# Closest Point Turbulence for Liquid Surfaces

THEODORE KIM
University of California, Santa Barbara
and
JERRY TESSENDORF
Clemson University
and
NILS THÜREY
Scanline VFX

We propose a method of increasing the apparent spatial resolution of an existing liquid simulation. Previous approaches to this "up-resing" problem have focused on increasing the turbulence of the underlying velocity field. Motivated by measurements in the free surface turbulence literature, we observe that past certain frequencies, it is sufficient to perform a wave simulation directly on the liquid surface, and construct a reduced-dimensional surface-only simulation. We sidestep the considerable problem of generating a surface parameterization by employing an embedding technique known as the *Closest Point Method* (CPM) that operates directly on a 3D extension field. The CPM requires 3D operators, and we show that for surface operators with no natural 3D generalization, it is possible to construct a viable operator using the *inverse Abel transform*. We additionally propose a fast, *frozen core* closest point transform, and an advection method for the extension field that reduces smearing considerably. Finally, we propose two turbulence coupling methods that seed the high resolution wave simulation in visually expected regions.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modelling*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Animation*

Additional Key Words and Phrases: Fluid simulation, liquid simulation, wave turbulence, physically-based modelling

## 1. INTRODUCTION

Despite much recent progress, it remains a challenge to simulate large-scale, high-resolution liquids. The recently popular "up-res" approach separates simulations into large- and small-scale details, and runs separate algorithms for each scale. In addition to being more efficient, imposing a one-way coupling between scales can facilitate design. A user can interact with a fast, low-resolution simulation, and later add additional high-resolution detail in a way that does not invalidate the low-resolution design. This approach has been successfully applied to several natural phenomena, including cloth simulations [Bergou et al. 2007] and single phase smoke simulations [Kim et al. 2008; Narain et al. 2008; Schechter and Bridson 2008; Nielsen et al. 2009; Huang et al. 2011; Yuan et al. 2011]. However, it has been less successful for liquid simulation. Practitioners have reported [Lait 2011] that applying single-phase techniques to liquid simulations introduces undesirable artifacts and does not create plausible new details.

We posit that the reason for this is that the physics being simulated has been incomplete. Previous methods for increasing the resolution of liquid simulations [Narain et al. 2008; Jang et al. 2010; Yuan et al. 2012] have assumed that if the turbulence of the underlying fluid velocity field is increased, high resolution surface dynamics will follow. However, the literature on free surface turbulence, also known as "wave turbulence" or "weak turbulence", maintains that the free surface, especially at high frequencies, possesses additional dynamics that are not mere images of the underlying velocity field. While the low frequency components of the velocity field initiate surface waves, many high frequency details arise from the independent oscillation of the surface membrane [Savelsberg and van de Water 2008; Falcon 2010].

We present a method that captures these additional dynamics by explicitly performing a wave simulation on the liquid surface. In doing so, we reduce the volumetric problem to a surface-only problem. We use the state-of-the-art in visual wave simulation, the *iWave* algorithm [Tessendorf 2004b; 2004a]. As we are simulating a scalar on a surface of rapidly changing topology, we immediately encounter the problem of consistently parameterizing a deforming surface. We sidestep this problem entirely by using a newly developed embedding method known as the *Closest Point Method* (CPM) [Ruuth and Merriman 2008]. The CPM operates on a 3D extension field instead of a 2D surface field, and thus requires no surface parameterization. However, it requires the existence of 3D spatial operators. Natural 3D analogs of 2D surface operators are often available, such as the 5-point 2D and 7-point 3D Laplacians. However, for many operators, such as the fractional Laplacian in the iWave algorithm, no obvious 3D equivalent is available, and it is unclear if the CPM can be used. We show that a viable CPM operator can be constructed by taking the *inverse Abel transform* of the original surface operator.

The CPM has predominantly been used on rigid 3D objects, where the cost of computing a closest point transform [Mauch 2003] can be amortized. We instead deal with a deforming sur-

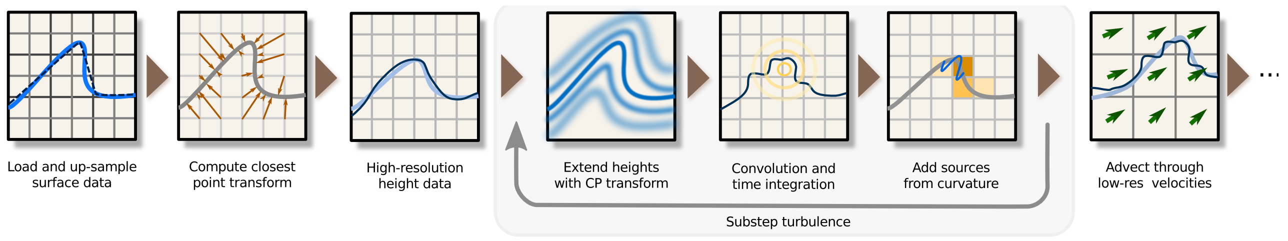

Load and up-sample surface data
Compute closest point transform
High-resolution height data
Extend heights with CP transform
Convolution and time integration
Add sources from curvature
Advect through low-res velocities
...
Substep turbulence

Fig. 1: An overview of the different steps of our simulation algorithm. We read in data from an existing level set solver and add additional surface detail by performing a surface-only wave simulation. The final result can be used as a bump or displacement map during rendering.

face where the transform is computed and advected every frame. In order to prevent this from becoming the bottleneck, we propose an iterative transform based on the *Nacelle* algorithm [Tessendorf 2011] that is faster than fast marching-based methods [Adalsteinsson and Sethian 2003] and more effective at maintaining sharp features. Lastly, we propose two turbulence seeding methods that provide visually consistent methods of coupling the high-resolution surface simulation to the low-resolution volume simulation.

Our specific contributions are as follows:

—A method of constructing operators for the Closest Point Method when no natural 3D operator is available.

—A fast, iterative *Nacelle* method for building the closest point transform of a deforming surface.

—A *frozen core* version of the Nacelle method and an efficient narrow-band advection method that improves surface details.

—Two turbulence seeding strategies that introduce waves in visually expected regions.

## 2. PREVIOUS WORK

**Prior graphics work:** Level set simulation of liquids was pioneered by Foster and Fedkiw [2001]. Since then, many techniques have been proposed for simulating liquids with higher spatial resolution. Recent works include coarse grid projections [Lentine et al. 2010], higher order reinitialization methods [Heo and Ko 2010], complementary Lagrangian meshes [Wojtan et al. 2009], and fast tall cell methods [Chentanez and Müller 2011]. Bargteil et al. [2006] developed a method for texturing such simulations and successfully ran a reaction-diffusion simulation on the surface. However, significant surface parameterization problems arose, which led to subsequent work [Bargteil et al. 2006; Kwatra et al. 2007; Narain et al. 2007] where the parameterization was synthesized each step. A surface texture was then synthesized from exemplars; no simulation took place. Our method sidesteps the parameterization problem entirely and allows a non-trivial iWave simulation to be performed on the surface.

One of the goals of our algorithm is to facilitate the design of liquid animations, so it can also be considered a liquid control algorithm. Many approaches, such as keyframes [McNamara et al. 2004; Shi and Yu 2005] and guiding shapes [Nielsen and Bridson 2011] have been developed to address this problem. Our method can be used to add additional surface detail to the results of these algorithms, so we consider them to be complementary.

A good survey of techniques for simulating ocean waves is available in Darles et al. [2011]. While these techniques give good results for scenes without interaction, we do a full 3D simulation that automatically adds sources and handles obstacle interactions. Other recent work on wave simulation has included the development of fast, Lagrangian “wave particles,” [Yuksel et al. 2007], and the addition of the FFT algorithm described by Tessendorf [2004b] to a shallow water solver [Chentanez and Müller 2010]. Several previous authors have attempted to simulate waves on deforming surfaces. Angst et al. [2008] simulated waves on a fixed character mesh surface that does not undergo any topology changes. They only simulated the traditional wave equation, not the iWave equation. Thürey et al. [2010] also simulated the wave equation on a Lagrangian mesh in order to capture surface tension effects. The focus of their method was on sheet breakup and large scale instabilities, so they did not achieve the fine-scale wave detail that we are able to produce. Kim et al. [2009] simulated a vortex sheet along the liquid surface to capture high resolution interface effects. Again, their approach initiated detailed sheet breakup, which is orthogonal to the surface detail that we capture in this current work, and could be combined with ours to achieve highly detailed liquids.

The closest work to ours is Patel et al. [2009], which performs an orthogonal projection of a 2D iWave simulation onto a 3D river. This approach works best when the 3D liquid is well-approximated by a 2D plane, which is a well-founded assumption for rivers, but clearly not true for general liquids. Figure 4, for example, would be difficult to capture without introducing significant distortions, but is a trivial test case for our method. The algorithm also requires the user to manually specify turbulence injection sites, whereas we propose a method that injects turbulence automatically.

We use the *Closest Point Method* (CPM) [Ruuth and Merriman 2008], a level set-based, parameterization-free surface simulation method, to perform our iWave simulation. The CPM is not the first level set-based method proposed for simulating surface phenomena (see e.g. previous variational formulations [Bertalmío et al. 2001; Greer 2006]), but it sidesteps many of the complexities present in previous methods, so we prefer it here. All of the level set-based methods require 3D generalizations of 2D surface operators, so even if a variational method was employed, the 3D iWave kernel we present in §3.3 would be needed. Other works have used the CPM to simulate fire [Hong et al. 2010], the wave equation, and the Navier-Stokes equations [Auer et al. 2012]. All of these works deal with cases where the surface operators have obvious 3D analogs, such as the gradient and Laplacian. To our knowledge, ours is the first method that successfully uses an operator that is a non-trivial 3D generalization. Hong et al. [2010] apply the CPM to deforming surfaces by propagating scalars between frames using the extension field. We present a fast, iterative method of computing the extension that could be used to accelerate their method, and detail-preserving mechanisms that could further improve their results.

**Prior physics work:** Free surface turbulence is a well studied topic in physics and engineering, and many excellent survey papers are available [Brocchini and Peregrine 2001; Dias and Kharif 1999]. Within this literature, it is well-established (see e.g. [Falcon

2010]) that general hydrodynamic turbulence and free surface turbulence are two distinct phenomena. The former follows the Kolmogorov energy spectrum, and the latter the Kolmogorov-*Zakharov* spectrum [Zakharov et al. 1992]. While liquid free surfaces respond to low frequency eddies in the underlying velocity field, they also exhibit behavior indicative of a "surface skin" layer [Brocchini and Peregrine 2001] that continually tears and undergoes "surface renewal" [Komori et al. 1989].

