Chimera Grids for Water Simulation

R. Elliot English* Stanford University
Linhai Qiu* Stanford University
Yue Yu* Stanford University
Ronald Fedkiw* Stanford University Industrial Light + Magic

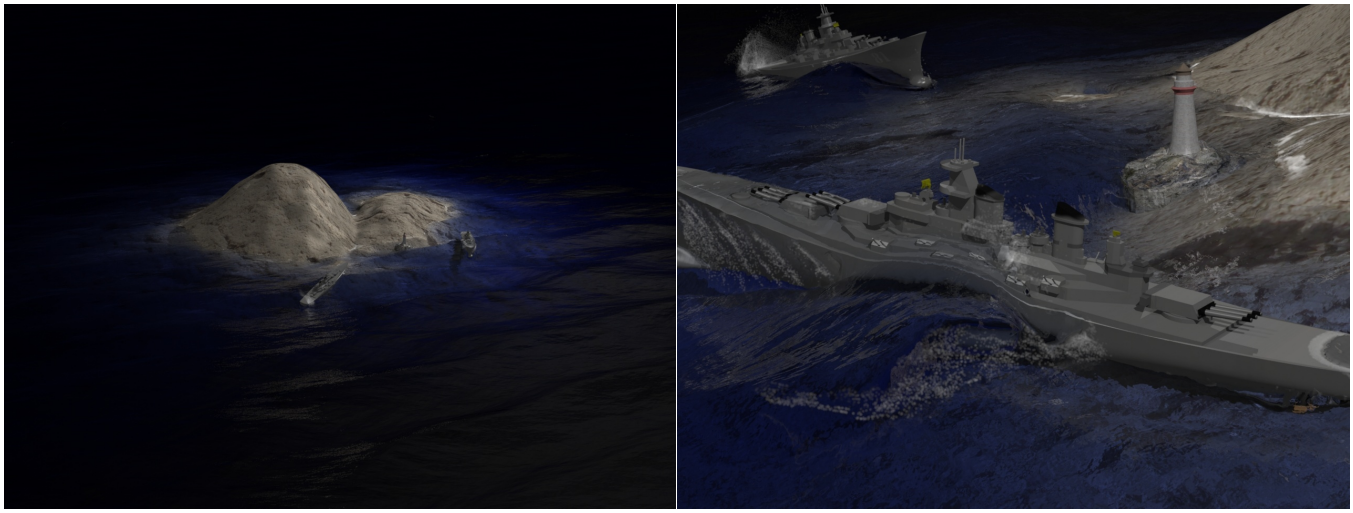

Figure 1: *Two ships in stormy seas near Longfellow island. We refine the domain near the ships by placing grids in their object spaces to add detail and allow them to propel themselves using their two-way solid-fluid coupled propellers.*

Abstract

We introduce a new method for large scale water simulation using Chimera grid embedding, which discretizes space with overlapping Cartesian grids that translate and rotate in order to decompose the domain into different regions of interest with varying spatial resolutions. Grids can track both fluid features and solid objects, allowing for dynamic spatial adaptivity without remeshing or repartitioning the domain. We solve the inviscid incompressible Navier-Stokes equations with an arbitrary-Lagrangian-Eulerian style semi-Lagrangian advection scheme and a monolithic SPD Poisson solver. We modify the particle level set method in order to adapt it to Chimera grids including particle treatment across grid boundaries with disparate cell sizes, and strategies to deal with locality in the implementation of the level set and fast marching algorithms. We use a local Voronoi mesh construction to solve for pressure and address a number of issues that arise with the treatment of the velocity near the interface. The resulting method is highly scalable on distributed parallel architectures with minimal communication costs.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling;

Keywords: chimera grids, adaptivity, water simulation

1 Introduction

Over the last decade the simulation of fluids has become an important part of computer animation due to the broad range of effects it can produce. Many of these advances have been due to increasing computational power as well as more efficient and accurate algorithms. However, the recent trend towards multiprocessor and multicore architectures, rather than increased sequential speed, has made numerical advances more difficult. While parallelism has been exploited, it has often been though the brute force parallelization of sequential algorithms resulting in simulations that lack the ability to reach larger scales.

In order to write efficient parallel software one has to carefully consider both how data is represented as well as the way algorithms access and use data — primarily due to the high cost of communication in parallel environments. Even with multiprocessor and multicore environments, data layout is critical in order to maximize cache coherency. Unstructured methods such as those using meshes made up of tetrahedra (see e.g. [Feldman et al. 2005; Klingner et al. 2006]) and particle based methods such as Smoothed-Particle Hydrodynamics (see e.g. [Desbrun and Cani 1996; Müller et al. 2003; Premoze et al. 2003]) in particular tend to have poor cache coherency due to their non-sequential data layout in memory. Unstructured methods have significant issues with domain decomposition due to the expensive and complicated partitioning algorithms necessary to evenly distribute the workload across processing nodes. Moreover, these methods typically require frequent "remeshing" forcing continuous re-partitioning and the movement of large amounts of data between processors.

In contrast, structured methods offer a lightweight implicit structure, cache coherent data layout, accurate numerical stencils, and straightforward domain decomposition. However, simply using a single uniform Cartesian grid is not often feasible since many simulations require the resolution of details on many different scales. Spatial adaptivity has played a critical role in allowing large scale and accurate simulations by permitting refinement in areas of interest such as at interfaces and near the camera. A number of structured adaptive methods have been developed within the graphics communities such as lattice based tetrahedral methods (see e.g. [Chentanez et al. 2007; Batty et al. 2010] – and [Molino et al. 2003; Labelle and Shewchuk 2007]) and octree methods (see e.g. [Losasso et al. 2004; Losasso et al. 2006]). Unfortunately these

*e-mail: {eenglish,lqiu,yuey,rfedkiw}@stanford.edu

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SCA 2013, July 19 – 21, 2013, Anaheim, California.
Copyright © ACM 978-1-4503-2132-7/13/07 $15.00

