# WaveBlender: Practical Sound-Source Animation in Blended Domains

KANGRUI XUE, Stanford University, United States of America
JUI-HSIEN WANG, Adobe Research, United States of America
TIMOTHY LANGLOIS, Adobe Research, United States of America
DOUG JAMES, Stanford University, United States of America and NVIDIA, USA

Synthesizing sound sources for modern physics-based animation is challenging due to rapidly moving, deforming, and vibrating interfaces that produce acoustic waves within the air domain. Not only must the methods synthesize sounds that are faithful and free of digital artifacts, but, in order to be practical, the methods should be easy to implement and support fast parallel hardware. Unfortunately, no current solutions satisfy these many conflicting constraints.

In this paper, we present WaveBlender, a simple and fast GPU-accelerated finite-difference time-domain (FDTD) acoustic wavesolver for simulating animation sound sources on uniform grids. To resolve continuously moving and deforming solid- or fluid-air interfaces on coarse grids, we derive a novel scheme that can temporally blend between two subsequent finite-difference discretizations. Our blending scheme requires minimal modification of the original FDTD update equations: a single new blending parameter $\beta$ (defined at cell centers) and approximate velocity-level boundary conditions. Sound synthesis results are demonstrated for a variety of existing physics-based sound sources (water, modal, thin shells, kinematic deformers), along with point-like sources for tiny rigid bodies. Our solver is reliable across different resolutions, GPU-friendly by design, and can be 1000× faster than prior CPU-based wavesolvers for these animation sound problems.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: Sound synthesis, Computer animation, Finite-difference time-domain method

**ACM Reference Format:**
Kangrui Xue, Jui-Hsien Wang, Timothy Langlois, and Doug James. 2024. WaveBlender: Practical Sound-Source Animation in Blended Domains. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24), December 03–06, 2024, Tokyo, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3680528.3687696

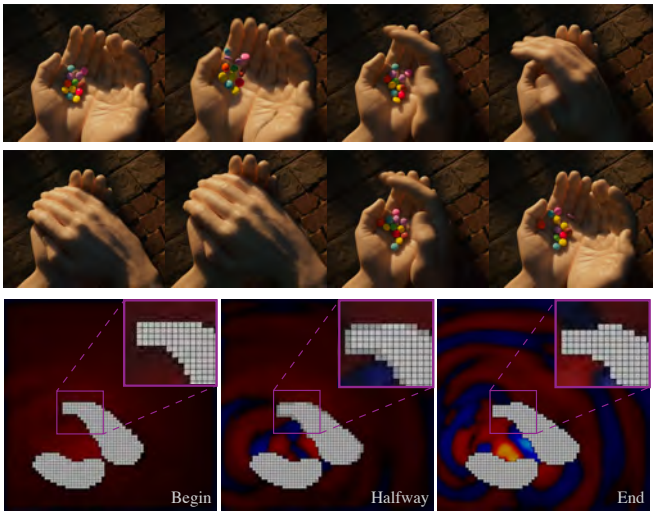



Fig. 1. **Candy Shake in Blended Hands:** Twelve hard candies are (Left to Right) bounced in (Top) open cupped hands, then shaken between (Middle) nearly closed hands. (Bottom) Given the rasterizations of two subsequent 60Hz animation frames, "Begin" & "End" (corresponding roughly to the top right image), WaveBlender simulates acoustic waves while continuously blending between these two discretizations, here shown resolving the acoustic emissions of excited Smarties modeled using point-like acceleration noise sources. The resulting candy-clicking sounds have a distinctly different character depending on hand pose due to the changing near-field wave scattering and cavity resonances.

## 1 Introduction

Sound sources in computer animation and virtual environments are often abstracted as "canned sound sources" or geometrically idealized as points or isolated rigid bodies. However, many staples of physics-based animation sound fundamentally involve motion and shape change that must be resolved to model the sound-radiation process. Important examples include: (i) deformable solids (such as noisy thin shells or occluders such as human hands), (ii) sloshing and splashing liquids (which emit sound differently depending on the shape of their liquid-air interfaces), and (iii) changes in acoustic scattering and diffraction from objects moving in close proximity (such as multibody systems, contacting solids, granular media, etc.). The time-varying shape of the air-domain acoustic medium, and the boundary excitations, produce interesting transients and reflections that are effectively captured using finite-difference time-domain (FDTD) wave simulations [Wang et al. 2018].

Unfortunately, capturing changes in the air domain due to boundary movement and deformation complicates time-domain acoustic wave simulations. First, the motion of boundaries through the grid and their vibration-specific boundary conditions must be handled in a perceptually plausible way for general sound-source animations. The scheme must be free from discretization artifacts over long durations, such as millions of audio time steps, which leads to the final design goal: the method should be simple, efficient, and suitable for GPU parallelism.

In this paper, we present WaveBlender, a simple and fast GPU-accelerated FDTD-based method for simulating animation sound

Authors' Contact Information: Kangrui Xue, Stanford University, United States of America, kangruix@stanford.edu; Jui-Hsien Wang, Adobe Research, United States of America, juiwang@adobe.com; Timothy Langlois, Adobe Research, United States of America, tlangloi@adobe.com; Doug James, Stanford University, United States of America and NVIDIA, USA, djames@cs.stanford.edu.


*SA Conference Papers '24, December 03–06, 2024, Tokyo, Japan*

ACM ISBN 979-8-4007-1131-2/24/12
https://doi.org/10.1145/3680528.3687696

sources on convenient uniform grids (see Figure 1). At each animation keyframe, the boundary surfaces (e.g., solid or liquid interfaces) are rasterized onto the grid, and simple first-order accurate boundary conditions are imposed. In order to support moving and deforming boundaries, we introduce the notion of a *blended domain* wherein an additional blending field (centered on cells and varying in time) $\beta \in [0, 1]$ is introduced to blend air (0) or solid (1) states between two animation keyframes. We construct a novel $\beta$-blended FDTD discretization of the pressure-velocity acoustic wave equations, which can be seen as a specific blending between the discretizations at two keyframes, along with approximate velocity-level vibration boundary conditions based on one-way source coupling.

Like prior FDTD acoustic wavesolvers that assume one-way source coupling [Wang et al. 2018], WaveBlender trades numerical accuracy for compatibility with existing animation sound-source models. The WaveBlender scheme is based on a first-order boundary handling scheme with staircased geometry, further trading accuracy for uniform grids and fully explicit timestepping. Nonetheless, we observe that WaveBlender synthesizes plausible low-noise sounds suitable for sound rendering applications. Examples include rigid-body modal sound sources, vibrating thin shells, coupled-bubble liquid sound sources, and kinematically deforming occluders with pre-recorded input audio. In addition to substantial speedups, we observe improved robustness on challenging examples compared to prior state-of-the-art animation sound techniques for dynamic interfaces [Wang et al. 2018]: the $\beta$-blended domains are more robust to topological changes in the air domain, such as when cavities open or close in liquid or solid simulations.