In particular, Savelsberg and van de Water [2008] experimentally observed that the correlation between the surface gradient and the subsurface velocity field rapidly diverges in turbulent flows. This indicates that the common approach of using the velocity field to capture free surface turbulence [Narain et al. 2008; Jang et al. 2010; Yuan et al. 2012] is insufficient. Savelsberg and van de Water [2008] also established that the surface could be approximated as a set of advected capillary-gravity wave sources, and concluded that "the surface is most likely excited by the largest subsurface turbulence scales only." Motivated by their experimental results, we allow the curvature induced by subsurface turbulence to inject waves into our simulation, but then perform a separate, advected wave simulation along the liquid surface.

## 3. A FREE SURFACE TURBULENCE ALGORITHM

In this section, we describe our algorithm for simulating turbulence on a free surface. We still start with preliminaries on the Closest Point Method (CPM) and iWave algorithms, show how they can be unified, and then present the complete algorithm. A high level overview of our approach can be seen in Figure 1.

### 3.1 The Closest Point Method

The Closest Point Method [Ruuth and Merriman 2008] is an embedding method for simulating partial differential equations (PDEs) on arbitrary surfaces. As with other embedding methods, it works directly on 3D volumes that avoid the problems of traditional surface-based simulation, such as the construction of low-distortion surface parameterizations, and the development of specialized surface-based operators such as the Laplace-Beltrami operator [Wardetzky et al. 2007; Chuang et al. 2009]. While the algorithm operates in 3D, it supports narrow banding, which allows it to scale according to the complexity of the surface, not the volume.

Similar to previous embedding methods, the CPM operates on the 3D *extension field* of the surface, which is constructed by assigning each grid point the scalar value of the nearest surface point. Simulation proceeds by applying the 3D version of the desired PDE to the extension field. For example, in the case of surface diffusion, the familiar 7-point Laplacian stencil would be used instead of a Laplace-Beltrami operator. More concretely, an explicit CPM for diffusing a surface scalar $u_{\mathrm{2D}}$ through $T$ timesteps of size $\Delta t$ on a fixed surface mesh would proceed as shown in Algorithm 1.

**Algorithm 1**: diffuseUsingCPM($u_{\mathrm{2D}}$)

```
1 begin
2     Build the closest point transform, $CP$, of $u_{\mathrm{2D}}$
3     Build the extension field, $u^1_{\mathrm{3D}} = CP(u_{\mathrm{2D}})$
4     for $t$ = 1 to $T$ do
5         $u^*_{\mathrm{3D}} = u^t_{\mathrm{3D}} + \Delta t \cdot \nabla^2 u^t_{\mathrm{3D}}$
6         $u^{t+1}_{\mathrm{3D}} = CP(u^*_{\mathrm{3D}})$
7 end
```

In this algorithm, $\nabla^2$ corresponds to the 7-point Laplacian, and $CP$ interpolates and propagates the scalar values at grid points adjacent to surface out to the entire volume. The algorithm is very similar to a basic 3D explicit integration, with the key addition of the extension step on Line 6. Despite its apparent simplicity, the CPM has been shown to produce the correct curved surface behavior. We will not recap here the validations that have been performed on the method (see e.g. [Macdonald et al. 2011] for a recent example), and will instead introduce relevant details when we later construct our 3D fractional Laplacian.

The CPM is not the first embedding method proposed for implicit surfaces, as variational versions have been available for some time [Bertalmío et al. 2001; Greer 2006]. Unlike the variational versions, the CPM does not require the underlying PDEs to be rewritten to include tangent plane projections that constrain the dynamics to level sets near the interface. Greer [2006] described degeneracies that can occur if the narrow band boundary conditions are not carefully set in a variational method, but no such non-physical boundary conditions are needed by the CPM. Even if a variational version is preferred, all existing embedding methods require 3D generalizations for their 2D operators, so the results presented in §3.3 are still needed. All of the methods require the construction of extension fields, so the closest point transform we describe in §3.5 could also be used to accelerate the variational approaches.

### 3.2 The iWave Algorithm

The iWave algorithm [Tessendorf 2004b] produces more realistic water wave behavior than alternatives such as the traditional wave equation, and is used extensively in production (see e.g. [Carlson 2007; Flores and Horsley 2009; Angelidis et al. 2011]). It is derived from the linearized Bernoulli's equation for irrotational flow,

$$\frac{\partial \phi}{\partial t} = -p - U, \tag{1}$$

where $\phi$ is the velocity potential, $p$ is the pressure, and $U$ is the potential energy. It can be stated in undamped form as the equation:

$$\frac{\partial^2 h}{\partial t^2} = -g\sqrt{-\nabla^2}h. \tag{2}$$

Here, $h$ is the fluid height, $t$ is time, $g$ is the gravity magnitude, and $\sqrt{-\nabla^2}$ is a *fractional Laplacian* operator [Podlubny 1999; Miller and Ross 1993]. Aside from the radical, it is very similar to the traditional wave equation, $\partial^2 h/\partial t^2 = c\nabla^2 h$. The fractional term arises because the gradient of the potential $\phi$ in Bernoulli's equation is constrained to be divergence-free, $\nabla^2\phi = 0$, and a squaring term in the vertical direction $h$ must be accounted for. For this reason, it is also referred to as the *vertical derivative* operator. For further details, see §3.2 in [Tessendorf 2004b].

The fractional nature of the operator significantly complicates its spatial discretization, because fractional derivatives usually have non-local support, and the resulting operator is divergent in frequency space due to the $k^2$ term, where $k$ denotes the spatial frequency. Tessendorf [2004b] addresses the first problem by imposing a hard spatial cutoff, and the second by introducing a Gaussian "soft-cutoff" that suppresses the growth of the $k^2$ term. The final vertical derivative operator $G_{\mathrm{2D}}(r)$ is then stated in polar coordinates [Tessendorf 2008] as

$$G_{\mathrm{2D}}(r) = \int_0^\infty k^2 e^{-k^2} J_0(kr)dk, \tag{3}$$

where $r$ is the radial coordinate, $e^{-k^2}$ is the soft-cutoff, and $J_0$ is the zeroth Bessel function of the first kind. The hard spatial cutoff is realized by only evaluating Eqn. 3 out to a user-specified $r$. Eqn. 3 is discretized into a convolution kernel using Algorithm 2.

**Algorithm 2**: iWave2DKernel($W$, $k_M$)

```
Data: iWave2D is the convolution kernel, W is the spatial
      width of the desired kernel, and k_M is the maximum
      desired wave frequency to be captured.
1 begin
2     iWave2D = 0
3     $h = \lfloor W/2 \rfloor$
4     for y = -h to h do
5         for x = -h to h do
6             $r = \sqrt{x^2 + y^2}$
7             for k = 0 to $k_M$ do
8                 iWave2D$(x, y) += k^2 e^{-k^2} J_0(kr)$
9 end
```

## 3.3 Building a 3D Vertical Derivative

In order to simulate iWave on a surface using the CPM, we need a 3D version of Eqn. 3 and Algorithm 2. However, unlike the Laplacian operator, the vertical derivative has no obvious 3D analog. Indeed, the definition of the operator seems to be inherently surface-based, as the radical arises from taking the square root in the normal direction. The salient spatial function in Eqn. 3 is the $J_0$ Bessel function of the first kind, so a reasonable first attempt is to replace it with the *spherical* Bessel function of the first kind, $j_0(r) = \frac{\sin r}{r} = \mathrm{sinc}(r)$. We found that generating a 3D kernel using a simple, naïve replacement of $J_0$ with $j_0$ results in an unstable simulation and unusable results. More care is clearly needed in the construction of the operator. The broader question is: what makes a good CPM operator? Ruuth and Merriman [2008] reason that if $u_{3\mathrm{D}}$ is an extension of the scalar field $u_{2\mathrm{D}}$, then $u_{3\mathrm{D}}$ does not vary in the normal direction, and so at the surface,

$$\nabla u_{3\mathrm{D}} = \nabla_S u_{2\mathrm{D}}, \tag{4}$$

where $\nabla_S$ denotes the 2D surface gradient. Therefore, $u_{3\mathrm{D}}$ will only vary along the surface, and the $\nabla_S$ operator will only induce motion in the surface tangent directions.

We examine this intuition in a slightly different form. Say we have the scalar field $u_{2\mathrm{D}}(x, y)$, and its extension $u_{3\mathrm{D}}(x, y, z)$. We can state Eqn. 4 in terms of a convolution about the origin with an arbitrary operator $D$:

$$\int_{x,y} D_{2\mathrm{D}}(x,y)\, u_{2\mathrm{D}}(x,y)\, dx\, dy = \int_{x,y,z} D_{3\mathrm{D}}(x,y,z)\, u_{3\mathrm{D}}(x,y,z)\, dx\, dy\, dz.$$

Since $u_{3\mathrm{D}}$ is an extension field, it must be constant in some normal direction. The direction is arbitrary, but for expository purposes, let us choose the $z$ direction:

$$\int_{x,y} D_{2\mathrm{D}}(x,y)\, u_{2\mathrm{D}}(x,y)\, dx\, dy = \int_{x,y} u_{3\mathrm{D}}(x,y,0)\, dx\, dy \int_z D_{3\mathrm{D}}(x,y,z)\, dz.$$

By construction, $\int_{x,y} u_{2\mathrm{D}}(x,y) = \int_{x,y} u_{3\mathrm{D}}(x,y,0)$, so if we assume that $D_{3\mathrm{D}}(x,y,z)$ is spherically symmetric, which is reasonable given that the Laplacian and gradient operators also display this form of symmetry, this further reduces to:

$$D_{2\mathrm{D}}(x,y) = \int_z D_{3\mathrm{D}}(x,y,z)\, dz. \tag{5}$$

Eqn. 5 provides an answer to our original question: a good 3D CPM operator *should project down to the original 2D operator*. Simple inspection shows that this condition is met by the familiar 7-point Laplacian and gradient operators. This can be viewed as ensuring that a CPM simulation on the extension field of a 2D plane produces the same results as a straight 2D simulation.