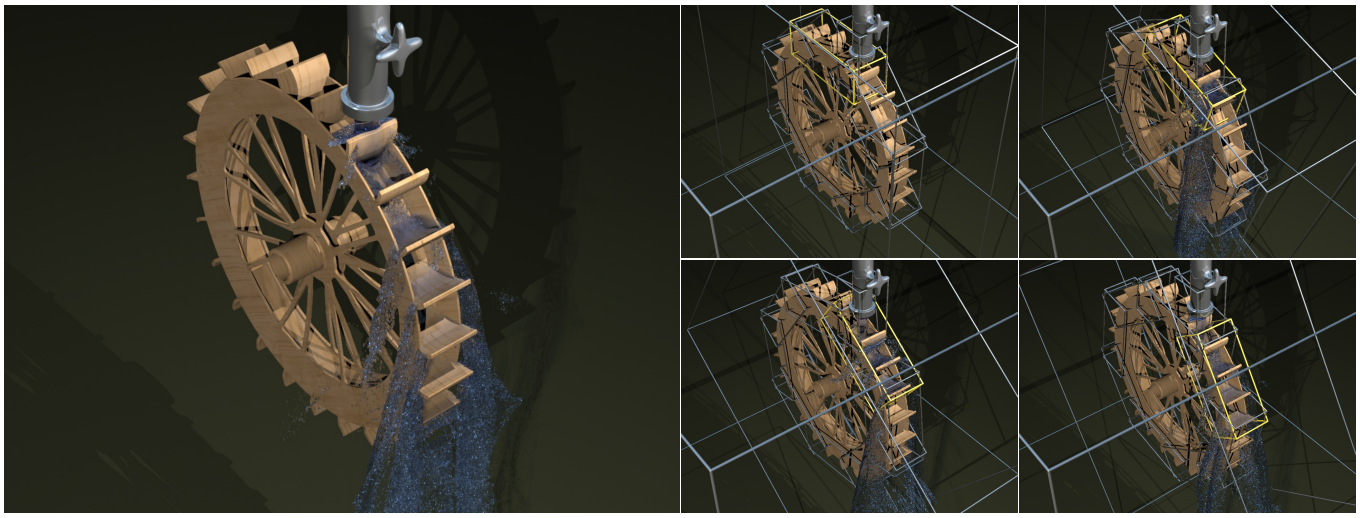

Figure 2: *Pouring water drives a water wheel via two-way solid-fluid coupling. The domain around the water wheel buckets is resolved by eight grids that rotate with the wheel in order to resolve the thin bucket walls. In addition, a single larger grid encases the entire wheel and also rotates in the same rigid body frame as the wheel.*

methods typically require complicated pointer and tree structures in order to efficiently store and access data limiting their ability to scale. Adaptive Mesh Refinement (AMR) methods (see e.g. [Berger and Oliger 1984; Berger and Colella 1989; Sussman et al. 1999]) exploit a block Cartesian grid structure by constructing an adaptive hierarchy of Cartesian grid patches. AMR uses axis-aligned grid placement in order to simplify implementation and numerical algorithms, however this requires a large number of small patches in order to efficiently resolve non-axis-aligned features. This abundance of small patches creates problems with domain decomposition and cache coherency similar to those for unstructured grids.

Because of the aforementioned issues with both unstructured methods and structured adaptive methods, we are motivated to pursue the use of Chimera grids (see e.g. [Benek et al. 1983; Henshaw 1994; Kiris et al. 1997]). Chimera grids are essentially a superset of AMR where grids can be arbitrarily oriented to better resolve non-axis-aligned features with a smaller number of grids. Axis-aligned AMR patches can be seen as a piecewise-constant approximation while rotated Chimera grids are a piecewise-linear approximation. Abstractly speaking, a Chimera style approach is not entirely new to the graphics community (see e.g. [Rasmussen et al. 2004; Shah et al. 2004; Patel et al. 2005; Dobashi et al. 2008; Tan et al. 2008; Cohen et al. 2010; Golas et al. 2012]), however each previous approach has been limited in either generality or scalability when compared to a full-blown Chimera grid scheme.

In this paper we extend the Chimera grid method of [English et al. 2012] to support water simulation by making a number of novel contributions to allow fluid to move from one grid to another while minimizing artifacts. In Section 3 we extend the particle level set method to overlapping grids. We address a number of issues related to reinitialization of the signed distance function after advection, enforcement of the free surface's continuity among the different the Cartesian grids and the Voronoi mesh, as well as extrapolation on the Voronoi mesh in order to provide valid signed distance values to the pressure solver. Particle handling at grid boundaries is also addressed. In Section 5 we discuss extending the Voronoi diagram pressure solver of [English et al. 2012] to free surfaces by using a second order accurate cut-cell free surface pressure condition, introduce an improved mapping of the velocities between the Cartesian grids and composite Voronoi mesh near the free surface, and extrapolating velocities across the free surface both on the Voronoi mesh and Cartesian grids. We extend the method to support monolithic two-way solid-fluid coupling which we demonstrate on a number of large scale problems in Section 6. Finally, we present a method for seamlesly rendering the free surface as the composite of the level sets on each grid in Section 8.

2 Chimera Grids

Chimera grid embedding uses a number of overlapping grids which are arbitrarily positioned, oriented and even of multiple types in order to decompose the domain into regions of interest. In many cases curvilinear grids which conform to an object's surface are used to model the boundary layer with great detail, while a regular Cartesian grid is used to capture the flow further away. This has allowed Chimera grid methods to be extremely effective in practical engineering applications. In fact, they were originally designed to simulate the compressible transonic flow regimes around the space shuttle using the Overflow solver [Benek et al. 1983].

Our Chimera grid approach builds upon the work of [English et al. 2012] discretizing space using a number of overlapping regular Cartesian grids with equal cell edge lengths undergoing rigid motion as illustrated in Figure 8. We emphasize that each Chimera grid can be thought of as a rigid body with a regular Cartesian grid stored in object space. In fact, our implementation uses a rigid body particle system to track each grid, while also allowing grids to be simply attached to existing rigid bodies. We define the transformation from a grid's object space to world space as $\vec{x}_{\text{world}} = \mathbf{R}\vec{x}_{\text{object}} + \vec{s}$ where $\vec{x}_{\text{world}}$ is the world space location, $\vec{x}_{\text{object}}$ is the object space location, $\mathbf{R}$ is the rotation, and $\vec{s}$ is the translation. Similarly, we transform vectors from object space to world space using $\vec{v}_{\text{world}} = \mathbf{R}\vec{v}_{\text{object}}$.

Chimera grids are extremely flexible and provide a convenient way to track flow features such as vortices and thin splashes as well as solid objects. We generally use a large background grid as can be seen in the bottom right of Figure 3. This gives a large simulation domain allowing waves to travel a long distance without hitting or reflecting off of the domain boundaries. In addition, we place smaller and finer grids near interesting features such as the drops and splashes in Figures 3 and 4. Note that in Figure 4 the grids follow the armadillo drops to track and preserve their shape

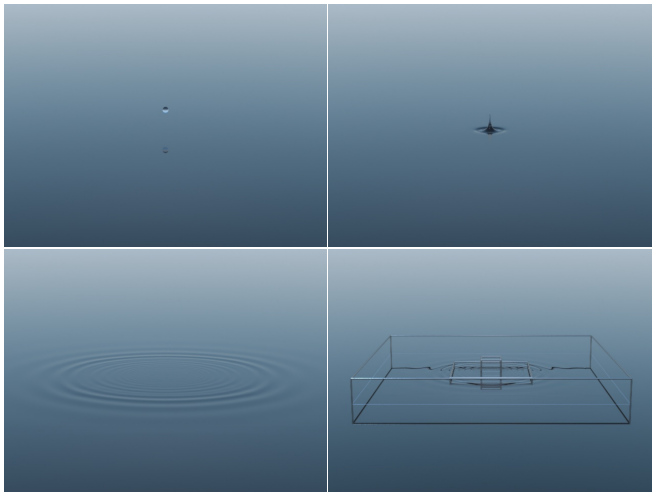

Figure 3: *A single water drop impacts a flat water surface generating several circular waves. The large domain allows these waves to propagate a long distance without reaching the domain boundaries. Note that the waves pass from one grid to the next without visible artifacts. The bottom right figure is the same as the bottom left except that we zoom the camera back even further in order to show the computational domain and the three grids used in the simulation.*