For scenes involving many small sound sources on coarse grids, WaveBlender supports user-generated point sources. In particular, we describe a point-like model for tiny acceleration noise sources (for impact-related "clicks"), such as those associated with granular or debris simulations, using equivalent wave-field forcing. We also introduce an auxiliary $\beta$-field for user-modeled occluders, such as to model changing cavity resonance when a container fills up with tiny grain-like rigid-bodies (see Figure 9).

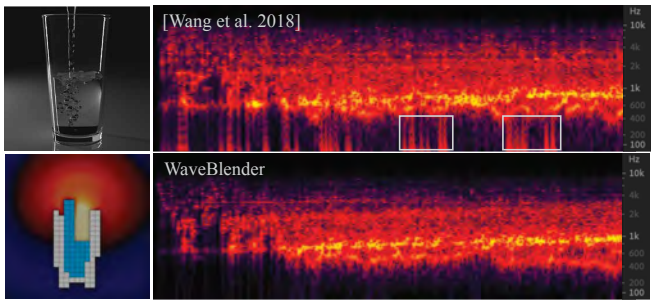



Fig. 2. **Avoiding Popping Artifacts:** A procedural "Glass Pour" animation is simulated using coarse cells ($\Delta x = 12.5$ mm) with both the [Wang et al. 2018] wavesolver and WaveBlender. While the [Wang et al. 2018] result suffers from repeated "popping" artifacts visible in the spectrogram's low frequencies (here, a few instances are highlighted), WaveBlender's $\beta$-blending scheme helps avoid discontinuities even when geometry is under-resolved.

## 2 Related Work

Sound rendering has a long history in computer graphics, animation, and movie post-production [Takala and Hahn 1992], such as creative use of studio recordings (e.g., Foley techniques), or procedurally generated audio effects and music (e.g., in interactive environments such as video games) [Cook 2002; Smith 1992]. However, within computer graphics, physics-based sound modeling techniques have only been developed in the last two decades [James et al. 2016].

One line of work focuses on modeling vibrational sound sources at the object or scene level, such as rigid and deforming solids [Chadwick et al. 2012a; O'Brien et al. 2001, 2002; Pai et al. 2001; van den Doel et al. 2001], fractured or crumpled solids [Cirio et al. 2016; Zheng and James 2010], thin shells [Chadwick et al. 2009; Cirio et al. 2018; Schreck et al. 2016], and water sounds [Langlois et al. 2016; Moss et al. 2010; Xue et al. 2023; Zheng and James 2009]. Oftentimes, these works employ simpler source-to-ear acoustic transfer models such as the popular frequency-domain precomputed acoustic transfer (PAT) method [James et al. 2006; Wang and James 2019].

On the other hand, a huge body of work is dedicated to modeling acoustic transfer: the physical process by which sound travels through and interacts with the surrounding environment until it reaches the listener. For larger scenes and band-limited sources, or when computational resources are severely limited, geometric acoustics methods [Chandak et al. 2008; Funkhouser et al. 1998, 1999; Kuttruff 2016; Savioja and Svensson 2015] are a popular choice. Wave-like diffraction effects can be added to these methods, as shown in [Schissler et al. 2014, 2021; Tsingos et al. 2001; Yeh et al. 2013]. Wave-based methods, on the other hand, are widely used due to their excellent broadband characteristics and higher accuracy [Mehra et al. 2012; Raghuvanshi et al. 2009, 2010]. For real-time applications, previous work has explored efficient precomputation-based representations, such as equivalent sources and multipole expansions [James et al. 2006; Mehra et al. 2013]. To reduce storage costs and speed up run-time field evaluation, other work has explored perceptual and parametric encoding [Raghuvanshi and Snyder 2014, 2018]. However, all these methods require either the precomputed objects to be in free-space isolation or the scenes to be static. This limits the application and realism in more complex near-field scenarios where waves scatter off other interfaces, similar to global illumination in visual rendering.

Recently, Wang et al. [2018] proposed a wave-based general-purpose acoustic transfer solver capable of simulating a wide range of animation sound effects. By abstracting sound sources as *acoustic shaders*, they showed how FDTD methods could capture perceptually important sound characteristics of objects deforming and interacting in the environment. Unfortunately, their wavesolver suffers from robustness issues and "popping" artifacts (see Figure 2), particularly when the topology of the involved thin interfaces is complex, such as in two-phase bubbly fluid simulations [Xue et al. 2023]. Moreover, the sparse matrix solution involved in resolving pressure updates is not GPU friendly. The sample-and-hold GPU wavesolver introduced in [Xue et al. 2023] alleviated robustness issues by freezing the geometry over regular time intervals, but its parallel-in-time approach incurs re-simulation overhead and lacks smoothly animated domains. In contrast, we introduce a new

blended domain FDTD wavesolver inspired by Allen and Raghuvanshi [2015], which developed a time-varying perfectly matched layer (PML) that can blend geometries – for example, tone holes – in and out to support dynamic 2D wind instrument simulations. WaveBlender was motivated by the fact that the original time-varying PML formulation performs poorly for continuously deforming interfaces and especially when Neumann boundary conditions must be applied, such as for vibrating sound sources.

Beyond computer graphics, acoustic methods for dynamic interfaces remain an active research topic; for example, see immersed boundary methods for acoustic scattering [Bilbao 2022], as well as dynamic grid models for the trombone in 1D [Willemsen et al. 2021] and thin plates in 2D [Willemsen et al. 2022]). Although the underlying theory is similar, our work differs in that we seek a unified "sound rendering" pipeline for simulating a wider range of animated acoustic phenomena, albeit with less source-specific modeling accuracy and without two-way source coupling.

## 3 Background

*Acoustic Wave Equation.* Consider a scene with objects $O$, surrounded by an acoustic medium $\Omega$, and let $\Gamma$ denote the boundary of $O$. Within the surrounding acoustic medium, $\mathbf{x} \in \Omega$, sound propagation is well-described by the set of coupled linear wave equations:

$$\frac{\partial p(\mathbf{x},t)}{\partial t} = -\rho_0 c_0^2 \nabla \cdot \mathbf{v}(\mathbf{x},t), \tag{1a}$$

$$\frac{\partial \mathbf{v}(\mathbf{x},t)}{\partial t} = -\frac{1}{\rho_0} \nabla p(\mathbf{x},t), \tag{1b}$$

where $p$ is the pressure perturbation, $\mathbf{v}$ the particle velocity, and $\rho_0$ and $c_0$ are the acoustic medium's density (1.204 kg/m$^3$ for air) and speed of sound (343.2 m/s for air) at rest. Here, $p$ and $\mathbf{v}$ are both functions of position and time and initialized to zero.