More formally, the projection of a spherically symmetric function is an *Abel transform*. So, if a natural 3D operator is not available, we can construct one by taking the *inverse* Abel transform. Fortunately, the $J_0$ function is both spherically symmetric and has a known Abel transform pair [Bracewell 1999],

$$\mathcal{A}^{-1}\left(J_0(r)\right) = \frac{1}{\pi}\mathrm{sinc}(r),$$

where $\mathcal{A}^{-1}$ denotes the inverse Abel transform. Using this relation, we can now build the inverse of the 2D iWave kernel:

$$\mathcal{A}^{-1}\left(\int_0^\infty k^2 e^{-k^2} J_0(kr)dk\right) = \frac{1}{\pi}\int_0^\infty k^3 e^{-k^2}\mathrm{sinc}(kr)dk. \tag{6}$$

Note that an extra $k$ appears due to the $J_0(kr)$ term, and we have folded it into the $k^3$ term. Eqn. 6 can now be used to generate a 3D vertical derivative kernel, provided that the removable singularity at $\mathrm{sinc}(0)$ is properly handled. Our 3D vertical derivative operator can now be stated as:

$$G_{3\mathrm{D}}(r) = \frac{1}{\pi}\int_0^\infty k^3 e^{-k^2}\mathrm{sinc}(kr)dk. \tag{7}$$

## 3.4 Reducing Projection Error

The $J_0$ and $j_0$ functions arise from the fractional operator, so they have non-local support that falls off relatively slowly in space. Some projection error is therefore inevitable, as the 3D kernel only extends a finite amount in the normal direction ($z$ in the preceding equations), and the integral in Eqn. 5 will be truncated to some subinterval of $[-\infty, \infty]$.

If a normal direction is known *a priori*, e.g. if the surface is known to be a static plane, then it is possible to correct for this error. For example, the projection error $\epsilon(x, y)$ for a single position $(x, y)$ in the $z$ direction is,

$$\epsilon(x,y) = D_{2\mathrm{D}}(x,y) - \int_{-h}^{h} D_{3\mathrm{D}}(x,y,z)\, dz, \tag{8}$$

where $h$ denotes the spatial cutoff of the kernel. If $\epsilon(x, y)$ is subtracted from an appropriate kernel cell, e.g. $D_{3\mathrm{D}}(x, y, 0)$, the projection error through $(x, y)$ would be reduced to zero. Unfortunately, the normal direction generally changes according to the liquid surface, so precomputing such corrections would introduce undesirable anisotropies into the 3D kernel.

However, it is possible to eliminate all projection error from the cell with the largest weight, the *center* cell. As the kernel is spherically symmetric, $\epsilon(0, 0)$ is the same regardless of projection direction. If $\epsilon(0, 0)$ is subtracted from the center kernel cell, the projection error through the center can be eliminated entirely. With this correction, we found that the relative error of a kernel of width 15

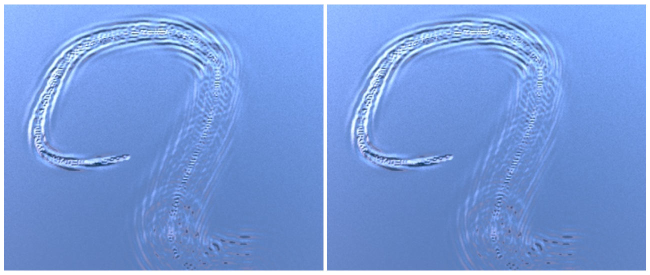


Fig. 2: Validation of our 3D iWave kernel: On the left is the result of a standard planar 2D iWave simulation after a figure-eight shaped mouse input and 280 timesteps. On the right is the result of our 3D simulation after the same number of timesteps on the same input. Our simulation method introduces less than 1% error per timestep and produces visually indistinguishable results.

is 0.26% under the $L_\infty$ norm. The complete algorithm for generating the 3D iWave kernel is now shown in Algorithm 3. In all our examples, we set $k_M = 10$.

**2D Validation:** We have verified that in the simple 2D planar case, our 3D kernel accurately reproduces the results of the original 2D iWave algorithm. We injected wave sources along a pre-defined curve on a planar surface, and used them as inputs to both our 3D solver and the original reference implementation of the 2D iWave solver [Tessendorf 2004b]. As can be seen in Figure 2, as well as the accompanying video, our approach is able to accurately reproduce the results obtained with the 2D iWave algorithm. To quantify the error of our three-dimensional iWave kernel, we computed the difference between the 2D and 3D simulations. We calculated the relative $L_2$ error of the height fields, and found a per-timestep error between 0.15% and 0.25%.

**Algorithm 3**: iWave3DKernel($W$, $k_M$)

```
Data: $W$ is the width of the desired kernel. $k_M$ is the
      maximum desired wave frequency to be captured.
      iWave2D is the kernel computed by Alg. 2.
1  begin
2      iWave3D = 0
3      $h = \lfloor W/2 \rfloor$
4      for $z$ = -$h$ to $h$ do
5          for $y$ = -$h$ to $h$ do
6              for $x$ = -$h$ to $h$ do
7                  $r = \sqrt{x^2 + y^2 + z^2}$
8                  for $k$ = 0 to $k_M$ do
9                      iWave3D$(x, y, z) += \frac{k^3}{\pi} e^{-k^2} \text{sinc}(kr)$
10     sum = 0
11     for $z$ = -$h$ to $h$ do
12         sum $+=$ iWave3D$(0, 0, z)$
13     iWave3D$(0, 0, 0) -=$ iWave2D$(0, 0)-$ sum
14 end
```

## 3.5 A Fast Closest Point Transform

In many previous applications of the CPM, the surface mesh is fixed, so the cost of computing the closest point transform of a surface can be amortized over many timesteps. In our application, the surface is known to be rapidly deforming, so no such amortization is possible. Therefore, it is crucial that a fast method of computing the closest point transform be devised. One approach is to use fast marching-based methods [Adalsteinsson and Sethian 2003], but this approach involves a heap search that is difficult to parallelize, and tends to smear out the scalar field. This smearing is usually considered a feature of fast marching-based methods, as it corresponds to rarefaction solutions of the Eikonal equation. In our application however, this smearing introduces spurious variations along the normal direction. The scan conversion algorithm of Mauch [2003] is another possibility, but it uses the surface triangle mesh, whereas we have a signed distance function that contains richer geometric information.

We have found that the signed distance function of the existing liquid simulation can be used to compute a fast, iterative, highly parallelizable closest point transform. A first order version of the algorithm is similar to the method described by Losasso et al. [2006]. Given a signed distance function $\varphi$, and the cell centers of the computational grid, we can compute the closest point of a cell by starting a particle, $\mathbf{c}_i$, at the cell's center, and iterating along the normal direction, $\varphi(\mathbf{c}_i) \cdot \nabla\varphi(\mathbf{c}_i)$, until $\varphi(\mathbf{c}_i) < \varepsilon$. We used the value $\varepsilon = 10^{-6}$ in our computations.

The *Nacelle* algorithm of Tessendorf [2011] describes a second order method of warping one level set onto another. We can use the same method to compute the closest point transform, which corresponds to a warp of all points to the zero level set. The iteration for each particle $\mathbf{c}_i$ then becomes,

$$\begin{aligned}
\Gamma &= \left(\nabla\varphi(\mathbf{c}_i)^T H(\varphi(\mathbf{c}_i))\nabla\varphi(\mathbf{c}_i)\right) / |\nabla\varphi(\mathbf{c}_i)| \\
\Delta &= -\varphi(\mathbf{c}_i)/|\nabla\varphi(\mathbf{c}_i)| \\
\mathbf{c}_i &= \mathbf{c}_i + \left(-1 + (1 + 2 \cdot \Delta \cdot \Gamma)^{\frac{1}{2}}\right) \frac{\nabla\varphi(\mathbf{c}_i)}{\Gamma},
\end{aligned}$$

where $H(\cdot)$ denotes the Hessian operator. The Hessian can approach zero in flat regions of the distance field and become problematic near the medial axis, so we test if $\Gamma < \varepsilon$ at the beginning of each iteration, and fall back to first order iteration if the condition is true. We found that this variant of the *Nacelle* algorithm is highly parallelizable, and very fast. Results obtained via fast marching and our *Nacelle* variant are shown in Figure 3.

We found that implementing our *Nacelle* variant was quite simple, as it is essentially a particle iteration augmented by a second order correction. None of the heaps or quadratic solves involved in fast marching are needed, and the final code is drastically simpler than the canonical implementation of the Mauch algorithm (`https://bitbucket.org/seanmauch/stlib.`). Unlike fast marching methods, it also supports efficient lazy evaluation: the extension value of any random cell can be queried and computed in $O(1)$ time without computing the values at all of the intervening cells between the queried cell and the interface. We exploit this feature when performing MacCormack advection in §3.6.

## 3.6 Building and Advecting the Extension Field

**A *Frozen Core* Extension Field:** Once the closest point transform has been computed, a method must be selected to extend surface values into the narrow band. Ruuth and Merriman [2008] originally used 4th order Newton divided differences, but subsequent work derived 4th and 6th order Weighted Essentially Non-Oscillatory (WENO) schemes [Macdonald and Ruuth 2009], which we will refer to as WENO4 and WENO6. Too frequent re-extension can be computationally expensive and smear out the scalar field unnec-