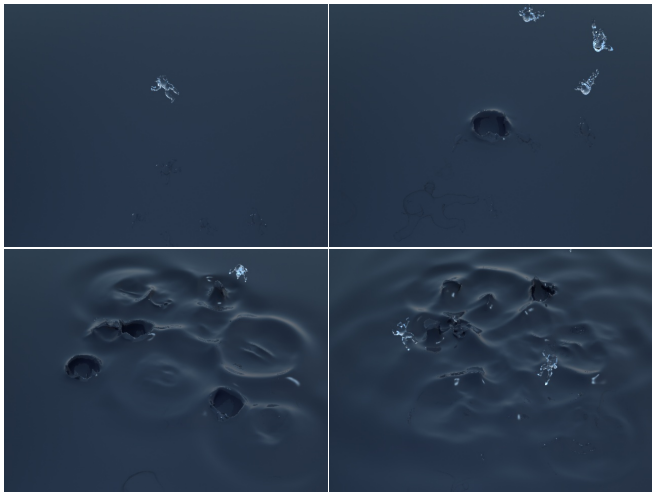

Figure 4: *Many water droplets in the shape of armadillos are dropped into still water. Each armadillo is enclosed by two moving grids: one tightly wrapped to resolve the armadillo as it falls and one slightly larger to catch the resulting splash. Our Chimera grid strategy allows detailed resolution of the small armadillo droplets even in this large domain.*

and detail. Chimera grids are also used to follow and track rigid objects as can be seen in Figure 7. The efficacy of our approach is further illustrated in Figures 2, 9, and 10, which show the resolution of thin features such as the buckets of a water wheel, the wings and tail of a glider, and the propellers on a ship.

Note that compared to certain unstructured and structured adaptive methods, we perform some duplicate computation in regions overlapped by more than one grid. However, assuming a reasonable grading and placement of grids, for most operations the overall cost of changing the method to avoid computing the solution in overlapped cells would exceed that of simply computing the solution throughout the entire domain. This is exacerbated by the domain decomposition used for parallelization. Furthermore, by operating on the entire grid we can exploit the single grid implementations of certain operations without modification, significantly reducing the implementation effort. The efficiency of our approach is emphasized by the large scale computation in Figures 1 and 11.

3 Level Set Method

Before directly addressing our full time evolution scheme we address representing and time evolving a free surface on overlapping grids and assume for now that the velocity field is given. To accomplish this, we follow the particle level set approach of [Enright et al. 2002a]. On each grid we store the level set ϕ values at cell centers as in the single grid case. At the beginning of each time step we fill the ϕ values in a number of layers of ghost cells along the exterior of each grid. For each ghost cell we interpolate a value using the finest overlapping grid with a valid interpolation stencil for that cell center. Since we are using a MAC grid discretization, this requires that the interpolation location lies within a grid's domain shrunken by half a grid cell.

Once we have filled the ghost cells, we advect the level set scalar field using the semi-Lagrangian approach of [Stam 1999; Enright et al. 2005] modified to support grid motion. In order to compute an updated ϕ value at an object space grid location $\vec{x}$, we trace a ray backwards in time from $\vec{x}$ using the time t^n velocity at $\vec{x}$. For simplicity, we first consider the case where a grid is stationary and give updated ϕ value at this location as $\phi^{n+1}(\vec{x}) = \phi^n(\vec{x} - \Delta t \vec{u}^n(\vec{x}))$ where Δt is the time step and $\vec{u}^n$ is the time t^n *object space* velocity. Note that the entire operation is performed in the object space of the grid, and the update equation is identical to that of standard semi-Lagrangian advection. We next consider the case of a moving grid. To update a specific grid point in object space, $\vec{x}_{\text{object}}$, we first compute where that point will be in world space at time t^{n+1}, $\vec{x}^{n+1}_{\text{world}} = \mathbf{R}^{n+1}\vec{x}_{\text{object}} + \vec{s}^{n+1}$, and then we put this world space point back into object space at time t^n in order to interpolate the velocity as $\vec{u}_{\text{fluid}} = \vec{u}^n((\mathbf{R}^n)^{-1}(\vec{x}^{n+1}_{\text{world}} - \vec{s}^n))$. This velocity then needs to be augmented by the grid motion obtaining an effective velocity of $\vec{u}_{\text{effective}}(\vec{x}_{\text{object}}) = \vec{u}_{\text{fluid}} - \vec{u}_{\text{grid}}$ where $\vec{u}_{\text{grid}} = (\mathbf{R}^{\mathbf{n}})^{-1}(\vec{x}^{n+1}_{\text{world}} - \vec{x}^n_{\text{world}})/\Delta t$ and $\vec{x}^n_{\text{world}} = \mathbf{R}^n\vec{x}_{\text{object}} + \vec{s}^n$. We then compute the final value at $\vec{x}_{\text{object}}$ as follows:

$$\phi^{n+1}(\vec{x}_{\text{object}}) = \phi^n(\vec{x}_{\text{object}} - \Delta t \vec{u}_{\text{effective}}) \tag{1}$$

A loose time step restriction is necessary in order to guarantee that $(\mathbf{R}^n)^{-1}(\vec{x}^{n+1}_{\text{world}} - \vec{s}^n)$ is in the interior of the grid and its accompanying ghost cell layers so that we can interpolate $\vec{u}^n$. As described in [English et al. 2012] we use several layers of ghost cells to allow for grid motion in addition to layers used for advection. Note that like generalized semi-Lagrangian methods Equation 1 follows the same world space path as standard semi-Lagrangian advection.

After advection, we again fill the ghost cells before reinitializing the interface independently on each grid using the fast marching method and higher order interface initialization as described in [Losasso et al. 2006]. Note that we fill enough ghost cell layers to cover the bandwidth used by the fast marching method. This allows us to march over both the interior and ghost cells in a single process and achieve the correct result on the interior of the domain. When using the fast marching method, which marches only in a limited band near the interface, the stopping distance on the fine grid can be too small to guarantee a valid interpolation stencil for filling overlapped cells. However, this only occurs when the grading between grids is large and can be avoided by increasing the stopping distance on the fine grid. In order to prevent the level set representations from drifting apart in overlapped regions on differ-

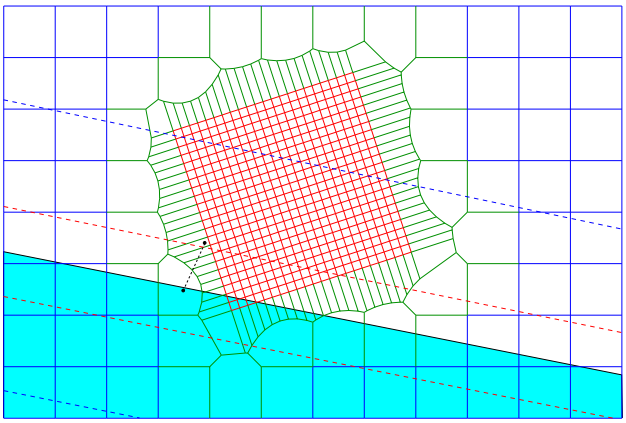

Figure 5: *A free surface is shown overlaying our Voronoi pressure discretization. When reinitializing the level set after advection, values are computed for only a limited band of cells around the interface. The limits of these bands are drawn as dashed lines (with the color indicating the corresponding grid). Due to the unstructured connectivity between grids on the Vonoroi mesh, cells on coarse grids along the interface may be adjacent to cells on the fine grid outside of the reinitialization bandwidth as illustrated by the two cells with black dots at their cell centers. In order to use a second order accurate free surface boundary condition we compute an updated ϕ value for the cell on the fine grid by linearly extrapolating from the cell on the coarse grid.*

ent grids, we inject the ϕ values in cells which are overlapped by a finer grid. When filling ghost cells we interpolate from the finest grid with a valid interpolation stencil. After injecting to these cells we need to perform a second reinitialization step in order to guarantee valid values through to the stopping distance. This time, after filling the ϕ values in ghost cells, we reinitialize the level set values in the overlapped cells only in order to minimize dissipation.