Normal surface accelerations, $a_n$ on $\Gamma$ generate outgoing waves, and this is usually modeled using Neumann boundary conditions using the normal component of (1b):

$$\frac{\partial \mathbf{v}(\mathbf{x},t)}{\partial t} \cdot \hat{\mathbf{n}} = -\frac{1}{\rho_0} \hat{\mathbf{n}} \cdot \nabla p(\mathbf{x},t) \Longrightarrow a_n = -\frac{1}{\rho_0} \partial_n p(\mathbf{x},t), \tag{2}$$

where $\hat{\mathbf{n}}$ denotes the unit normal vector. In velocity-level discretizations, we can specify the normal boundary velocity directly:

$$\mathbf{v}(\mathbf{x},t) \cdot \hat{\mathbf{n}} = v_b(\mathbf{x},t), \quad \mathbf{x} \in \Gamma. \tag{3}$$

*Discretizing the Equations.* We simulate the wave equation using the finite-difference time-domain (FDTD) method [Inan and Marshall 2011]. We use a staggered MAC grid, where pressure is defined at cell centers (integer indices) and velocity on cell faces (fractional indices), as is standard in acoustics and graphics [Bridson 2008].

Before timestepping, we rasterize the object $O$ onto the grid using a conservative rasterizer based on the triangle-box overlap test in [Akenine-Möller 2005]. We then identify "air" cells (entirely within the acoustic medium $\Omega$) and "solid" cells (either entirely within the interior of $O$, or that intersect the boundary $\Gamma$).

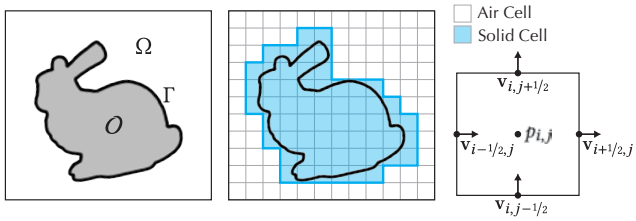



Fig. 3. **Scene Setup:** Given a sound-source object, our solver begins by rasterizing it onto a uniform staggered grid. **Cyan** lines indicate cell faces on the solid-air boundary where $\mathbf{v}_b$ is specified.

The FDTD update rule for step $n$ gives the pressure at the center and the velocity on the faces of an air cell at $(i, j, k)$:

$$p^{n+1} = p^n - \rho_0 c_0^2 \left(\tilde{\nabla} \cdot \mathbf{v}^{n+1/2}\right) \Delta t, \tag{4a}$$

$$\mathbf{v}^{n+1/2} = \mathbf{v}^{n-1/2} - \frac{1}{\rho_0} \left(\tilde{\nabla} p^n\right) \Delta t. \tag{4b}$$

Here, $\tilde{\nabla}$ denotes the discrete gradient operator. For example, in the x-component of the velocity update:

$$\tilde{\nabla} p_{i+1/2,j,k} = \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x}. \tag{5}$$

For faces on the solid boundary, the velocity is prescribed directly by the boundary conditions (3).

Assuming a uniform grid cell size $\Delta x$, stability holds if the timestep size $\Delta t$ satisfies the Courant-Friedrichs-Lewy (CFL) condition, $c\Delta t \leq \Delta x/\sqrt{\eta}$, where $\eta$ is the number of spatial dimensions (e.g., 3 for 3D).

*Blending Schemes from Prior Work.* Our work is inspired by that of Allen and Raghuvanshi [2015], which models interactive wind instruments in a 2D wavesolver. To support dynamic instrument geometry (e.g., opening and closing tone holes), the authors introduce a time-varying blending parameter $\beta$ at each cell. As important background, we first introduce how their blending scheme works. Consider the following modification to the velocity component of the wave equation (1) applied over the entire domain:

$$(1-\beta)\frac{\partial \mathbf{v}(\mathbf{x},t)}{\partial t} + \beta \mathbf{v}(\mathbf{x},t) = -(1-\beta)^2 \frac{1}{\rho_0} \nabla p(\mathbf{x},t) + \beta \mathbf{v}_b(\mathbf{x},t) \tag{6}$$

where $\beta \in [0,1]$. This modification essentially blends between the momentum equation[1] (1b) (when $\beta = 0$) and the enforcement of the normal boundary velocity $\mathbf{v} = \mathbf{v}_b$ (when $\beta = 1$). When $\beta$ varies in time from 0 to 1, the cell's velocity field shifts from being entirely affected by pressure gradients to being prescribed by boundary conditions. Using a semi-implicit scheme, this equation can be discretized to give the following velocity update:

$$\mathbf{v}^{n+1/2} = \frac{(1-\beta)\mathbf{v}^{n-1/2} - (1-\beta)^2 \left(\tilde{\nabla} p^n / \rho_0\right) \Delta t}{(1-\beta) + \beta \Delta t} + w_\beta \mathbf{v}_b^{n+1/2}. \tag{7}$$

The first term on the right corresponds to the original momentum equation, while the second term is contributed by the boundary condition $\mathbf{v}_b$ with weight $w_\beta = \frac{\beta \Delta t}{(1-\beta)+\beta \Delta t}$.

[1]The square term was empirically chosen by Allen and Raghuvanshi [2015] for producing natural-sounding transients in wind instrument modeling applications.

## 4 Our Blending Scheme

In this section, we derive a new blending scheme resulting in smoother updates that are better suited for dynamically moving sound sources.

### 4.1 WaveBlender FDTD Formulation

To motivate our formulation, we take a closer look at (7). Consider increasing $\beta$ linearly from 0 to 1 over some normalized time window $t \in [0, 1)$ (as in [Allen and Raghuvanshi 2015]). We show in Figure 4 that the velocity update weight $w_\beta$ is quite steep. Thus, even when $\mathbf{v}_b$ is prescribed by a simple translating monopole source (see Figure 5), such blending introduces unwanted distortion.

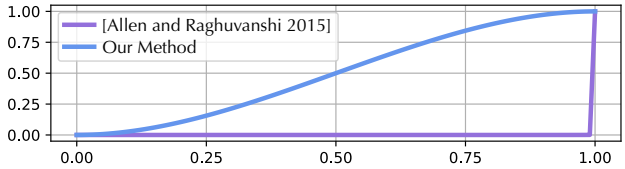



Fig. 4. **Velocity update weight** $w_\beta$ plotted against normalized blending time $t$ shows that our method smoothly blends in the boundary conditions, whereas the original "Aerophones scheme" (FDTD step rate at 128 kHz, blending over 10 ms windows) suffers from rapid changes near the end.

Based on this observation, we propose a modification: instead of blending in the continuous formulation, we blend in the discrete formulation. Intuitively, since we want the velocity updates to come smoothly from the boundary conditions, we should ease in $w_\beta$. Our proposed blending scheme for the FDTD update rule is therefore:

$$p^{n+1} = p^n - \rho_0 c_0^2 \left(\tilde{\nabla} \cdot \mathbf{v}^{n+1/2}\right) \Delta t, \tag{8a}$$

$$\mathbf{v}^{n+1/2} = (1-\beta)\left(\mathbf{v}^{n-1/2} - \frac{\tilde{\nabla} p^n}{\rho_0}\Delta t\right) + \beta \mathbf{v}_b^{n+1/2}. \tag{8b}$$

As before, when $\beta = 0$ (air cell), the velocity update comes from the momentum equation, and when $\beta = 1$ (solid cell), it comes from the boundary conditions. Contrasting this formulation with the previous one, we are now setting $w_\beta = \beta$, and therefore the choice of $\beta(t)$ directly controls the smoothness of the velocity updates. For all our results, we use the piecewise cubic smoothstep function: when blending from air to solid, we set $\beta(t) = 3t^2 - 2t^3$ over the normalized time window $t \in [0, 1)$; when blending from solid to air, we simply reverse the direction, i.e., $\beta(t) = 3(1-t)^2 - 2(1-t)^3$. Under this formulation, harmonic distortion is significantly reduced for the translating monopole source test case (see Figure 5).