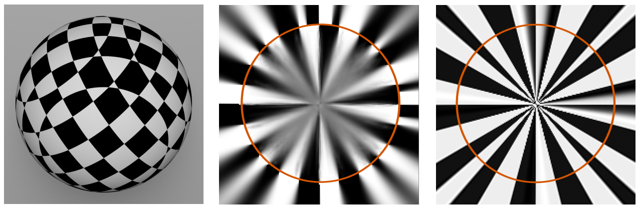


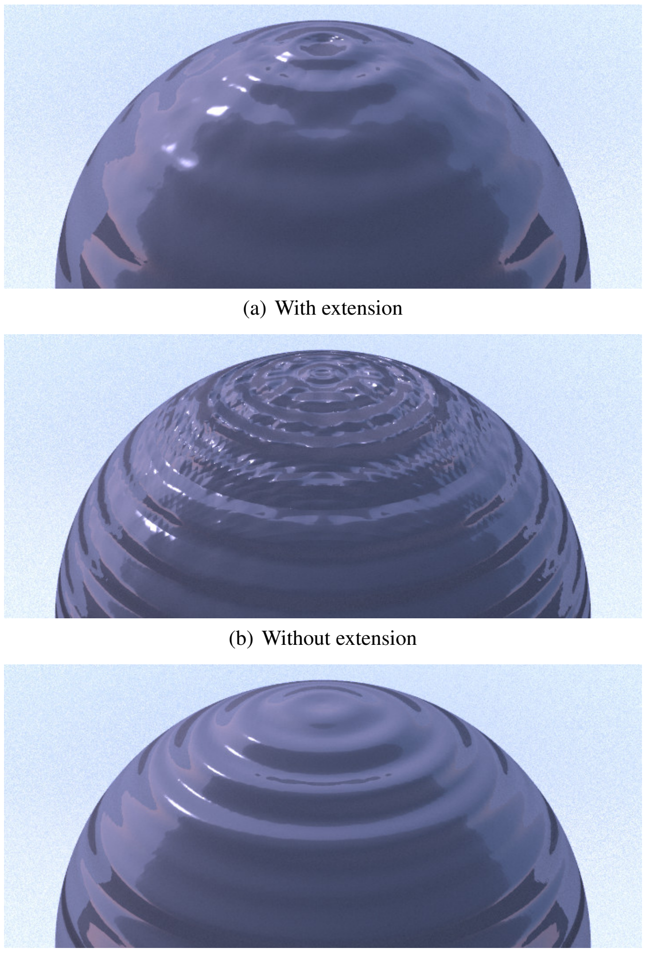
Fig. 3: Left to right: a sphere with a checkerboard surface; the center slice of the sphere's extension field, computed using fast marching; the same slice computed using our *Nacelle* variant. Our variant is faster and does not smear out the scalars. Computing the $200^3$ extension field took 3 minutes and 20 seconds with fast marching, and 21 single-threaded seconds with our algorithm.

essarily. Greer [2006] observed that this is analogous to the well-known problems of periodic velocity field re-extension and signed distance field reinitialization. We also encounter severe smearing when re-extending every timestep (Figure. 4(a)), even when using WENO4 or WENO6. Never re-extending the surface scalars captures crisper features (Figure. 4(b)), but it becomes unclear if valid surface dynamics are being simulated. In both cases, undesirable anisotropies appear that reveal the underlying grid.

We found that a subtle modification fixes both of these problems. When computing the extension field, we 'freeze' the values that are less than one grid cell away from the interface. These values define the on-surface solution, and are accurately computed by the solver, so they should be smeared out as little as possible. We refer to this as a *frozen core*, as it freezes the values at the core of the narrow band. This change significantly improves the crispness of the results, and also suppresses the appearance of grid anisotropies, as shown in Fig. 4(c). This strategy is not entirely novel, as Adalsteinsson and Sethian [2003] make use of a similar technique when initializing their fast marching method. However, we have not seen these substantial improvements noted anywhere else in the literature, so they are worth emphasizing here.

**Narrow Band MacCormack Advection:** We found that first order semi-Lagrangian advection [Stam 1999] smeared out details captured by the CPM, and opted instead to use the MacCormack advection scheme of Selle et al. [2008]. Two modifications significantly improved our results. First, we replaced the linear interpolant for the backtraces with the same WENO4 scheme used for extension field construction. Second, we observed that extending and advecting the surface field unnecessarily interpolated the field *twice*: once during extension, and again during advection. In order to remove this unnecessary smearing, we construct the extension field using *nearest neighbor* interpolation. We use the *Nacelle* algorithm to find the nearest surface point, but instead of interpolating grid values to obtain a final result, we simply grab the value from the nearest grid cell. This essentially computes an anti-rarefaction solution that suppresses all variation in the normal direction. As interpolation still occurs during advection, the field is still smoothed, and we did not observe any stairstepping artifacts. In addition to producing significantly crisper surface details (see Figure 6), the removal of the additional WENO4 call makes this approach computationally cheaper.

The advection scheme must support narrow banding to avoid introducing a volumetric bottleneck into the algorithm. Narrow banding with first order semi-Lagrangian advection is straightforward, as backtraces can be computed for a band around the interface, and the extension values for this band can be computed on-the-fly using the *Nacelle* algorithm. However, the MacCormack method advects



(a) With extension

(b) Without extension

(c) With *frozen core* extension

Fig. 4: We inserted a small circular wave at the top of a sphere and simulated 500 timesteps using different re-extension strategies. *Top to bottom:* with WENO4 re-extension every timestep, with no re-extension, and WENO4 re-extension using the *frozen core* from §3.6 every step. The frozen core result does not smear out the waves and suppresses grid anisotropies that ruin the symmetry of the other cases.

the field forward and backwards to compute an error term. The narrow bands for these stages differ, and computing the values for the backwards stage is significantly more complex: they correspond to an *advected* extension field, not just an extension field. Therefore, we cannot obtain valid values for the backwards band by simply applying the *Nacelle* algorithm. A obvious solution to this problem would be to fatten the narrow band for the forward stage according to the grid velocities, but in practice this results in a significant amount of unnecessary computation.

We instead performed a preliminary pass to determine the exact set of cells needed to perform narrow band MacCormack advection. We traced the velocities forward to find all the cells needed for the forward band, and traced *these* cells backwards to determine the cells needed in the backwards band. We then constructed the extension field for *all* of these cells and performed the advection. We give an overview of the subtleties of this process in Figure 5. The preliminary pass did not contain any calls to the WENO4

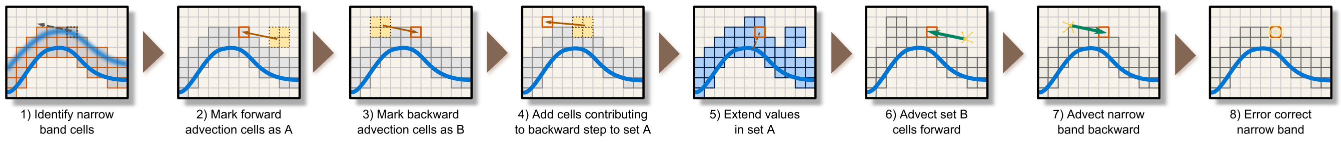

1) Identify narrow band cells
2) Mark forward advection cells as A
3) Mark backward advection cells as B
4) Add cells contributing to backward step to set A
5) Extend values in set A
6) Advect set B cells forward
7) Advect narrow band backward
8) Error correct narrow band

Fig. 5: An overview of our narrow band advection. For clarity, we only show one side of the narrow band, and use linear interpolation stencils. The active cells for each step are highlighted, and the source cell with its velocity is shown in step 1. Note that both step 2 and 4 are adding cells to set $A$, which contains all cells that are initialized by extension in step 5. The cells from step 4 are needed to compute the backward advection in step 7.

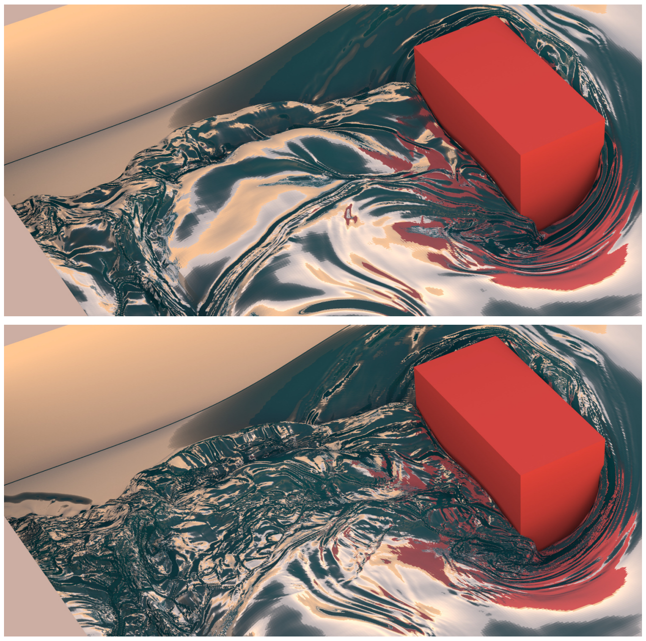


Fig. 6: **Top:** Without *frozen core* extension and improved advection from §3.6. **Bottom:** Same frame, with modifications from §3.6. Note that the turbulent wake behind the block is lost entirely without our improvements.

interpolant, and thus consisted mostly of fast integer operations that consumed $5\%$ to $6\%$ of the running time. By comparison, when we fattened the narrow band according to cell velocities, it doubled the extension field building time. This build time becomes the main bottleneck at high resolutions (See Table I), so this approach added at least an additional $25\%$ to the running time. Our two pass method is clearly more efficient.

## 3.7 Turbulence Seeding

Wave propagation is only perceived as realistic if waves are seeded in visually expected regions. Following the intuition of Kim et al. [2008], we identify under-resolved regions of the fluid surface where details are being lost. For liquids, this corresponds to regions of high surface curvature, so we inject turbulence into locations where the absolute principal surface curvatures are close to the Nyquist limit of the current grid. This is in line with other curvature-based strategies in liquid simulations, such as those employed recently by Thüery et al. [2010] and Yu et al. [2012] to seed a (classical) surface wave simulation, as well as the particle seeding strategy of Foster and Fedkiw [2001]. Intuitively, this corresponds to regions where the 'surface skin' layer of the liquid tears and initiates surface oscillations.

We compute a source field for injecting surface waves by filtering the maximum curvature values with a Catmull-Rom spline centered at half the grid resolution, and a falloff value of one-fifth the grid resolution. Once the seeding regions have been located, we set the source term in these regions to the local Gaussian curvature in order to reflect the variations occurring along the surface. We found that the curvature computation method from Museth et al. [2002] provided smooth, robust results for both the principal and Gaussian curvatures. Note that all of these quantities can be computed efficiently on the low-resolution grid.

We found that in some scenarios, adding this source field to the height field was sufficient (Figs. 8 and 10). However, if the appearance of a higher apparent surface resolution is desired, convolving the source field once with the vertical derivative operator creates the impression that scattering has occurred across a wider range of scales, and produces higher frequency waves. For this additional convolution, we use a vertical derivative operator (Eqn. 7) integrated over the $[2, k_M)$ domain, and use the extension field of the result. This seeding method was used in Figure 7. We exclude the $[0, 2)$ range because these frequencies are close to the Nyquist frequencies already present on the grid. We add the source field to the height fields of both the current and previous timesteps in order to convey the impression that the waves have persisted for some time, but are currently scattering into higher frequencies. Otherwise, the sources induce instantaneous velocities that produce visual spikes in the height field. We have found that these two turbulence seeding strategies, Gaussian curvature and convolved Gaussian curvature, work well in practice. These are by no means the only strategies possible, but we leave further exploration to future work.