We briefly look forward to the pressure solver described in Section 5. In order to apply a second order accurate free surface boundary condition in the solver, a valid level set value is required at each cell along the free surface. Due to the limited bandwidth used in the fast marching method and the unstructured connectivity between grids on the Voronoi mesh used in the pressure solve (as depicted in Figure 8), certain coarse grid cells may neighbor fine grid cells which do not have valid ϕ values as illustrated in Figure 5. For each of these cells, we extrapolate a ϕ value from a neighboring cell on the Voronoi mesh by computing $\nabla\phi$ at this neighboring cell using values from this cell's intrinsic Cartesian grid. The neighboring cell must both have a valid value of ϕ and a valid value of $\nabla\phi$, and if more than one neighbor has valid information we use the neighbor with the smallest value of $|\phi|/\Delta x$ representing the value closest to the interface with the most accurate information. After computing these updated ϕ values, we interpolate values for any Cartesian cell centers that were removed during the Voronoi mesh construction as well as a one cell thick layer of ghost cells. These cells are filled using barycentric interpolation on the Delaunay triangle mesh dual of the Voronoi diagram. Without this interpolation step to enforce consistency between the Voronoi mesh and the Cartesian grid level set representations, water could become stuck along grid boundaries on the coarse grid since the fluid in these cells does not directly influence the pressure solution. We perform a final reinitialization step in these interpolated cells in order to guarantee that they are valid through to the stopping distance.

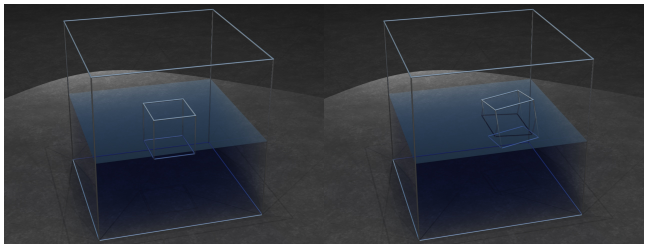

Figure 6: *A kinematically driven grid moves in and out of still water without disturbing the surface. Throughout the entire simulation the water velocity properly remains within rounding tolerance of zero. The left figure shows the initial state and the right figure shows the water surface unchanged after 600 frames.*

3.1 Particle Level Set Method

We store a number of layers of particles along the free surface as is typical in the single grid case. After filling the ghost cells we seed particles in each ghost and overlapped cell using the updated ϕ values. While one could instead exchange particles in this step, seeding new particles in ghost cells each time step prevents large particles originating from a coarse grid from adversely effecting the level set function on a fine grid. Following seeding, the majority of the particle advection and level set correction steps remain the same as those in the single grid case. We advect the particles using a second order accurate Runge-Kutta scheme. Subsequently, the particles are used to correct the ϕ values both after advection and after the first reinitialization step only. Recall that the first reinitialization modifies the entire grid while the second reinitialization only modifies overlapped regions.

At the end of each time step we flag any particles that have crossed the interface by more than their radius (or a small threshold) and designate these as "removed particles". This operation is performed in both the ghost and interior regions of each grid. We discard any removed particles that are within the interpolation stencil of an overlapped interior cell. These particles in overlapped regions represent underresolved features on the coarse grid which are simulated more accurately on the fine grid. We keep the removed particles generated in the ghost regions of each grid as they allow us to track the mass loss which can occur at grid boundaries. This approach allows a thin feature on a fine grid to move onto a coarse grid and be tracked as ballistic or passively advected particles for subsequent rendering, instead of simply having the feature disappear at a grid boundary. Finally, before ending the time step, we delete all the remaining non-removed particles in the ghost regions of each grid.

4 Navier-Stokes Equations

We now proceed to describe the remainder of our time evolution scheme. The inviscid incompressible Navier-Stokes equations are given as follows:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = -\frac{1}{\rho}\nabla p + \vec{f} \tag{2}$$

$$\nabla^T \vec{u} = 0 \tag{3}$$

where ρ is density, p is pressure, and $\vec{f}$ are external and body forces scaled by ρ. We solve these equations using a splitting method where we first solve

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} + (\mathbf{u}^n \cdot \nabla)\mathbf{u}^n = \mathbf{f} \tag{4}$$

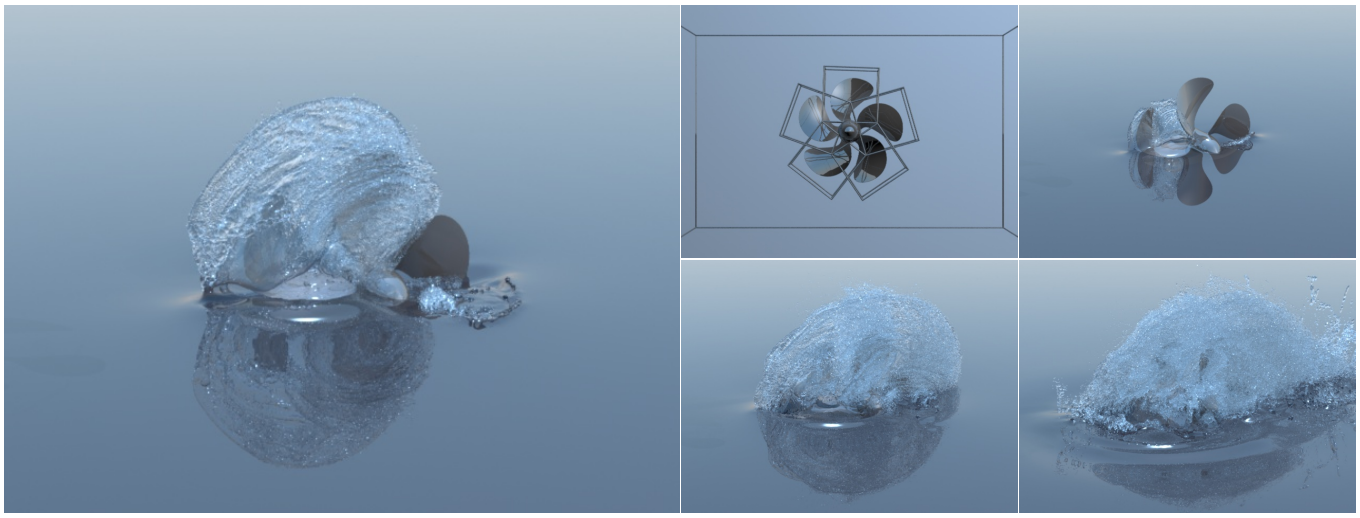

Figure 7: *A kinematically driven propeller spins with a number of attached Chimera grids. The grids move and rotate with the rigid body frame of the propeller.*

for $\mathbf{u}^*$. Since we use a MAC grid representation, vector fields are stored as scalar normal components at cell faces. We note that the component stored at each face corresponds to the component of the vector field in the world space normal direction of that face. Thus in order to compute a world space velocity on a specific grid, we first interpolate the velocity vector in object space and then rotate that vector into world space. In order to advect the velocity field we apply the arbitrary-Lagrangian-Eulerian style semi-Lagrangian scheme described in Section 3. Since grids are not aligned, it is necessary to interpolate a full world space velocity vector at each face and then take only the component along the face's normal direction when filling ghost cell velocity values. Similarly during advection, we advect a full velocity vector to each time t^{n+1} face and then rotate these vectors from the grid's time t^n frame into the grid's time t^{n+1} frame before taking the scalar component in each faces' normal direction.