Concretely, given the rasterized cell states at $t_1$ and $t_2$, we first normalize the blending window $[t_1, t_2)$ to $[0, 1)$ before computing $\beta$. We store $\beta$ at cell centers. For velocity updates, which are calculated on cell faces, we take the maximum $\beta$ of the two adjacent cells.

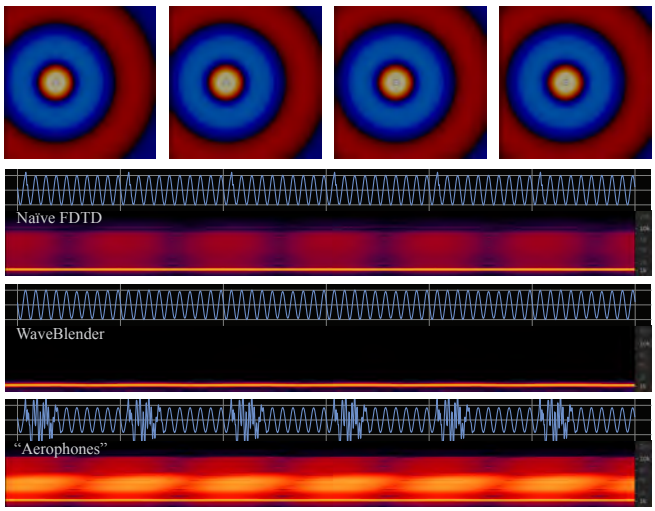



Fig. 5. **Comparison of uniform-grid methods on a moving monopole source** demonstrates clear grid-related artifacts ($\Delta x = 0.01$ m). (Top) A small cube (2 cells wide) pulsating at 1 kHz translates to the right at a constant speed (1 m/s) and is rasterized every 0.01 s. The listener is located 0.25 m to the right and travels with the cube at the same velocity. Naïve FDTD suffers from artifacts when boundaries discontinuously change, while these are mitigated by WaveBlender's $\beta$-blending scheme. Finally, direct application of the "Aerophones scheme" (7) also produces artifacts, in part because their method was not designed to support moving sound sources.

### 4.2 Properties of Blended Domains

The modified WaveBlender FDTD update equations (8) are defined over the entire domain. However, the choice of $\mathbf{v}_b$ can be ambiguous when cell faces do not coincide with the solid-air boundary (such as the faces indicated by dots in Figure 6). In these cases, we prescribe $\mathbf{v}_b^{n+1/2} = \mathbf{v}^{n-1/2}$ to prevent abrupt changes in the velocity field and to ensure impedance consistency. To see this, first note that (8b) can be rearranged in the following form with a term intentionally omitted (shown ~~crossed out~~):

$$\mathbf{v}^{n+1/2} = \mathbf{v}^{n-1/2} + \beta\cancel{\left(\mathbf{v}_b^{n+1/2} - \mathbf{v}^{n-1/2}\right)} - (1-\beta)\frac{\tilde{\nabla} p^n}{\rho_0}\Delta t. \tag{9}$$

In the continuous form, this represents the wave equation on an altered acoustic medium (the "$\beta$-medium"):

$$\frac{\partial \mathbf{v}(\mathbf{x}, t)}{\partial t} = -\frac{(1-\beta)}{\rho_0}\nabla p(\mathbf{x}, t) \triangleq -\frac{1}{\rho_\beta}\nabla p(\mathbf{x}, t), \tag{10}$$

where $\rho_\beta = \rho_0/(1-\beta)$ is the boosted effective density. For the pressure update to remain as in (8a), this increase in density implies a corresponding decrease in the speed of sound, $c_\beta^2 = (1-\beta)c_0^2$.

*Stability and Impedance of the $\beta$-Medium.* Following the above analysis, we note two additional observations:

- Since $c_\beta \le c_0$, the CFL condition for the modified WaveBlender FDTD update is always satisfied if the original CFL condition is satisfied, so the WaveBlender FDTD update remains stable.
- The specific acoustic impedance $z_\beta = \rho_0 c_0/\sqrt{1-\beta}$ determines the ratio between reflected and transmitted waves. By having a medium with uniform $\beta$ and thus uniform impedance, acoustic waves are guaranteed to move around freely in the blended cells. When $\beta$ approaches 1, the impedance asymptotically approaches infinity, reflecting waves perfectly as one would expect for a rigid boundary.

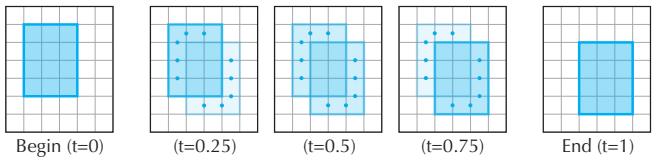



Fig. 6. **Blended Domains:** An object translates down and to the right, blending between two rasterized keyframes ("Begin" and "End"). Within the blended domain, **cyan** lines indicate cell faces on the solid-air boundary, while dots indicate cell faces where the choice of $\mathbf{v}_b$ is ambiguous.

We emphasize that modifying the density and speed of sound is nonphysical and strictly a byproduct of our blending scheme. When $\mathbf{v}_b$ does not have any ambiguity, we simply use the boundary condition. How this can be connected with various animation sound sources is illustrated next in §5.

## 5 Incorporating Sound Sources

With the blended domains wavesolver framework in place, we now discuss the computation of $\mathbf{v}_b$ for various sound-source phenomena. We adopt the *acoustic shader* abstraction first introduced by Wang et al. [2018] to represent specific sound source models and their physical properties. At runtime, WaveBlender queries each acoustic shader for the velocity $\mathbf{v}_b(\mathbf{x}, t)$ at each boundary point.

### 5.1 Zoo of Acoustic Shaders

Unlike Wang et al. [2018], however, we prescribe surface velocities (instead of accelerations) and implement our acoustic shaders on GPU (instead of CPU). We defer the modeling details to their original paper and instead focus on our changes.

*5.1.1 Pre-Recorded Sounds.* Perhaps the simplest application of our solver involves playing a pre-recorded sound through a planar area patch (i.e., a small speaker) to capture acoustic interactions with animating objects in the scene. Given an input audio signal $a_0(t)$ and a designated surface patch $\Gamma_0$, we set the normal surface acceleration $\mathbf{a}_n(\mathbf{x}, t) = a_0(t), \mathbf{x} \in \Gamma_0$. $\mathbf{v}_b(\mathbf{x}, t)$ is then obtained by numerical integration (trapezoidal rule) and copied to device memory.