The seeding strategy should be made aware of internal obstacles in order to avoid injecting spurious turbulence. In Figure 7, a large amount of surface curvature exists where the liquid wraps around the central column. Much of this curvature is along the liquid-*obstacle* interface, not the liquid-air interface, so injecting turbulence in these regions is incorrect, and can result in overly lively waves around the column. This problem is addressed by zeroing out the source term in regions surrounding internal obstacles.

## 3.8 The Complete Algorithm

We have now described all of the components of our algorithm. The complete algorithm is shown in Algorithm 4. The extensions performed on Line 2 are the nearest neighbor extensions described in §3.6, while the extensions on lines 6 and 8 use the WENO4 interpolant. As the iWave algorithm uses explicit integration, the surface wave simulation is run for $T$ user-specified substeps for every step of the coarse simulation. We found that setting $T = 5$ worked well in all of our examples. Line 11 encodes the Leapfrog scheme from Tessendorf [2004b].

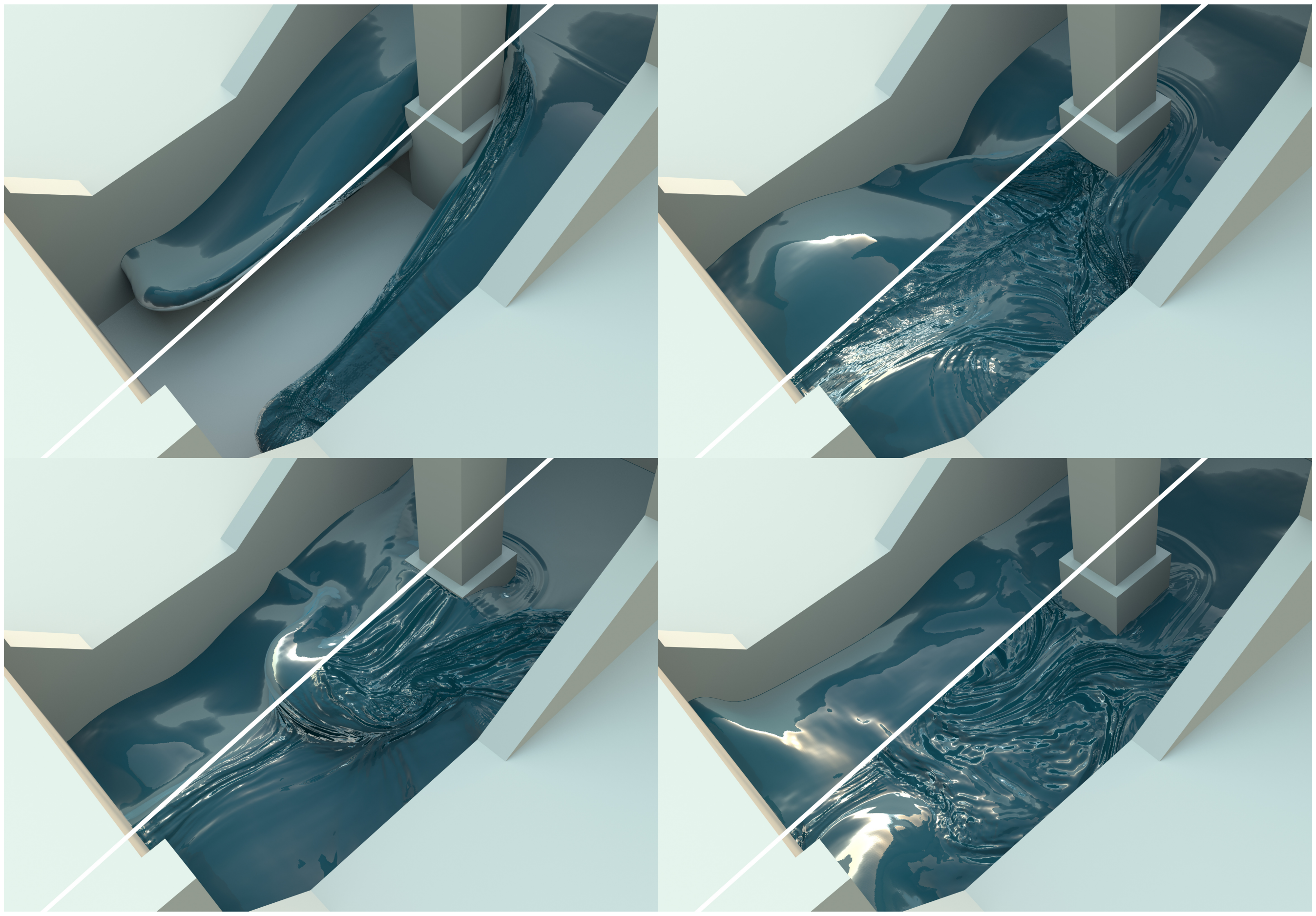


Fig. 7: A $100^2 \times 50$ PhysBAM simulation, up-resed to $800^2 \times 400$. The left half of each image shows the original simulation, and the right shows our up-resed version. Our simulation took roughly $10\times$ the time of the original, whereas a brute force simulation would take roughly $512\times$, i.e. $8^3$, the time. The source and height values around the column were clamped to zero as respectively described in §3.7 and §4.

## 4. DISCUSSION AND RESULTS

The iWave algorithm uses the 'deep water' approximation $h \gg \lambda$, where $h$ is the water depth and $\lambda$ is the wave length. As we are simulating high frequencies, our $\lambda$s are quite small, so this approximation is valid even if the liquid is globally shallow. We note that this 'relatively deep' approximation is fairly common in the fluid mechanics literature, and some practitioners [Johnson 1997] prefer to use the terms 'short' and 'long' waves in lieu of 'deep' and 'shallow' in order to avoid any confusion. If alternate dispersion relations are desired, Eqn. 7 can be scaled using the exact same terms described in Tessendorf [2004b].

**Implementation:** We ran all of our simulations on a 12-core 2.66 Ghz Mac Pro with 96 GB of memory. All of our simulations fit into memory, so we did not need to use a hierarchical or blocked data structure. However, we expect that such structures would yield additional speedups due to improved memory locality. We used the WENO4 interpolant [Macdonald and Ruuth 2008], which has a stencil width of four. The more expensive WENO6 was also tested, but it did not improve the results sufficiently to justify the additional computation. We used a 3D iWave kernel (Eqn. 7) with a stencil width of 15. Reducing its spatial extent would reduce the running time, but at the cost of reduced wave propagation speeds. We used OpenMP to parallelize the convolution, *Nacelle* computation, and extension field computation stages of our algorithm. While we parallelized the most computationally intensive functions, the entire algorithm can be run in parallel, so it is an excellent candidate for GPU acceleration. We found that the obstacle interaction method of the original iWave algorithm, which set the height values on the interior of obstacles to zero, worked well in our 3D generalization. No additional considerations were needed. All of our results were rendered using a modified version of PBRT [Pharr and Humphreys 2010] that read in the height fields from our simulations as solid textures and then used them as bump maps. The solid texture lookup function in PBRT was modified to use the WENO4 interpolant.

**Houdini test:** We compared both the scalability and quality of our algorithm to the Houdini simulation from Lait [2011]. We used the Houdini 12 solver, which utilizes a parallel Preconditioned Conjugate Gradient solver, and collected timing information for the scene at the resolutions of $100^3$, $200^3$ and $400^3$. At the highest resolution, the simulation took nearly a week, which is totally

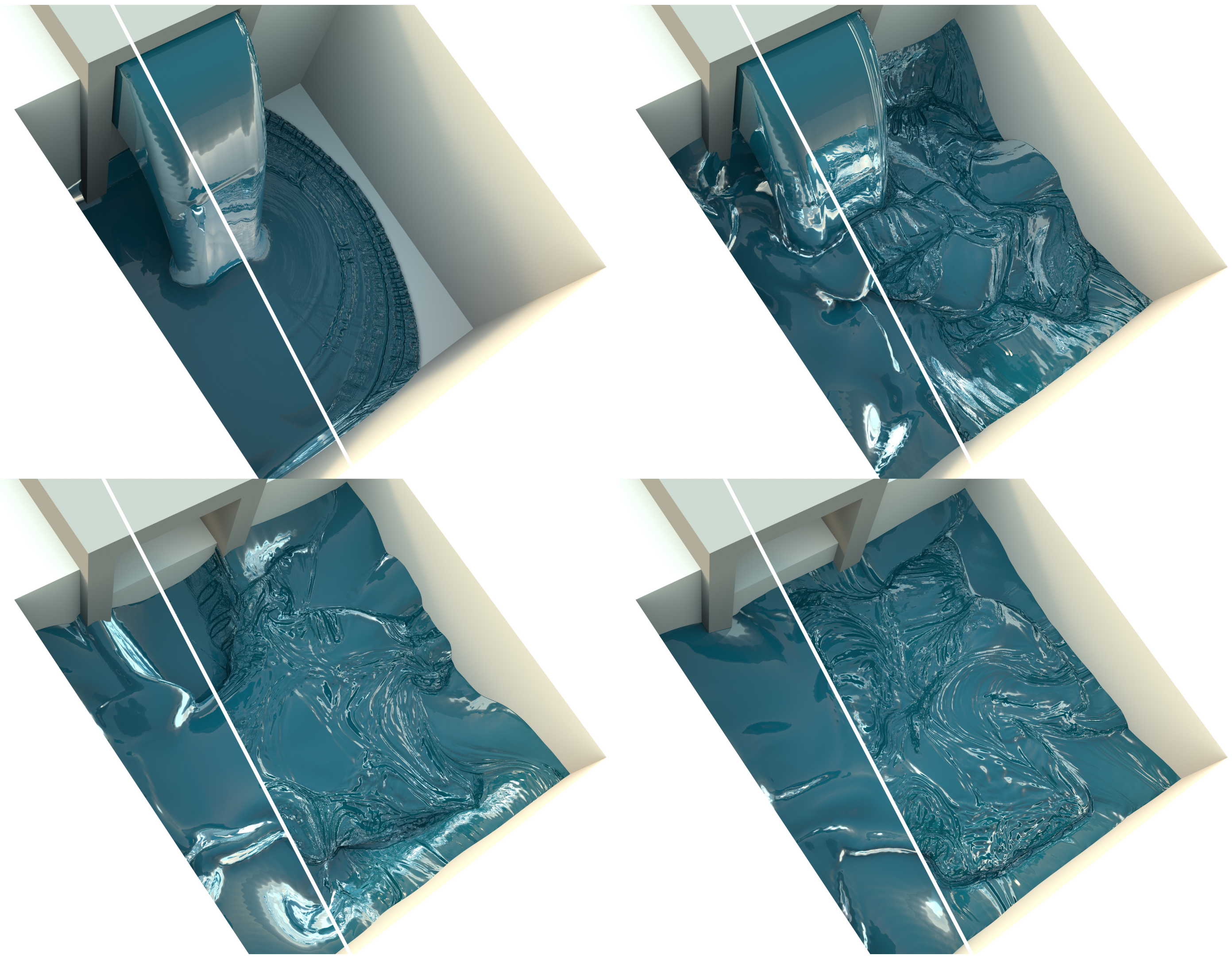


Fig. 8: A $100^3$ PhysBAM simulation, up-resed to $800^3$. The left half of each image shows the original simulation, and the right shows our up-resed version. The $8\times$ up-resing only took approximately twice the time of the original simulation.

impractical for production. The overall motion of the liquid clearly changed between resolutions.