Finally we solve for the pressure using

$$\nabla^2 \hat{p} = \rho \nabla^T \mathbf{u}^* \tag{5}$$

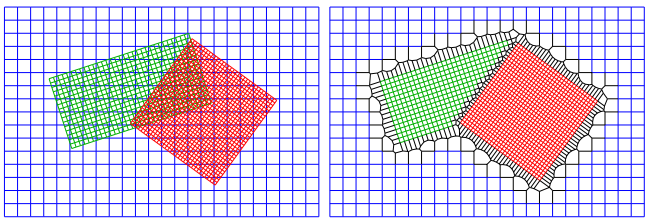

Figure 8: *The left figure shows three grids laid out in typical fashion with one grid covering the entire domain and two smaller grids resolving interesting features and/or objects. The right figure shows the Voronoi mesh used in our pressure solver in order to determine which degrees of freedom interact with one another as well as the resulting stencils.*

where $\hat{p}$ is the Δt scaled pressure, before updating the velocity field via

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\nabla \hat{p}}{\rho} \tag{6}$$

5 Pressure Solver

Many Chimera grid methods use a Gauss-Seidel approach (see e.g. [Dobashi et al. 2008]) to solve Equation 5 for the pressure, iteratively computing boundary conditions while solving independently on each grid. Convergence is slow, if it occurs at all, and for even moderately sized systems it is not feasible to iterate to convergence. Monolithic solvers have been devised by incorporating boundary condition interpolation operators into the system itself [Henshaw 1994], however this strategy produces an asymmetric system which is difficult to solve and also has convergence issues requiring a more complicated solver or preconditioner. Although cut cell methods produce symmetric positive definite discretizations, they are generally only first order accurate resulting in spurious errors in the velocity field that do not vanish under grid refinement. Explicit and implicit deferred correction methods [Traoré et al. 2009; Chang and Yuan 2012] have been used to address these errors, however they also suffer from convergence issues. Water surfaces are particularly sensitive to the approximations made during discretization, and we found that it was essential to solve the hydrostatic case exactly as illustrated in Figure 6 in order to prevent spurious vortices from forming at grid boundaries. Therefore we leverage the monolithic symmetric positive definite discretization for solving Poisson equations on arbitrarily translated and rotated Cartesian grids from [English et al. 2012].

Similar to the pressure solvers of [Sin et al. 2009] and [Brochu et al. 2010], a Voronoi diagram is used to construct a staggered velocity-pressure discretization as illustrated in Figure 8. Proceeding in order from the finest grid to the coarsest, we remove both cells which contain the cell center of a finer non-removed cell and cells whose cell center lies within a finer non-removed cell. We then construct the Voronoi mesh for the non-removed cells. For each non-removed cell along an intergrid boundary, we first find all neighboring non-removed cell centers within a specified distance τ. To improve performance, we compute τ for each cell center by finding the finest grid whose domain shrunken by $|\Delta\vec{x}|$ contains the cell center and then set $\tau = 2|\Delta\vec{x}|$. Then for each neighboring cell, we construct the perpendicular bisecting plane between the original cell center and the neighboring cell center clipping this plane against planes similarly formed with the other neighboring cell centers. If the result is not empty, it is a convex polygon which becomes the Voronoi face between these cells.

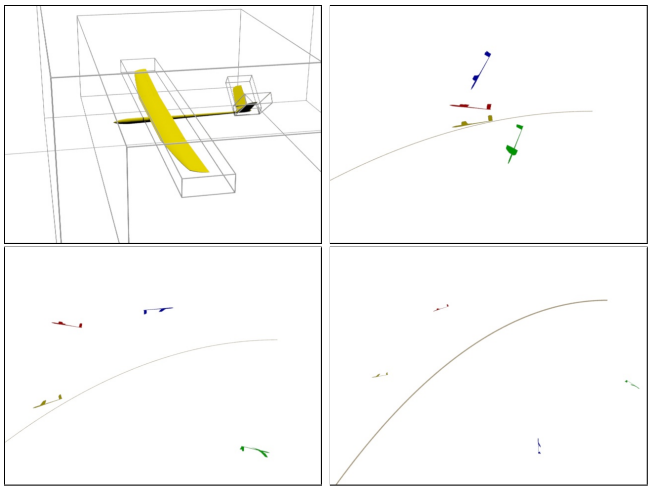

Figure 9: *Resolving a three dimensional glider (yellow) with a number of Chimera grids gives it the ability to fly in a large domain. The solid line shows the parabolic path a ballistic rigid body would take if it did not fly. The blue glider is simulated with only the background grid and falls almost immediately. Meanwhile, the red and green gliders each use only two of the six grids attached to the yellow glider. Although the green glider crashes, for illustration purposes, we can carefully adjusted the position and orientation of the red glider's two grids allowing it to fly. Note that this hand-picked rasterization will remain constant throughout the entire simulation since the grids move with the glider enabling the user to handcraft efficient rasterizations that improve the physical realism. We emphasize to the reader that we did not specially tune the positions of the grids in the case of the yellow glider as it was well resolved by its finer grids. Slight perturbations to the placement of its grids did not alter the glider's behavior.*

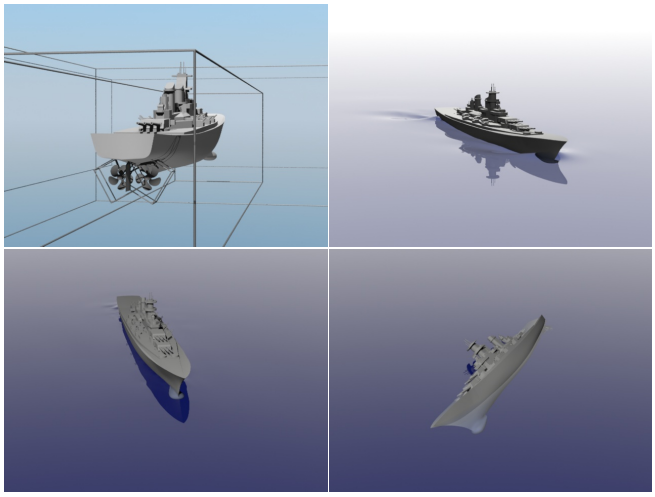

Figure 10: *The top two figures show a two-way coupled rigid body ship that moves under its own power due to the rotation of two articulated propellers. The lower left figure illustrates that removing the grids surrounding the propellers limits the realism of the simulation resulting in a ship that churns water but cannot propel itself forward. The lower right figure shows that removing the grids around the ship results in a loss of buoyancy and the ship falls through the large grid sinking to the bottom of the ocean.*

