*, 15–22.

IRVING, G., GUENDELMAN, E., LOSASSO, F., AND FEDKIW, R. 2006. Efficient simulation of large bodies of water by coupling two- and three-dimensional techniques. In *Proc. SIGGRAPH*, 805–811.

JEONG, W.-K., ROSS, AND WHITAKER, T. 2007. A fast eikonal equation solver for parallel systems. In *SIAM conference on Computational Science and Engineering*.

KIM, D., SONG, O.-Y., AND KO, H.-S. 2008. A semi-lagrangian cip fluid solver without dimensional splitting. *Computer Graphics Forum 27*, 2 (April), 467–475.

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. In *Proc. SIGGRAPH*, 820–825.

LENTINE, M., ZHENG, W., AND FEDKIW, R. 2010. A novel algorithm for incompressible flow using only a coarse grid projection. In *Proc. SIGGRAPH*, 114:1–114:9.

LONG, B., AND REINHARD, E. 2009. Real-time fluid simulation using discrete sine/cosine transforms. In *Proc. ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 99–106.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. In *Proc. SIGGRAPH*, 457–462.

LOSASSO, F., TALTON, J., KWATRA, N., AND FEDKIW, R. 2008. Two-way coupled sph and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics 14*, 4, 797–804.

MCADAMS, A., SIFAKIS, E., AND TERAN, J. 2010. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

MOLEMAKER, J., COHEN, J. M., PATEL, S., AND NOH, J. 2008. Low viscosity flow simulations for animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 9–18.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 154–159.

MÜLLER, M. 2009. Fast and robust tracking of fluid surfaces. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

NEYRET, F. 2003. Advected textures. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 147–153.

PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A. E., AND WHITAKER, R. T. 2003. Particle-based simulation of fluids. *Comput. Graph. Forum 22*, 3, 401–410.

RASMUSSEN, N., ENRIGHT, D., NGUYEN, D., MARINO, S., SUMNER, N., GEIGER, W., HOON, S., AND FEDKIW, R. 2004. Directable photorealistic liquids. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 193–202.

ROBINSON-MOSHER, A., SHINAR, T., GRETARSSON, J., SU, J., AND FEDKIW, R. 2008. Two-way coupling of fluids to rigid and deformable solids and shells. *ACM Trans. Graph. 27* (August), 46:1–46:9.

SANDERS, J., AND KANDROT, E. 2010. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional.

SELLE, A., FEDKIW, R., KIM, B., LIU, Y., AND ROSSIGNAC, J. 2008. An unconditionally stable MacCormack method. *J. Sci. Comput. 35*, 2-3, 350–371.

SIN, F., BARGTEIL, A. W., AND HODGINS, J. K. 2009. A point-based method for animating incompressible flow. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 247–255.

SOLENTHALER, B., AND PAJAROLA, R. 2009. Predictive-corrective incompressible sph. In *Proc. SIGGRAPH*, 1–6.

STAM, J. 1999. Stable fluids. In *Proc. SIGGRAPH*, 121–128.

TAKAHASHI, T., UEKI, H., KUNIMATSU, A., AND FUJII, H. 2002. The simulation of fluid-rigid body interaction. In *ACM SIGGRAPH conference abstracts and applications*, 266–266.

THÜREY, N., AND RÜDE, U. 2004. Free Surface Lattice-Boltzmann fluid simulations with and without level sets. *Proc. of Vision, Modelling, and Visualization VMV*, 199–207.

THÜREY, N., AND RÜDE, U. 2009. Stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids. *Computing and Visualization in Science 12 (5)*.

THUREY, N., MULLER-FISCHER, M., SCHIRM, S., AND GROSS, M. 2007. Real-time breakingwaves for shallow water simulations. In *Proc. Pacific Conf. on CG and App.*, 39–46.

ŠŤAVA, O., BENEŠ, B., BRISBIN, M., AND KŘIVÁNEK, J. 2008. Interactive terrain modeling using hydraulic erosion. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 201–210.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2010. Physics-inspired topology changes for thin fluid features. In *Proc. SIGGRAPH*, no. 4, 1–8.

YU, J., AND TURK, G. 2010. Enhancing fluid animation with adaptive, controllable and intermittent turbulence. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. In *Proc. SIGGRAPH*, 965–972.