By comparison, our solver was able to up-res the $100^3$ simulation to $200^3$ in *less than half of the time of the base simulation*. When comparing our results to the direct $200^3$ solution, we observed that our algorithm captured higher frequency motions (Figure 10). The main bottleneck in the simulation at this resolution was writing the large volume files to disk, which took up 46% of the running time. Using a sparse volume data structure, or an integrated simulation-renderer solution, would yield additional speedups. We also up-resed the $100^3$ simulation to $400^3$, $800^3$, and $1000^3$. The $800^3$ and $1000^3$ simulations in particular captured extremely detailed surface motion, and would take months for the Houdini 12 solver to compute. Disk I/O remained one of the bottlenecks, respectively taking 25%, 22%, and 21% of the running times. Convolution and extension field construction times also become more significant, which suggests that more aggressive parallelization could yield further speedups.

The running time exhibits inferior scaling when increasing from $400^3$ to $800^3$, though the scaling is still significantly better than the greater than $8\times$ scalings observed for the direct volumetric solvers. The disk I/O does not appear to be solely responsible for this, as the convolution and extension field stages also exhibit roughly this scaling. Most likely the memory traffic from the large volumes is saturating the bus, which further suggests that investigating high bandwidth architectures such as a GPU might be fruitful.

**PhysBAM tests:** Our algorithm is agnostic to the source of the level set data, so Figures 7 and 8 show the results of running our algorithm on two simulations produced using the PhysBAM code release [Dubey et al. 2011]. In keeping with our goal of up-resing, we again ran the simulation at relatively coarse $100^3$ and $100^2 \times 50$ resolutions. The simulations were run single-threaded, as the multi-threaded version of the PhysBAM release is listed as “experimen-

| Example | Base Res. | Upped Res. | Frame Time | Total Time | Scaling | Convolution | Extension Field | Disk I/O | Advection |
|---|---|---|---|---|---|---|---|---|---|
| Houdini | $100^3$ | N/A | 00:00:24 | 01:39:00 | - | - | - | - | - |
| Figure 10 | $200^3$ | N/A | 00:03:44 | 14:59:00 | 9.08 | - | - | - | - |
| | $400^3$ | N/A | 00:38:00 | 152:00:00* | 10.1 | - | - | - | - |
| | $100^3$ | $200^3$ | 00:00:12 | 00:48:14 | - | 00:04:45 (10%) | 00:11:11 (23%) | 00:22:16 (46%) | 00:04:55 (10%) |
| | $100^3$ | $400^3$ | 00:00:58 | 03:55:00 | 4.87 | 00:37:34 (16%) | 01:13:22 (31%) | 00:58:40 (25%) | 00:32:15 (14%) |
| | $100^3$ | $800^3$ | 00:05:29 | 21:56:15 | 5.60 | 03:28:43 (16%) | 06:27:35 (29%) | 04:43:19 (22%) | 02:49:09 (13%) |
| | $100^3$ | $1000^3$ | 00:11:12 | 44:51:56 | 2.07 | 05:46:41 (13%) | 11:33:35 (26%) | 09:21:30 (21%) | 05:31:15 (12%) |
| Pouring | $100^3$ | N/A | 00:01:52 | 06:13:20 | - | - | - | - | - |
| Figure 8 | $100^3$ | $200^3$ | 00:00:07 | 00:24:38 | - | 00:02:20 (9%) | 00:03:27 (14%) | 00:14:12 (58%) | 00:01:54 (8%) |
| | $100^3$ | $400^3$ | 00:00:34 | 01:53:55 | 4.62 | 00:13:58 (12%) | 00:28:13 (25%) | 00:38:51 (34%) | 00:13:29 (12%) |
| | $100^3$ | $800^3$ | 00:03:15 | 13:00:52 | 6.85 | 01:23:50 (11%) | 03:33:08 (27%) | 03:40:38 (28%) | 01:45:20 (13%) |
| Dam | $100^2{\times}50$ | N/A | 00:00:18 | 01:33:00 | - | - | - | - | - |
| Break | $100^2{\times}50$ | $200^2 \times 100$ | 00:00:06 | 00:30:44 | - | 00:01:57 (6%) | 00:02:45 (9%) | 00:21:05 (69%) | 00:01:41 (5%) |
| Figure 7 | $100^2{\times}50$ | $400^2 \times 200$ | 00:00:21 | 01:45:33 | 3.43 | 00:12:30 (12%) | 00:23:47 (23%) | 00:38:13 (36%) | 00:11:56 (11%) |
| | $100^2{\times}50$ | $800^2 \times 400$ | 00:02:48 | 11:13:09 | 6.38 | 01:20:12 (12%) | 02:52:27 (26%) | 02:50:46 (25%) | 01:35:31 (14%) |

Table I. : All timings are in **hours:minutes:seconds**. The *Scaling* column denotes the scaling relative to the timing in the preceding row. Rows marked *N/A* in the *Upped Res.* column are timings for direct Houdini or PhysBAM simulations. The columns *Convolution - Advection* refer to fractions of the *total* running time. The entry * is the projected time based on 100 frames. The Houdini, Pouring, and Dam Break examples were respectively run for 240, 200, and 300 timesteps. The Houdini and PhysBAM timings included Disk I/O, so they have been included in timings of our algorithm to facilitate comparisons.

**Algorithm 4**: surfaceWaves($\varphi^t$, $\mathbf{v}^t$, $CP^{t-1}$, $h^t$, $h^{t-1}$)

```
Data: $\varphi^t$ and $\mathbf{v}^t$ are the current level set and velocity field of
      the coarse simulation; $CP^t$ is the closest point
      transform of timestep $t$, $h^t$ is the surface height at
      timestep $t$; $\alpha$ is a damping coefficient, $T$ is the number
      of substeps, $\Delta t$ is the surface simulation timestep size.
1  begin
2      $h^t = CP^{t-1}(h^t)$, $h^{t-1} = CP^{t-1}(h^{t-1})$
3      Advect $h^t$ and $h^{t-1}$ using $\mathbf{v}^t$.
4      Build closest point transform of $\varphi^t$, $CP^t$.
5      source = filtered curvature of $\varphi^t$, convolved by Eqn. 7.
6      source = $CP^t$(source)
7      for i = 1 to T do
8          $h^t = CP^t(h^t)$, $h^{t-1} = CP^t(h^{t-1})$
9          $d = h^t$ convolved by Eqn. 7
10         temp = $h^t$
11         $h^t = \frac{(2-\alpha\Delta t)h^t - h^{t-1}}{1+\alpha\Delta t}$ + source − $d$
12         $h^{t-1}$ = temp + source
13         Clear the source field if $i = 1$
14 end
```

tal". We expect that a multi-threaded implementation would produce timings competitive with the Houdini solver.

We observed timing breakdowns and scalings that were similar to the Houdini example. Disk I/O dominates initially, but decreases to roughly a quarter of the running time at higher resolutions. The same decrease in scaling at $800^3$ is also observed. In the "Dam Break" example (Figure 7), we were able to up-res the simulation by a factor of four along each spatial axis using approximately the same amount of time as the original simulation, and in the "Pouring" example (Fig. 8), in *less* time than the original simulation.

**Other wave kernels:** Once the components of our turbulence algorithm are in place, it becomes straightforward to experiment with other models of wave motion. We ran the "Dam Break" example using the traditional wave equation instead of the iWave kernel in Figure 9. The traditional wave equation still gives useful results, but the smaller kernel introduced a smaller timestep size, and the final results tended to suppress higher-frequency waves. However, if the user prefers this 'look', we found that it could be achieved with minimal code modification. Other models, such as the Korteweg-de Vries and non-linear Schrödinger equations [Johnson 1997] could also be used to achieve alternate looks.

**The importance of damping:** We found that the damping parameter, $\alpha$ in Algorithm 4, had a significant impact on the quality of the final results. For less damped simulations, $0 < \alpha < 0.2$, the waves persisted longer than expected and produced a distracting "memory" effect. Higher dampings, $0.2 < \alpha < 0.4$, produced behavior more in line with perceptual expectations. This is in agreement with the default setting of $\alpha = 0.3$ in the original 2D iWave implementation [Tessendorf 2004b]. A comparison of various $\alpha$ settings can be seen in Figure 11 and the supplemental video.

## 5. CONCLUSIONS AND FUTURE WORK

We have described an efficient, closest point method of increasing the apparent spatial resolution of an existing liquid simulation. We have addressed two main obstacles to performing this in a Eulerian setting: the construction of a 3D iWave operator, and the efficient extension of surface scalars. We have additionally described methods for maintaining simulation details, and proposed two turbulence seeding methods. The algorithm can produce surface features in running times competitive with, and sometimes superior to, the original base simulation. We have used our algorithm to simulate liquids containing detailed, high frequency motion that, to our knowledge, have not been captured by any previous method.