For each Voronoi mesh face not coincident with a Cartesian grid face, [English et al. 2012] interpolated $\mathbf{u}^*$ from the finest available Cartesian grid. In certain cases, we found that these interpolated velocities caused noticeable artifacts at grid boundaries. This was particularly evident when the velocity for a Voronoi face between to two fine grid cells was interpolated from a much coarser grid resulting in large error in the velocity. In order to avoid this problem, we first compute the cell center velocity at each adjacent cell centers by averaging on their respective Cartesian grids before taking an average of these cell center velocities to compute the Voronoi mesh face velocity. We further modify this in the case of Voronoi faces along the free surface as follows. Similar to the issues described for ϕ values in Section 3 along boundaries between coarse and fine grids, certain cells may not have valid velocities since they are beyond the free surface velocity extrapolation bandwidth. Hence, for these faces we take the velocity from the incident water cell rather than averaging from both cells. We found that these modifications produce a much more accurate velocity and do not cause any noticeable artifacts along grid boundaries.

In order to handle free surfaces, we set a constant Dirichlet pressure boundary condition in air cells and use a second order accurate cut-cell approach [Enright et al. 2003; Batty et al. 2010] at each face between an air cell and a water cell. This is straightforward to apply recalling that we ensured a valid ϕ value in each cell through extrapolation at the end of Section 3. After computing the pressure, we update each face on the Voronoi mesh including all Cartesian grid faces coincident to faces of the Voronoi mesh. In order to update Cartesian grid faces not coincident to a Voronoi face, we first compute a full velocity vector at the center of every water cell in the Voronoi mesh. This requires a local least squares fit of the face values for irregular cells. Next, we extrapolate cell velocities across the level set water interface on the Voronoi mesh in order to guarantee a valid interpolation stencil for removed water cells. This extrapolation is accomplished using two Jacobi style iterations across the entire mesh where the velocity is computed by averaging the velocities from all adjacent cells with valid velocities. Then the velocity for each removed cell is computed using standard multilinear interpolation in the Cartesian regions and barycentric interpolation over the Delaunay triangle dual mesh elsewhere. Finally, Cartesian grid face velocities are computed by averaging from the neighboring Cartesian grid cell centers using only a one sided average in the case that only one of its neighbors is a water cell (with the other being air).

After updating each water face, we fill the ghost cells and extrapolate velocities into the air using the fast velocity extrapolation of [Enright et al. 2002b]. Note that we may not be able to interpolate a valid velocity for every ghost face depending upon the discretization of the overlapping grid. We handle this case by updating the invalid ghost cell faces in the extrapolation procedure as if they were air faces.

In our tests we found that we were computationally bound and that communication costs were relatively minimal compared the other parts of the code. However, as we scale up to more processors the preconditioned conjugate gradient method used to solve the two-way solid-fluid coupled Poisson equation system will become the bottleneck especially due to the frequent communication costs. We applied a block diagonal preconditioner in which we partition the variables in two sets corresponding to the pressures, and solid-fluid coupling/articulation Lagrange multipliers. The diagonal block corresponding to the pressures was preconditioned using an incomplete Cholesky (0-fill) factorization and the block corresponding to the lagrange multipliers was diagonally preconditioned. When computing in parallel, the incomplete Cholesky preconditioner was applied independently on each computational node to the diagonal block corresponding to the local pressures only. We did not experi-

ence any significant drop in the effectiveness of the preconditioner as a result. However, we observed that without the diagonal preconditioning of the Lagrange multiplier block, the residual could remain large in the parts of the equations that represent the solid-fluid coupling even though the incompressibility of the fluid is adequately enforced. In fact, the method would sometimes not even converge, resulting in spurious behavior including objects incorrectly sinking. However, this problem is not a result of our pressure discretization in particular and occurs with many monolithic solid-fluid coupling methods as the resolution is increased. On the other hand, we experienced no convergence issues or spurious behavior after applying the diagonal preconditioner.

6 Object Interaction

As shown in Figures 1, 2, and 7 – 11 we often attach grids to solid objects. Since each grid simply moves with its object preserving the relative placement of the fluid degrees of freedom with respect to the object interface, the object only needs to be rasterized once at the beginning of a simulation. This allows for a greater level of user control in crafting the initial rasterization (see the note about the red plane in Figure 9), especially as far as rasterized normals, watertightness, etc. are concerned.

After filling the level set ghost cells at the beginning of the time step, we extrapolate the level set into objects using the fast marching method. Similarly after filling the velocity ghost cells, we set the velocity on each face inside an object to the pointwise object velocity. In the case of kinematic objects, we set a Neumann boundary condition on each occluded face of the Voronoi mesh. In the two-way coupled case, we follow the approach of [Robinson-Mosher et al. 2011] using a constraint equation and Lagrange multiplier force for each occluded face of the Voronoi mesh in order to enforce equality between fluid and solid velocities.

In order to compute the time step, we assume that objects move with constant linear and angular velocity in order to approximate the time t^{n+1} locations of the grids bound to these objects. However, the actual displacement during a time step will be affected by acceleration under gravity and buoyancy, interaction with other objects through contact, collision, and articulation, etc. Thus, the actual motion of the grid may exceed that predicted by the time step computation at the beginning of the time step. We handle this by filling additional ghost cell layers before advection and found the added cost to be small.

7 Pseudocode

We now summarize our time evolution scheme which is derived from the Newmark scheme used in [Robinson-Mosher et al. 2008]. Note that we use forward Euler in the solid position step in order to eliminate the additional solid-fluid coupled solve for efficiency. The resulting scheme is as follows:

1. Compute time step
2. Integrate solid positions using time t^n velocities
3. Update grid positions
4. Fluid advection step (§ 3 and § 3.1)
   (a) Fill ϕ and velocity ghost cells, and reseed particles in ghost cells and overlapped regions
   (b) Advect ϕ
   (c) Advect particles
   (d) Modify the levelset by particles independently on each grid
   (e) Fill ϕ ghost cells and reintialize ϕ independently on each grid
   (f) Modify the levelset by particles independently on each grid again
   (g) Inject ϕ values into cells overlapped by finer grids and reintialize ϕ in injected cells
   (h) Advect velocities
   (i) Inject velocities into cells overlapped by finer grids
5. Integrate solid velocities using gravity and contact forces
6. Integrate fluid velocities using gravity
7. Construct Voronoi mesh (§ 5)
8. Extrapolate ϕ values on Voronoi mesh along the free surface at grid boundaries (§ 5)
9. Interpolate ϕ values in removed cells using Delaunay mesh and reinitialize interpolated cells (§ 3)
10. Pressure projection step (§ 5 and § 6)
    (a) Compute Voronoi mesh face velocities
    (b) Construct pressure Poisson system and solve for pressure and constraint Lagrange multipliers
    (c) Update velocities on Voronoi mesh and at coincident Cartesian faces
    (d) Extrapolate velocities on Voronoi mesh
    (e) Interpolate cell centered velocities using multilinear and barycentric interpolation and update remaining Cartesian face velocities
    (f) Integrate solid velocities using constraint Lagrange multipliers
11. Fill velocity ghost cells and extrapolate velocities independently on each grid
12. Remove escaped particles, exchange removed particles between grids and delete particles in overlapped regions, ghost cells and inside solids (§ 3.1)