*5.1.2 Rigid Bodies.* Using the modal sound pipeline, the dynamics of vibrating rigid bodies can be approximated by $M$ uncoupled oscillators, $\ddot{\mathbf{q}}(t) + \tilde{C}\dot{\mathbf{q}}(t) + \tilde{K}\mathbf{q}(t) = U^T f(t)$, where $\mathbf{q} \in \mathbb{R}^M$ denotes the vector of modal displacements, $\tilde{C}$ and $\tilde{K}$ are the reduced damping and stiffness matrices, and $U$ is the constant eigenmode matrix.

To compute $\mathbf{v}_b$, we first compute the projection of the modal matrix $U$ onto the boundary faces. Specifically, for each boundary face, we locate the closest point on the object's surface mesh and barycentrically interpolate the modal matrix weights from the vertices. This yields a "modal-to-boundary" transfer matrix, which we compute at the start and end keyframes of the blend and linearly interpolate at all times in between. $\mathbf{v}_b(\mathbf{x}, t)$ is obtained by multiplying the transfer matrix with the vector of modal velocities $\dot{\mathbf{q}}(t)$ on the GPU.

Acceleration noise profiles are computed during runtime identically to [Wang et al. 2018]. The velocity is numerically integrated (trapezoidal rule) from the acceleration profiles. This results in a single velocity vector for the rigid body, which is projected onto each boundary face's normal and added to $\mathbf{v}_b(\mathbf{x}, t)$.

*5.1.3 Thin Shells.* We adopt the harmonic shells model [Chadwick et al. 2009]. From Wang et al. [2018], we have a dataset of precomputed vertex displacements and accelerations for a given animation. To compute $\mathbf{v}_b(\mathbf{x}, t)$, for each boundary face, we locate the closest point on the object's surface mesh and take the average acceleration of the triangle's vertices, once at the start of the blend. The velocity is numerically integrated (trapezoidal rule) from the averaged accelerations and projected onto the boundary face's normal.

*5.1.4 Bubble-Based Water.* We use the analytical monopole model from [Xue et al. 2023], where each bubble's additive contribution to the boundary velocity is treated as a point source:

$$\mathbf{v}_b(\mathbf{x}, t) = \frac{(\mathbf{x} - \mathbf{x}_{\text{bub}}) \cdot \hat{\mathbf{n}}}{4\pi \|\mathbf{x} - \mathbf{x}_{\text{bub}}\|^3} \, \dot{v}_{\text{bub}}(t) = A_{\text{bub}}(\mathbf{x}) \, \dot{v}_{\text{bub}}(t). \tag{11}$$

Here, $\dot{v}_{\text{bub}}$ denotes the bubble's volume pulsation velocity. The spatial factors, $A_{\text{bub}}(\mathbf{x})$, are computed on the GPU and combined to produce a "bubble-to-boundary" transfer matrix, which we sample and hold at 1 ms intervals. $\mathbf{v}_b(\mathbf{x}, t)$ is obtained by multiplying the transfer matrix with the bubble volume velocities (collected into a vector) on the GPU. Unlike prior approaches from [Xue et al. 2023] and [Wang et al. 2018], we compute $A_{\text{bub}}(\mathbf{x})$ on the boundary faces directly, instead of on an intermediate water surface mesh.

### 5.2 Point Sources

Rasterizing geometry and imposing Neumann (or other) boundary conditions is one way to model sound sources, but it requires sufficiently fine grids and can be overkill for many simple sources used in graphics, especially particle systems and numerous small rigid bodies. Alternately, we can use generalized point-like models for both momentum (**F**) and divergence ($q$) sources.

Here we consider the special case of a spherical acceleration noise source. Adding a force density term to the momentum equation,

$$\frac{\partial \mathbf{v}(\mathbf{x}, t)}{\partial t} = -\frac{1}{\rho_0} \nabla \mathbf{p}(\mathbf{x}, t) + \frac{1}{\rho_0} \mathbf{F}(\mathbf{x}, t), \quad \mathbf{x} \in \Omega \tag{12}$$

we can introduce the point-like force density, $\mathbf{F} = -\mathbf{f}(t)\,\delta(x - \bar{x})$ to model a point-like acceleration noise source centered at $\bar{x}$. From Howe [2002], an equivalent forcing for a small sphere of radius $r$, volume $V_r$, undergoing acceleration $\mathbf{a}(t)$ is given by $\mathbf{f}(t) = 2\pi\rho_0 r^3 \, \mathbf{a}(t) = \frac{3}{2}\rho_0 V_r \, \mathbf{a}(t)$; in our implementation, for small nonspherical objects, e.g., ellipsoidal candies, we use this formula with $V_r$ replaced by the object volume. The force at $\bar{x}$ is then distributed to nearby MAC grid velocity locations using trilinear interpolation weights.

For stiff and tiny objects, the rapid linear acceleration profile $\mathbf{a}(t)$ is reasonably approximated using Hertz-like contact models for acceleration noise (see [Chadwick et al. 2012a,b] for details). Following (11) from [Chadwick et al. 2012b] we use $\mathbf{a}(t) = \frac{\pi}{2\tau} \Delta\mathbf{v} \, S(t; t_0, \tau)$, where the collision begins at time $t_0$, $\tau$ is the Hertz contact time scale, and $\Delta\mathbf{v}_i$ is the body's resulting velocity change. Here $S(t; t_0, \tau)$ is the acceleration "bump" profile, which we take to be a half-sine pulse, $S = \sin(\pi(t - t_0)/\tau)$ on $t \in [t_0, t_0 + \tau]$ [Johnson 1985].

## 6 GPU Implementation

By blending between fixed rasterizations over coarse rates, WaveBlender eliminates the need to re-rasterize the scene at every time

step. This feature, along with the use of uniform grids and fully explicit timestepping, unlocks an efficient GPU implementation. Here, we discuss implementation details of our WaveBlender system.

## 6.1 Batched FDTD Wavesolver

Given a simulation of length $T$, we divide it into fixed-size $T/B$ batches, where $B$ denotes the length of the batch. The batch length determines the rasterization (and hence blending) rate, and there is a trade-off between rasterization and overhead costs and the level at which we can resolve motion and limit blending-related discretization error. Larger batch lengths improve performance, but can also result in large volumes of cells blending at once.

At the start of each batch $[t_1, t_2)$, given the rasterized state at $t_1$, we compute the next rasterized state at $t_2$ using the process outlined in Figure 3. For each rasterization, we identify boundary points as the positions of cell faces that lie between solid and air cells. The complete set of boundary points $\mathcal{B}$ for the current batch is then given by the union of the boundary points at $t_1$ and at $t_2$. From here, all that remains is for the acoustic shaders to precompute $\mathbf{v}_b(\mathbf{x}, t)$ at all sampled times $t_1 \le t \le t_2$ and sampled positions $\mathbf{x} \in \mathcal{B}$. Shader samples are stored as a contiguous block of linear memory and mapped to their corresponding cells.