Since we are dealing specifically with the problem of "up-resing" a liquid, our algorithm only performs a one-way coupling. It remains to be seen if the higher frequency detail can be coupled back to the coarse simulation, as was done in Thürey et al. [2010]. Our algorithm only requires signed distance and velocity fields as inputs, so it could be applied to animated meshes, as was done in [Angst et al. 2008], by computing the signed distance field of the mesh and extrapolating velocities from the vertices. For particle-based liquid simulations, a distance function such as the one pro-

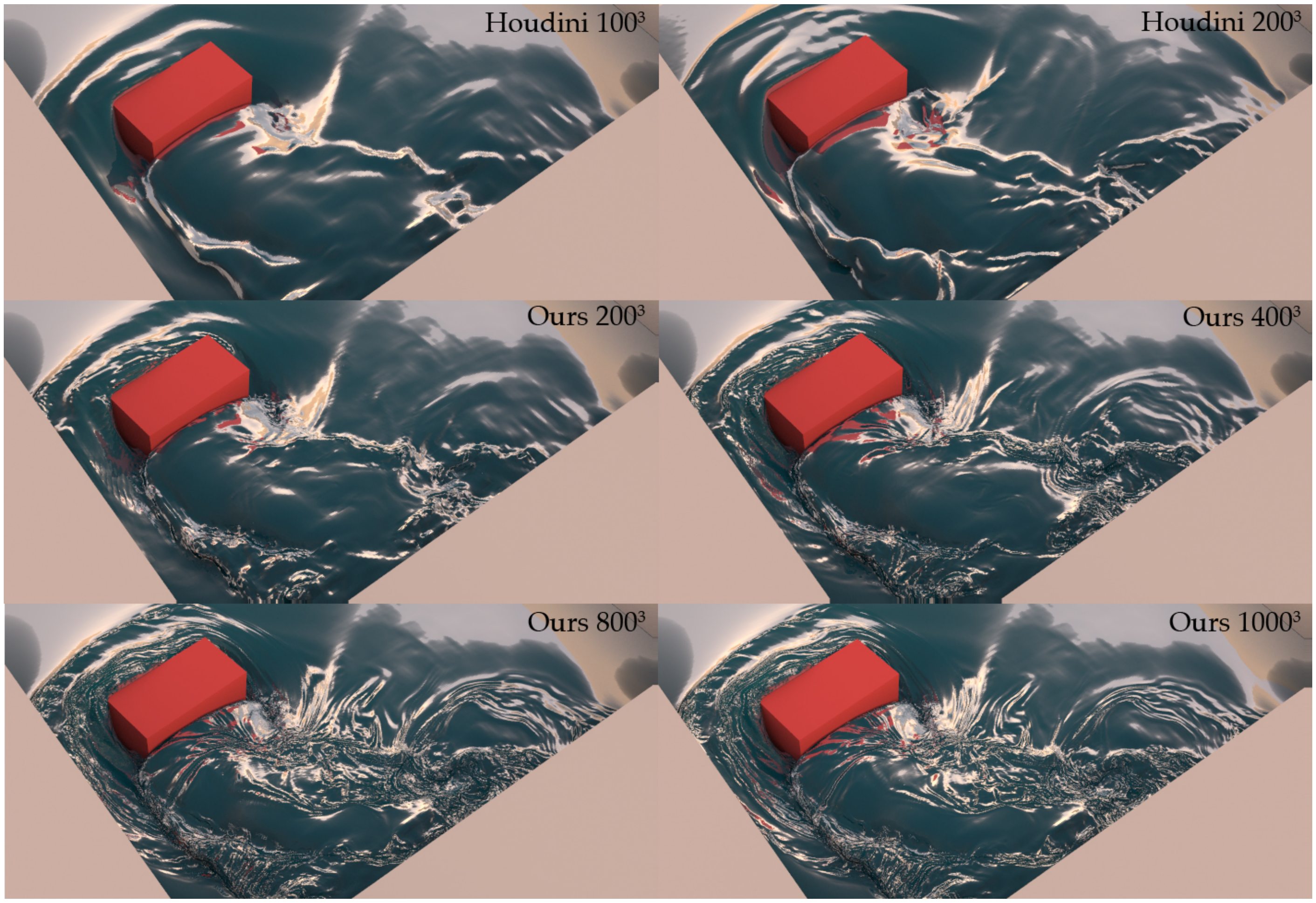


Fig. 10: **In reading order:** Original $100^3$ Houdini simulation, 2, 4, 8, and $10\times$ upres, and direct $200^3$ Houdini simulation. Note how even at $2\times$ upres, higher frequency waves than those in the direct $200^3$ solution are captured.

posed by Zhu and Bridson [2005] could provide a basis for our approach. While an implicit version of the CPM [Macdonald and Ruuth 2009] would allow our algorithm to take larger timesteps, modifications would be needed to the Leapfrog scheme used by the iWave algorithm. Such a scheme could significantly improve the efficiency of simulating high phase-velocity capillary waves. In §3.3, we imposed a physically consistent spherical symmetry constraint on the 3D kernel. Relaxing this assumption presents opportunities for motif-based stylizations such as those in Ma et al. [2009].

We have shown that CPM surface physics can be viewed in terms of an Abel transform, and that surface scalar extension and convolution become bottlenecks at high resolutions. There is a rich body of literature surrounding the Fourier-Abel-Hankel transform cycle [Bracewell 1999], so there may be a signal processing approach that can accelerate these stages. Finally, we have used the inverse Abel transform to generalize one non-trivial 2D operator, the fractional Laplacian, to 3D. We are confident that this methodology will be useful in making other surface-only operators compatible with the CPM.

## ACKNOWLEDGMENTS

The authors would like to acknowledge funding and support from NSERC (Discovery Grant: Many-Core Physically-Based Simulation) and Side Effects Software. The authors would like to thank the anonymous reviewers, whose careful feedback significantly improved this manuscript, John Delaney for assisting with rendering, Jeff Lait for sharing the Houdini scene for Fig. 10, and Colin Macdonald for answering questions about the Closest Point Method during the preliminary stages of this work.

## REFERENCES

ADALSTEINSSON, D. AND SETHIAN, J. 2003. Transport and diffusion of material quantities on propagating interfaces via level set methods. *J. Comput. Phys. 185,* 1, 271 – 288.

ANGELIDIS, A., ANON, J., BRUINS, G., REISCH, J., AND VARAGNOLO, E. 2011. Ocean mission on Cars 2. In *ACM SIGGRAPH 2011 Talks*. 17:1–17:1.

ANGST, R., THÜREY, N., BOTSCH, M., AND GROSS, M. 2008. Robust and efficient wave simulations on deforming meshes. *Comput. Graph. Forum 27 (7)*, 6, 1895 – 1900.

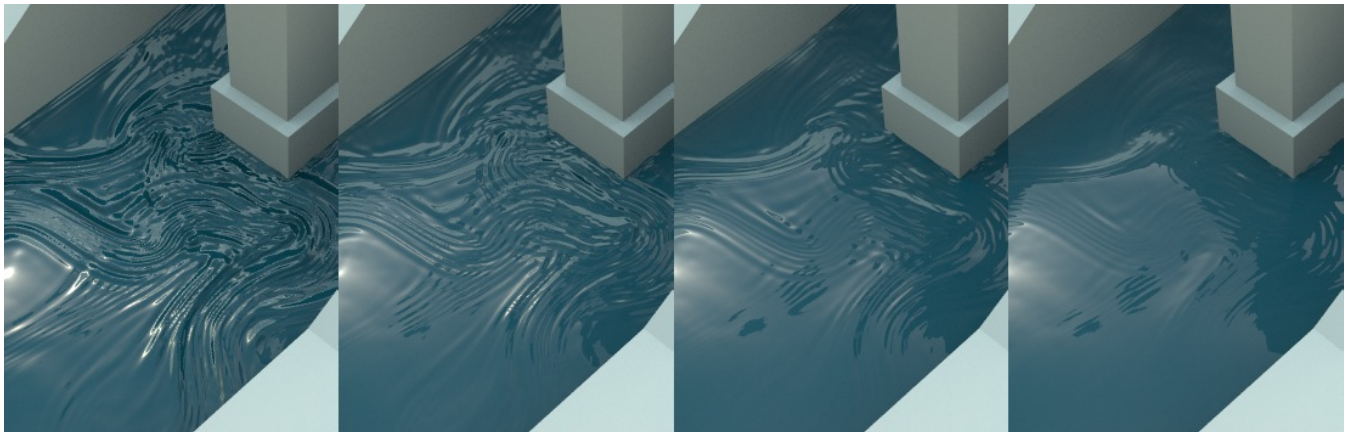


Fig. 11: Different settings for the damping $\alpha$. **Left to right:** $\alpha = 0.1, 0.2, 0.3$ and $0.4$. Waves persist for too long for low dampings, but can die off too quickly before contributing any detail with high dampings.

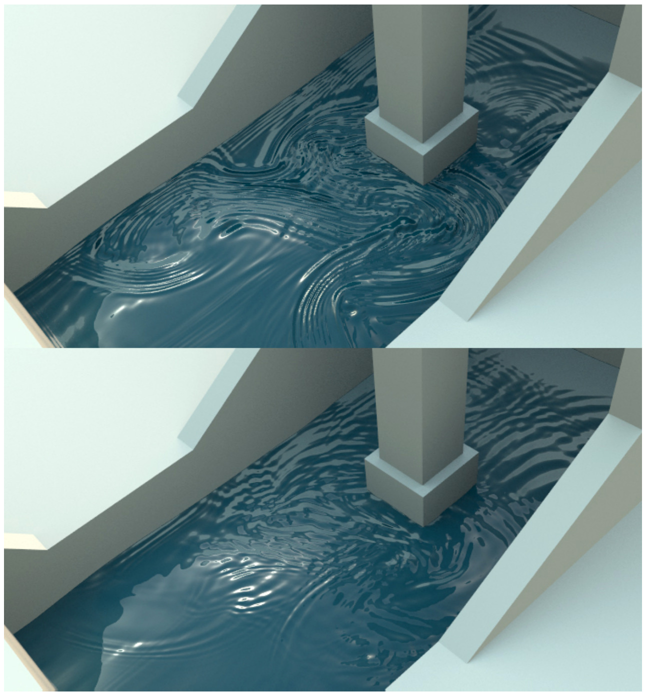


Fig. 9: **Top:** iWave kernel **Bottom:** Traditional wave equation. When the traditional wave kernel is used, usesul results are obtained, but the waves are not as sharp, and tend towards a preferred frequency.