8 Results

In order to render the free surface without visible cracks at grid boundaries we modify the raycasting approach used in [Enright et al. 2002b] to intersect a composite signed distance function. For each sample location the composite signed distance function is computed by considering each overlapping grid in order from coarsest to finest and blending the ϕ value from the current grid with the ϕ value computed from blending the coarser grids. The blending fraction is found by scaling and clamping the signed distance function to the grid boundary so that the fraction is 0 at the edge of the grid's interpolation domain and 1 several cells within the grid. While the resulting composite ϕ function is no longer a strict signed distance function, the implicit surface defined at $\phi(\vec{x}) = 0$ is smooth and by blending the normals in a second step using the same procedure, rendering artifacts are avoided. In Figure 7 we additionally render the removed negative particles as transparent spray by directly ray tracing the particles as metaballs which are blended with the level set. Additionally in Figure 7 when a removed negative particle falls back into the water we reincorporate it into simulation and apply a small impulse to the fluid velocity so that the particle does not simply disappear without affecting the water surface. In Figures 1 and 11 we render the removed negative particles as a density field defined as the sum of kernel functions centered at each particle. Additionally, a Phillips spectrum was used to bump map the water to give it added detail as described in [Tessendorf 2002].

We provide relative timing data for the propeller and island examples in Table 1. Note that while the overhead of the added second grid is significant, it does not dominate the simulation time and improves under refinement. Further optimization of the implementation could easily reduce this overhead. In the example shown in

	Entire time step	Filling ghost and overlapped cells	Reseed particles	Semi-Lagrangian and particle Advection	Construct mesh	Projection	Extrapolate velocities	Delete particles
island, 1 proc	914	23	41	157	318	310	12.3	51
island, 39 procs	85	3.3	5.2	17	24	29	2.2	4.2
propeller, 1 proc	299	7.2	8.5	50	114	109	5.2	2.7
propeller, 23 procs	47	1.8	.87	8.7	17	17	1.5	.19

Table 1: *Timing data (in seconds) for the island and propeller examples. The table includes timing only for the major steps of the algorithm.*

Figures 1 and 11, 17 grids were used with a total of 6 million cells. By extending the finest cell's Δx to the background grid, the effective resolution of the simulation was $36500 \times 3650 \times 36500$. While the effective resolution could be made arbitrarily large by decreasing the size of the smallest cell, it does give a sense of the disparate scales being resolved.

9 Conclusion

We proposed a novel spatially adaptive method for simulating free surface incompressible flows using Chimera grid embedding. We included a number of examples which demonstrated the ability of the method to produce accurate physical behavior in two-way solid-fluid coupling problems by locally refining space near solid objects. There is considerable future work to be done, such as developing a conservative advection scheme and extending the method to a FLIP based solver. Additionally, improving the velocity mapping between the Cartesian grids and the Voronoi mesh in order to eliminate higher order artifacts in the velocity field (noted as vorticity artifacts in [English et al. 2012]) which could limit the use of vorticity confinement is an important problem to be addressed. Nevertheless, we noticed no visible artifacts in the free surface due to this issue. However, as in many other adaptive scheme, if the grading was too large (by factors of approximately 6 or more), slight reflection of waves could be seen at boundary, although this was straightforward to avoid by using multiple enclosing grids to achieve an acceptable grading. Increasing the scope of dynamic adaptivity beyond simply following objects also holds significant promise — one could have grids follow vortices and other turbulent features in addition to objects by tracking. Overall, although our approach loses some flexibility while compared to fully unstructured methods, it strikes an optimal balance between structure and adaptivity, allowing large scale simulations with features on many scales to be concurrently simulated while utilizing modern computer hardware and distributed computing resources. Figures 1 and 11 demonstrate the potential of the method.

Acknowledgements

We would like to thank Wen Zheng for helping render several examples. We would like to thank Jure Leskovec and Christos Kozyrakis for additional computing resources as well as Andrej Krevl and Jacob Leverich for helping us use those resources. R.E.E. was supported in part by a Stanford Graduate Fellowship and an NSERC Postgraduate Scholarship. Y.Y. was supported in part by a Stanford Graduate Fellowship. Research was supported in part by ONR N00014-09-1-0101, ONR N-00014-11-1-0027, ONR N00014-11-1-0707, ARL AHPCRC W911NF-07-0027, and the Intel Science and Technology Center for Visual Computing. Computing resources were provided in part by ONR N00014-05-1-0479.

References

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

BENEK, J. A., STEGER, J. L., AND DOUGHERTY, F. C. 1983. A flexible grid embedding technique with applications to the euler equations. In *6th Computational Fluid Dynamics Conference*, AIAA, 373–382.

BERGER, M., AND COLELLA, P. 1989. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys. 82*, 64–84.

BERGER, M., AND OLIGER, J. 1984. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys. 53*, 484–512.

BROCHU, T., BATTY, C., AND BRIDSON, R. 2010. Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 47:1–47:9.

CHANG, L., AND YUAN, G. 2012. An efficient and accurate reconstruction algorithm for the formulation of cell-centered diffusion schemes. *J. Comput. Phys. 231*, 20, 6935–6952.

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

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

DESBRUN, M., AND CANI, M.-P. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Comput. Anim. and Sim. '96 (Proc. of EG Wrkshp. on Anim. and Sim.)*, Springer-Verlag, R. Boulic and G. Hegron, Eds., 61–76.

DOBASHI, Y., MATSUDA, Y., YAMAMOTO, T., AND NISHITA, T. 2008. A fast simulation method using overlapping grids for interactions between smoke and rigid objects. *Comput. Graph. Forum 27*, 2, 477–486.

ENGLISH, R., QIU, L., YU, Y., AND FEDKIW, R. 2012. An adaptive discretization of incompressible flow using a multitude of moving cartesian grids. *(submitted)*.

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys. 183*, 83–116.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3, 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 *Proc. 4th ASME-JSME Joint Fluids Eng. Conf., number FEDSM2003–45144. ASME*.