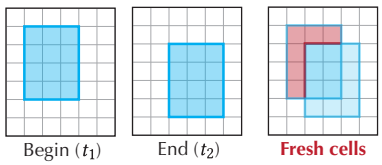
*Minimizing Host-Device Memory Transfer.* Since our shader memory footprints are small (several MB), the advantage of this approach is that all data transfer for the batch occurs upfront, reducing stalls. To further minimize data transfer, we precompute acoustic shader samples at a slower rate of 44.1–48 kHz, which is sufficient to capture all audible frequencies. Since our FDTD timestep rate is often higher (88.2 kHz and above for our target grid resolutions—as determined by the CFL condition), we linearly interpolate between shader samples in time.

Before rasterizing geometry, we check if each object has moved. If not, then rasterizing the object can be skipped, and if all objects have remained stationary, raster-related data transfer and shader memory management can be skipped altogether.

## 6.2 Per-Batch Overhead

In addition to rasterization and shader precomputation, the following procedures are performed at the start of each batch:

*6.2.1 Fresh Cell Extrapolation.* When solid cells become air cells after rasterization changes, the immediate pressure and velocity fields are undefined. This is commonly known as the "fresh cell problem" [Mittal and Iaccarino 2005], and the missing field values must be extrapolated from local boundary conditions and the field values of neighboring cells.




Similar to Wang et al. [2018], we compute fresh-cell pressures locally by enforcing the Neumann boundary conditions (2). Given a fresh cell centered at $\mathbf{x}_f$ and a neighboring air cell centered at $\mathbf{x}_i$ with pressure $p_i$, we require that the unknown fresh cell pressure $p_f$ satisfies:

$$p_f - p_i = -\rho_0 \, \mathbf{a}_n \cdot (\mathbf{x}_f - \mathbf{x}_i) \tag{13}$$

where $\mathbf{a}_n$ denotes the normal acceleration on the boundary. For fresh cells where multiple neighboring cells are air cells, we take the average of the computed $p_f$ values.

Additionally, our staggered MAC grid requires fresh cell velocities to be filled in. We find that setting each fresh cell velocity to the closest boundary velocity (as in [Cheny and Botella 2010]) can create spurious divergence sources. Instead, we perform a more expensive global minimization of fresh cell divergences. We set up a least-squares problem with one constraint per fresh cell, treating fresh cell velocities as unknowns, and solve it via QR decomposition; in our examples, we typically observe 10-1000 fresh cells per batch.

*6.2.2 Shader Velocity Re-initialization.* Shader samples for $\mathbf{v}_b(\mathbf{x}, t)$ are computed in the sound-source object's *local* frame, assuming that the object is acoustically isolated. When objects move into regions with nonzero velocity field buildup (e.g., from other sound sources), overriding the existing value of $\mathbf{v}(\mathbf{x}, t_1)$ at the start of the batch with the incompatible $\mathbf{v}_b(\mathbf{x}, t_1)$ introduces a discontinuity. Thus, it is desirable to reinitialize $\mathbf{v_b}(\mathbf{x}, t)$ to start at $\mathbf{v}(\mathbf{x}, t_1)$:

$$\mathbf{v_b}(\mathbf{x}, t) := [\mathbf{v_b}(\mathbf{x}, t) - \mathbf{v_b}(\mathbf{x}, t_1)] + \mathbf{v}(\mathbf{x}, t_1) \tag{14}$$

in order to align with the *global* velocity field. Note that an FDTD scheme using Neumann (acceleration) boundary conditions (2) will implicitly have boundary velocities initialized at $\mathbf{v}(\mathbf{x}, t_1)$.

*6.2.3 Runtime Cavity Detection.* In cases where the geometry of objects is under-resolved on coarse grids, closed cavities can artificially form in the rasterization. Over time, large nonphysical pressures can accumulate inside, which are then released into the simulation once these closed cavities happen to re-open. To address this, we perform a flood fill to detect grid regions that are disconnected from our listening position. These regions are treated as solid and $\beta$-blended as usual. However, since these cell volumes can be large, we exclude them from the fresh cell extrapolation when they re-open and initialize their pressure and velocity fields to zero.

## 6.3 Perfectly Matched Layer

When outgoing acoustic waves reach the edge of the simulation domain, special treatment is required to prevent them from reflecting back into the domain. As is common in FDTD wave simulations, we implement a perfectly matched layer (PML) at the domain boundary to artificially attenuate outgoing waves. Our implementation uses the split-field PML from [Liu and Tao 1997] (we found the non-split PML from [Allen and Raghuvanshi 2015] ill-suited for some of our examples due to significant null-space buildup of the velocity field). For all of our examples, we use a PML width of 8 cells.

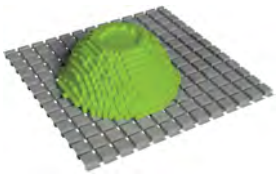
*Performance Considerations.* Given how small our domains are, the PML can occupy a significant portion (for instance, in an $80^3$ grid, nearly half of the cells belong to the PML). Instead of launching separate kernels for the interior region and the more arithmetic- and memory-intensive PML region (as in [Mehra et al. 2012; Wang and James 2019]), we find that fusing the kernels achieves higher performance on our hardware—especially for single-precision floating point. To reduce memory accesses [Micikevicius 2009], we enforce that the PML contains no objects (i.e., $\beta = 0$) and only compute the split pressure field for warps intersecting the PML.

Table 1. **Example Statistics:** Using identical cell sizes, step rates, and similar dimensions as [Wang et al. 2018], we compare our WaveBlender GPU timings with their original parallel-in-time CPU timings. The parallel-in-time method splits a simulation temporally into batches and distributes the batches across multiple machines, each running a CPU wavesolver instance; the number of machines is shown in parentheses (fourth-to-last column). In contrast, our WaveBlender implementation runs serially on a single machine. We show the full runtime, including example-specific data I/O and pre-processing costs (e.g., loading vertex accelerations from disk for thin shells and computing coupled-bubble velocities for bubble-based water), as well as the "core" WaveBlender timings (FDTD, shader, rasterization, and per-batch overhead). Here, "Real-time Factor" is defined as simulation runtime (core-only) divided by audio length and represents the remaining speedup needed to achieve real-time performance. For the "2016 Pouring Faucet" runtime comparison, where the [Wang et al. 2018] wavesolver was also run serially on a single machine, WaveBlender achieves a 1000x speedup.