AUER, S., MACDONALD, C., TREIB, M., SCHNEIDER, J., AND WESTERMANN, R. 2012. Real-time fluid effects on surfaces using the Closest Point Method. *Computer Graphics Forum (in press)*.

BARGTEIL, A. W., GOKTEKIN, T. G., O'BRIEN, J. F., AND STRAIN, J. A. 2006. A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph. 25*, 19–38.

BARGTEIL, A. W., SIN, F., MICHAELS, J. E., GOKTEKIN, T. G., AND O'BRIEN, J. F. 2006. A texture synthesis method for liquid animations. In *ACM SIGGRAPH/Eurographics Symposium on Computer animation*. 345–351.

BERGOU, M., MATHUR, S., WARDETZKY, M., AND GRINSPUN, E. 2007. Tracks: toward directable thin shells. *ACM Trans. Graph. 26,* 3 (July).

BERTALMÍO, M., CHENG, L.-T., OSHER, S., AND SAPIRO, G. 2001. Variational problems and partial differential equations on implicit surfaces. *J. Comput. Phys. 174,* 2, 759 – 780.

BRACEWELL, R. 1999. *The Fourier Transform and Its Applications*. McGraw–Hill.

BROCCHINI, M. AND PEREGRINE, D. H. 2001. The dynamics of strong turbulence at free surfaces. Part 1. description. *Journal of Fluid Mechanics 449*, 225–254.

CARLSON, D. 2007. Wave displacement effects for Surf's Up. In *ACM SIGGRAPH 2007 Sketches*. ACM, New York, NY, USA.

CHENTANEZ, N. AND MÜLLER, M. 2010. Real-time simulation of large bodies of water with small scale details. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 197–206.

CHENTANEZ, N. AND MÜLLER, M. 2011. Real-time Eulerian water simulation using a restricted tall cell grid. *ACM Trans. Graph. 30*, 82:1–82:10.

CHUANG, M., LUO, L., BROWN, B. J., RUSINKIEWICZ, S., AND KAZHDAN, M. 2009. Estimating the Laplace-Beltrami operator by restricting 3d functions. In *Eurographics Symposium on Geometry Processing*. 1475–1484.

DARLES, E., CRESPIN, B., GHAZANFARPOUR, D., AND GONZATO, J. 2011. A Survey of Ocean Simulation and Rendering Techniques in Computer Graphics. *Comput. Graph. Forum 30*, 43–60.

DIAS, F. AND KHARIF, C. 1999. Nonlinear gravity and capillary-gravity waves. *Annual Review of Fluid Mechanics 31*, 301–346.

DUBEY, P., HANRAHAN, P., FEDKIW, R., LENTINE, M., AND SCHROEDER, C. 2011. PhysBAM: physically based simulation. In *ACM SIGGRAPH 2011 Courses*. 10:1–10:22.

FALCON, E. 2010. Laboratory experiments on wave turbulence. *Discrete. Cont. Dyn.-B 13*, 819–840.

FLORES, L. AND HORSLEY, D. 2009. Underground cave sequence for Land of the Lost. In *ACM SIGGRAPH 2009 Talks*. ACM, New York, NY, USA, 6:1–6:1.

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

GREER, J. B. 2006. An improvement of a recent Eulerian method for solving pdss on general geometries. *J. Sci. Comput. 29,* 3 (June), 321–352.

HEO, N. AND KO, H.-S. 2010. Detail-preserving fully-Eulerian interface tracking framework. *ACM Trans. Graph. 29*, 176:1–176:8.

HONG, Y., ZHU, D., QIU, X., AND WANG, Z. 2010. Geometry-based control of fire simulation. *The Visual Computer 26*.

HUANG, R., MELEK, Z., AND KEYSER, J. 2011. Preview-based sampling for controlling gaseous simulations. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 177–186.

JANG, T., KIM, H., BAE, J., SEO, J., AND NOH, J. 2010. Multilevel vorticity confinement for water turbulence simulation. *The Visual Computer 26*, 873–881.

JOHNSON, R. S. 1997. *A Modern Introduction to the Mathematical Theory of Water Waves*. Cambridge University Press.

KIM, D., SONG, O.-Y., AND KO, H.-S. 2009. Stretching and wiggling liquids. *ACM Trans. Graph. 28,* 5 (Dec.), 120:1–120:7.

KIM, T., THÜREY, N., JAMES, D., AND GROSS, M. 2008. Wavelet turbulence for fluid simulation. *ACM Trans. Graph. 27*, 50:1–50:6.

KOMORI, S., MURAKAMI, Y., AND UEDA, H. 1989. The relationship between surface-renewal and bursting motions in an open channel flow. *Journal of Fluid Mechanics 203*, 103–123.

KWATRA, V., ADALSTEINSSON, D., KIM, T., KWATRA, N., CARLSON, M., AND LIN, M. 2007. Texturing fluids. *IEEE Transactions on Visualization and Computer Graphics 13,* 5 (sept.-oct.), 939 –952.

LAIT, J. 2011. Correcting low frequency impulses in distributed simulations. In *ACM SIGGRAPH Talks*. 53:1–53:2.

LENTINE, M., ZHENG, W., AND FEDKIW, R. 2010. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph. 29*, 114:1–114:9.

LOSASSO, F., FEDKIW, R., AND OSHER, S. 2006. Spatially adaptive techniques for level set methods and incompressible flow. *Computers & Fluids 35,* 10, 995 – 1010.

MA, C., WEI, L.-Y., GUO, B., AND ZHOU, K. 2009. Motion field texture synthesis. *ACM Trans. Graph. 28,* 5 (Dec.), 110:1–110:8.

MACDONALD, C. B., BRANDMAN, J., AND RUUTH, S. J. 2011. Solving eigenvalue problems on curved surfaces using the Closest Point Method. *J. Comput. Phys. 230,* 22.

MACDONALD, C. B. AND RUUTH, S. J. 2008. Level set equations on surfaces via the Closest Point Method. *J. Sci. Comput. 35,* 2–3 (June), 219–240.

MACDONALD, C. B. AND RUUTH, S. J. 2009. The implicit Closest Point Method for the numerical solution of partial differential equations on surfaces. *J. Sci. Comput. 31,* 6, 4330–4350.

MAUCH, S. 2003. Efficient algorithms for solving static hamilton-jacobi equations. Ph.D. thesis, California Institute of Technology, Pasadena, CA, USA.

MCNAMARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Trans. Graph. 23*, 449–456.

MILLER, K. AND ROSS, B. 1993. *An introduction to the fractional calculus and fractional differential equations*. Wiley & Sons.

MUSETH, K., BREEN, D. E., WHITAKER, R. T., AND BARR, A. H. 2002. Level set surface editing operators. *ACM Trans. Graph. 21*, 330–338.

NARAIN, R., KWATRA, V., LEE, H.-P., KIM, T., CARLSON, M., AND LIN, M. C. 2007. Feature-Guided Dynamic Texture Synthesis on Continuous Flows. In *Eurographics Symposium on Rendering*. 361–370.

NARAIN, R., SEWALL, J., CARLSON, M., AND LIN, M. C. 2008. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graph. 27*, 166:1–166:8.

NIELSEN, M. B. AND BRIDSON, R. 2011. Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph. 30*, 83:1–83:8.

NIELSEN, M. B., CHRISTENSEN, B. B., ZAFAR, N. B., ROBLE, D., AND MUSETH, K. 2009. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 217–226.

PATEL, S., TESSENDORF, J., AND MOLEMAKER, J. 2009. Monocoupled 3D and 2D river simulations. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Posters Session*.

PHARR, M. AND HUMPHREYS, G. 2010. *Physically-Based Rendering: From Theory to Implementation*. Morgan Kaufmann.

PODLUBNY, I. 1999. *Fractional Differential Equations*. Academic Press.

RUUTH, S. J. AND MERRIMAN, B. 2008. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys. 227,* 3, 1943 – 1961.

SAVELSBERG, R. AND VAN DE WATER, W. 2008. Turbulence of a free surface. *Physical Review Letters 100*, 034501.

SCHECHTER, H. AND BRIDSON, R. 2008. Evolving sub-grid turbulence for smoke animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 1–7.

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

SHI, L. AND YU, Y. 2005. Taming liquids for rapidly changing targets. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 229–236.

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

TESSENDORF, J. 2004a. Interactive water surfaces. In *Game Programming Gems 4*. Charles River Media.

TESSENDORF, J. 2004b. Simulating ocean water. In *ACM SIGGRAPH Courses*.

TESSENDORF, J. 2008. Vertical derivative math for iwave. `http://people.clemson.edu/~jtessen/papers_files/verticalderivativesforiwave.pdf`.

TESSENDORF, J. 2011. Resolution independent volumes. In *ACM SIGGRAPH Courses*.

THÜREY, N., WOJTAN, C., GROSS, M., AND TURK, G. 2010. A multiscale approach to mesh-based surface tension flows. *ACM Trans. Graph. 29*, 48:1–48:10.

WARDETZKY, M., MATHUR, S., KÄLBERER, F., AND GRINSPUN, E. 2007. Discrete Laplace operators: no free lunch. In *Eurographics Symposium on Geometry Processing*. 33–37.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2009. Deforming meshes that split and merge. *ACM Trans. Graph. 28 (3)*, 9.

YU, J., WOJTAN, C., TURK, G., AND YAP, C. 2012. Explicit mesh surfaces for particle based fluids. *Computer Graphics Forum 31,* 2, 815–824.

YUAN, Z., CHEN, F., AND ZHAO, Y. 2011. Pattern-guided smoke animation with Lagrangian coherent structure. *ACM Trans. Graph. 30*, 136:1–136:8.

YUAN, Z., ZHAO, Y., AND CHEN, F. 2012. Incorporating stochastic turbulence in particle-based fluid simulation. *The Visual Computer (in press)*.

YUKSEL, C., HOUSE, D. H., AND KEYSER, J. 2007. Wave particles. *ACM Trans. Graph. 26*.

ZAKHAROV, V. E., L'VOV, V. S., AND FALKOVICH, G. 1992. *Kolmogorov Spectra of Turbulence I: Wave Turbulence*. Springer–Verlag.

ZHU, Y. AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. 24*, 965–972.