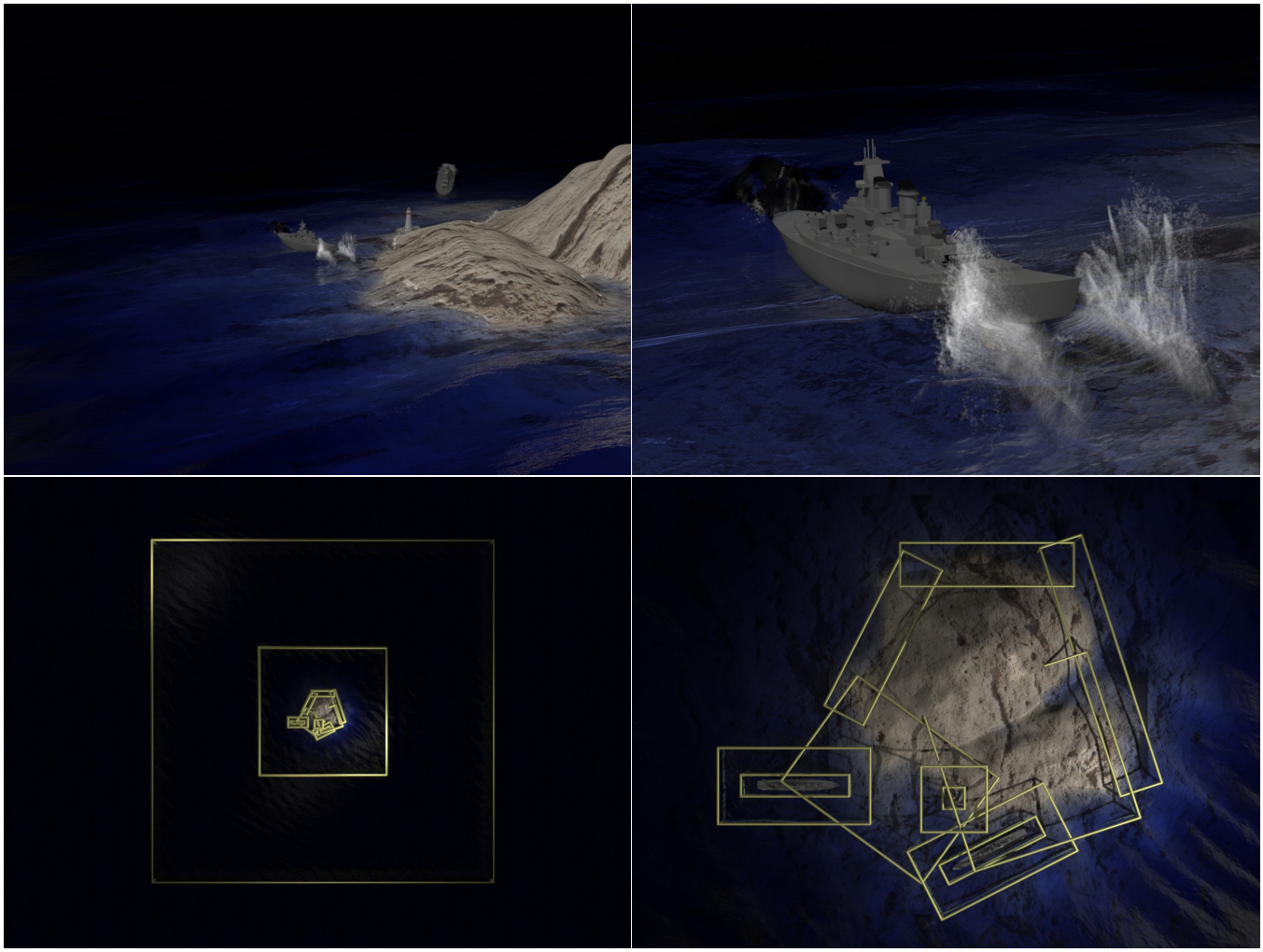

Figure 11: *Another view from the large scale island example in Figure 1. Each ship is driven by two propellers and uses the same attached grids as in the case of the single ship example in Figure 10. We initialize the fluid surface and velocity with a large number of rolling waves which slowly move across the domain. As a result of the large domain, the simulation does not require seeding additional waves using boundary conditions or external forces within the time scale of the simulation. The top two figures show the spray produced by the ship propellers when they are exposed on the surface of the water, noting that both figures are of the same frame. The lower two figures show the initial grid placement around the ships and island.*

ENRIGHT, D., LOSASSO, F., AND FEDKIW, R. 2005. A fast and accurate semi-Lagrangian particle level set method. *Computers and Structures 83*, 479–490.

FELDMAN, B., O'BRIEN, J., KLINGNER, B., AND GOKTEKIN, T. 2005. Fluids in deforming meshes. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 255–259.

GOLAS, A., NARAIN, R., SEWALL, J., KRAJCEVSKI, P., DUBEY, P., AND LIN, M. 2012. Large-scale fluid simulation using velocity-vorticity domain decomposition. *ACM Trans. Graph. 31*, 6, 148:1–148:9.

HENSHAW, W. D. 1994. A fourth-order accurate method for the incompressible Navier-Stokes equations on overlapping grids. *J. Comput. Phys. 113*, 1, 13–25.

KIRIS, C., KWAK, D., ROGERS, S., AND CHANG, I.-D. 1997. Computational approach for probing the flow through artificial heart devices. *Journal of biomechanical engineering 119*, 4, 452–460.

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. *ACM Trans. Graph. (SIGGRAPH Proc.) 25*, 3, 820–825.

LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*, 3 (July).

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (SIGGRAPH Proc.) 23*, 457–462.

LOSASSO, F., FEDKIW, R., AND OSHER, S. 2006. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids 35*, 995–1010.

MOLINO, N., BRIDSON, R., TERAN, J., AND FEDKIW, R. 2003. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *12th Int. Meshing Roundtable*, 103–114.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 154–159.

PATEL, S., CHU, A., COHEN, J., AND PIGHIN, F. 2005. Fluid simulation via disjoint translating grids. In *ACM SIGGRAPH 2005 Sketches*, ACM, New York, NY, USA, SIGGRAPH '05.

PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. 2003. Particle–based simulation of fluids. In *Comp. Graph. Forum (Eurographics Proc.)*, vol. 22, 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. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 193–202.

ROBINSON-MOSHER, A., SHINAR, T., GRÉTARSSON, J. T., SU, J., AND FEDKIW, R. 2008. Two-way coupling of fluids to rigid and deformable solids and shells. *ACM Trans. on Graphics 27*, 3 (Aug.), 46:1–46:9.

ROBINSON-MOSHER, A., SCHROEDER, C., AND FEDKIW, R. 2011. A symmetric positive definite formulation for monolithic fluid structure interaction. *J. Comput. Phys. 230*, 1547–66.

SHAH, M., COHEN, J. M., PATEL, S., LEE, P., AND PIGHIN, F. 2004. Extended galilean invariance for adaptive fluid simulation. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 213–221.

SIN, F., BARGTEIL, A. W., AND HODGINS, J. K. 2009. A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, ACM, New York, NY, USA, SCA '09, 247–255.

STAM, J. 1999. Stable fluids. In *Proc. of SIGGRAPH 99*, 121–128.

SUSSMAN, M., ALGREM, A. S., BELL, J. B., COLELLA, P., HOWELL, L. H., AND WELCOME, M. L. 1999. An adaptive level set approach for incompressible two-phase flow. *J. Comput. Phys 148*, 81–124.

TAN, J., YANG, X., ZHAO, X., AND YANG, Z. 2008. Fluid animation with multi-layer grids. In *ACM SIGGRAPH/Eurographics Symposium of Computer Animation 2008 Posters*.

TESSENDORF, J. 2002. Simulating ocean water. In *SIGGRAPH 2002 Course Notes #9 (Simulating Nature: Realistic and Interactive Techniques)*, ACM Press.

TRAORÉ, P., AHIPO, Y. M., AND LOUSTE, C. 2009. A robust and efficient finite volume scheme for the discretization of diffusive flux on extremely skewed meshes in complex geometries. *J. Comput. Phys. 228*, 14, 5148–5159.