| | | | | | | **Runtime: Full Comparison** | | **Runtime: Core-only** | |
|---|---|---|---|---|---|---|---|---|---|
| **Example** | **Length (s)** | **Cell Size (mm)** | **Dim.** | **Step Rate (kHz)** | **Blend Rate (Hz)** | Wang et al. [2018] (Parallel-in-time) | Ours (Full) | Ours (Core) | **Real-time Factor** |
| 2016 Pouring Faucet | 8.3 | 5 | $88^3$ | 192 | 100 | **55 hrs (x1)** | **2.53 min** | 1.30 min | 9.40x |
| 2016 Water Step | 4.2 | 5 | $90^3$ | 192 | 100 | – | 4.61 min | 0.87 min | 12.39x |
| Glass Pour | 5.32 | 12.5 | $48^3$ | 48 | 50 | – | 0.82 min | 0.20 min | 2.26x |
| Paddle Splash | 1.88 | 10 | $90^3$ | 96 | 100 | – | 11.77 min | 0.92 min | 29.36x |
| Blue LEGO Drop | 0.21 | 1 | $64^3$ | 615 | 1000 | 32 min (x10) | 0.33 min | 0.30 min | 85.71x |
| Spolling Bowl | 2.5 | 5 | $64^3$ | 120 | 1000 | 1.05 hrs (x8) | 1.97 min | 1.83 min | 43.92x |
| Cymbal | 2 | 10 | $80^3$ | 88.2 | 1000 | 53 min (x1) | 7.03 min | 1.48 min | 44.4x |
| Metal Sheet Shake | 10 | 14.3 | $100^3$ | 44.1 | 2000 | 24 hrs (x1) | 41.23 min | 10.39 min | 62.34x |
| Cup Phone | 8 | 7 | $88^3$ | 88.2 | 100 | 41 min (x20) | 0.33 min | 0.30 min | 2.25x |
| Cup Phone (low-res) | 8 | 12.5 | $48^3$ | 48 | 50 | – | **6.05 sec** | **5.82 sec** | **0.73x** |
| Talk Fan | 10.5 | 10 | $85^3$ | 88.2 | 1000 | 67 min (x20) | 3.35 min | 3.31 min | 18.91x |
| Trumpet | 11 | 10 | $80^3$ | 88.2 | 1000 | 33 min (x20) | 0.98 min | 0.96 min | 5.24x |
| Candy Fill'er Up | 7 | 5 | $60^2 \times 120$ | 120 | 60 | – | 1.58 min | 1.54 min | 13.20x |
| Candy Shake | 11 | 5 | $80^3$ | 120 | 240 | – | 2.60 min | 2.58 min | 14.07x |

## 7 Results

Please see our supplementary video for all results and comparisons to previous methods from [Wang et al. 2018; Xue et al. 2023]. We use the same mono auralization model as [Wang et al. 2018], with a single-point pressure sample used to estimate the far-field radiated sound. Despite the prototype nature of our implementation, we achieve greatly improved performance over prior methods due to GPU-accelerated FDTD and acoustic shaders. Example statistics are shown in Table 1, with detailed timing breakdowns given in Figure 7 for an Intel Core i9-13900KS 24-Core 3.2GHz, NVIDIA RTX 4090. Blend rates were empirically chosen to limit the blending layer to a few cells wide.



*Blending Cavities.* Aside from performance gains, we observe that the $\beta$-blending scheme along with our runtime cavity detection provide improved robustness and fewer "popping" artifacts, since cavities no longer instantaneously open/close as in previous methods. For example, (i) the "Spolling Bowl" cavities, which motivated [Wang et al. 2018] to use grooved floors (see inset) were unnecessary, and (ii) "2016 Water Step" previously suffered from rapid resolution-dependent thin-sheet cavity formation artifacts, but these can be avoided by clamping water boundary $\beta$ values to 0.9.

*Candy Examples.* The examples "Candy Shake" (Figure 1) and "Candy Fill'er Up!" (Figure 9) involve rigid-body hard candies simulated using "Houdini Bullet" at 2400 Hz, with filtered impulses used as acceleration noise point sources §5.2. Each candy is modeled as an oblate spheroid after "Smarties."

In the absence of acoustically absorbing boundary conditions for human skin ("Candy Shake") and wall losses ("Candy Fill'er Up!"), we implement a simple air-domain damping model to tame excessive resonances. Our model follows Equation 2 from [Allen and Raghuvanshi 2015]. We use damping values of $\sigma = 0.002/\Delta t$ for "Candy Shake" and $\sigma = 0.005/\Delta t$ for "Candy Fill'er Up!".

The "Candy Shake" demonstrates smoothly changing acoustic transfer effects as the shape of the hands changes. The "Fill'er up" shows 264 hard candies loudly filling a rigid container along with an auxiliary $\beta$-field used to approximate the accumulating candy grains. The time-varying container resonances can be seen in the spectrogram, and we also compare the simulated results without auxiliary $\beta$ (empty container) and without any geometry (see Figure 9).

## 8 Conclusion

We have introduced a new blended domain wavesolver framework suitable for computer animation sound rendering applications, which trades numerical accuracy for simplicity, robustness, and easily parallelizable uniform-grid FDTD wave simulations. The method temporally blends between FDTD discretizations of the acoustic wave equation in pressure-velocity form, with support for approximate velocity-level boundary conditions. In addition, we introduced idealized point source models for force-based modeling of subgrid objects, such as debris, to avoid grid refinement.

*Limitations and Future Work.* Our method is both simple and practical, but it has a number of inherent limitations and opportunities for future work. $\beta$-blending is fundamentally a heuristic approach, and blended domains with nonphysical slower sound speeds can adversely affect point sources placed inside. The scheme is based on

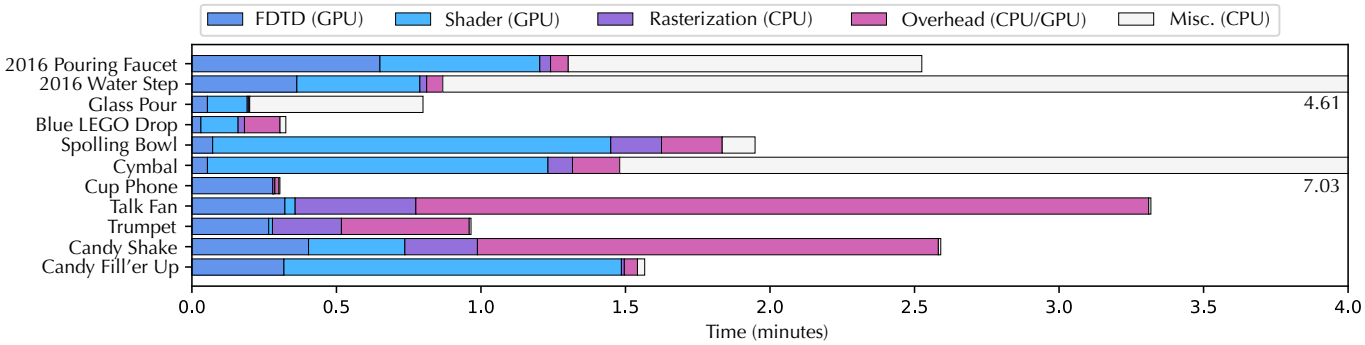



Fig. 7. **Timing Breakdowns:** Different acoustic shaders feature unique performance considerations. Here, "Overhead" refers to per-batch overhead (§6.2) and is largely dominated by fresh cell velocity QR solves on CPU, while "Misc." refers to miscellaneous example-specific data I/O and pre-processing costs excluded from the "core" WaveBlender timings. In examples where the blend rate is low (e.g., "2016 Pouring Faucet" and "Cup Phone"), GPU-based FDTD timestepping is the bottleneck. For other examples, rapid interface movements (such as in "Spolling Bowl", "Cymbal", and "Talk Fan") necessitate increased CPU-based rasterization and overhead costs, as well as more frequent shader evaluation and memory management.

first-order accurate spatial discretizations (complete with numerical dispersion and staircasing artifacts), and $\beta$-blending between domains further degrades accuracy. Future work should investigate discretizations with higher-order boundary handling and/or adaptivity, which can also support domain blending.

Like Wang et al. [2018], our acoustic shaders assume a simple one-way source coupling. For thin shells in particular, two-way coupling between vibrating plates and the surrounding acoustic field audibly affects the decay times [Chaigne and Lambourg 2001].

We introduce idealized point source models to model acceleration noise from small objects, but some objects may still contribute ringing content, and the objects they hit may also contribute other acoustic emissions, e.g., ground sound [Qu and James 2019].

Future work should extend boundary handling to support wall losses and absorption needed for more accurate modeling of reflections and taming of high-frequency parasitic resonant modes [Allen and Raghuvanshi 2015], while we currently only support air damping. Our solver is designed with vibration-based sources in mind, but other modeling applications, such as musical instruments, may require nonlinear wave effects [Allen and Raghuvanshi 2015].

In our research implementation, we use software rasterization and CPU geometry libraries to support a wide range of animation-sound datasets. However, optimized pipelines should port these CPU procedures onto the GPU. With FDTD timestepping no longer being the bottleneck, future work should also investigate GPU-friendly representations for sound sources themselves. Our implementation uses rectangular domains with a fixed PML, which may result in unnecessarily large volumes for some shapes. Future work should support dynamic domains that adaptively determine and track the sound source regions of interest. Finally, we hope that future research will lead to real-time integrated animation-sound systems.

## Acknowledgments

The authors thank the anonymous reviewers for their constructive feedback, Paulius Micivekius and Guillaume Thomas Collignon at Nvidia for GPU optimization assistance, Adobe and Meta for academic support, SideFX for donating Houdini licenses for academic research, and Maxon Redshift. This material is based on work supported by the Department of Energy, National Nuclear Security Administration under Award Number DE-NA0003968.

## References

Tomas Akenine-Möller. 2005. Fast 3D triangle-box overlap testing. In *ACM SIGGRAPH 2005 Courses* (Los Angeles, California) *(SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 8–es. https://doi.org/10.1145/1198555.1198747

A. Allen and N. Raghuvanshi. 2015. Aerophones in Flatland: Interactive Wave Simulation of Wind Instruments. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2015)* 34, 4 (Aug. 2015).

Stefan Bilbao. 2022. Immersed boundary methods in wave-based virtual acoustics. *J. Acoust. Soc. Am.* 151, 3 (March 2022), 1627–1638.

Robert Bridson. 2008. *Fluid Simulation for Computer Graphics.* A K Peters, Ltd.

J. N. Chadwick, S. S. An, and D. L. James. 2009. Harmonic Shells: A Practical Nonlinear Sound Model for Near-Rigid Thin Shells. *ACM Transactions on Graphics* (Aug. 2009).

J. N. Chadwick, C. Zheng, and D. L. James. 2012a. Faster Acceleration Noise for Multibody Animations using Precomputed Soundbanks. *ACM/Eurographics Symposium on Computer Animation* (July 2012).

J. N. Chadwick, C. Zheng, and D. L. James. 2012b. Precomputed Acceleration Noise for Improved Rigid-Body Sound. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)* 31, 4 (Aug. 2012).

Antoine Chaigne and Christophe Lambourg. 2001. Time-domain simulation of damped impacted plates. I. Theory and experiments. *The Journal of the Acoustical Society of America* 109, 4 (04 2001), 1422–1432.

A. Chandak, C. Lauterbach, M. Taylor, Z. Ren, and D. Manocha. 2008. Ad-frustum: Adaptive frustum tracing for interactive sound propagation. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (2008), 1707–1722.

Yoann Cheny and Olivier Botella. 2010. The LS-STAG method: A new immersed boundary/level-set method for the computation of incompressible viscous flows in complex moving geometries with good conservation properties. *J. Comput. Phys.* 229, 4 (2010), 1043–1076.

G. Cirio, D. Li, E. Grinspun, Mi. A. Otaduy, and C. Zheng. 2016. Crumpling sound synthesis. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 181.

Gabriel Cirio, Ante Qu, George Drettakis, Eitan Grinspun, and Changxi Zheng. 2018. Multi-scale Simulation of Nonlinear Thin-shell Sound with Wave Turbulence. *ACM Trans. Graph.* 37, 4, Article 110 (July 2018), 14 pages. http://www.cs.columbia.edu/cg/waveturb/

P. R. Cook. 2002. Sound Production and Modeling. *IEEE Computer Graphics & Applications* 22, 4 (July/Aug. 2002), 23–27.

T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West. 1998. A Beam Tracing Approach to Acoustic Modeling for Interactive Virtual Environments. In *Proceedings of SIGGRAPH 98 (Computer Graphics Proceedings, Annual Conference Series)*. 21–32.

T. A. Funkhouser, P. Min, and I. Carlbom. 1999. Real-Time Acoustic Modeling for Distributed Virtual Environments. In *Proceedings of SIGGRAPH 99 (Computer Graphics Proceedings, Annual Conference Series)*. 365–374.

M. S. Howe. 2002. *Theory of Vortex Sound.* Cambridge University Press.

Umran S. Inan and Robert A. Marshall. 2011. *Numerical Electromagnetics: The FDTD Method.* Cambridge University Press.

D. L. James, J. Barbic, and D. K. Pai. 2006. Precomputed Acoustic Transfer: Output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Transactions on Graphics* 25, 3 (July 2006), 987–995.

Doug L. James, Timothy R. Langlois, Ravish Mehra, and Changxi Zheng. 2016. Physically Based Sound for Computer Animation and Virtual Environments. In *ACM SIGGRAPH 2016 Courses* (Anaheim, California) *(SIGGRAPH '16)*. ACM, New York, NY, USA, Article 22, 8 pages. https://doi.org/10.1145/2897826.2927375

K. L. Johnson. 1985. *Contact Mechanics*. Cambridge University Press.

Heinrich Kuttruff. 2016. *Room Acoustics*. CRC Press.

T. R. Langlois, C. Zheng, and D. L. James. 2016. Toward Animating Water with Complex Acoustic Bubbles. *ACM Trans. Graph.* 35, 4, Article 95 (July 2016), 13 pages. https://doi.org/10.1145/2897824.2925904

Qing-Huo Liu and Jianping Tao. 1997. The perfectly matched layer for acoustic waves in absorptive media. *The Journal of the Acoustical Society of America* 102, 4 (10 1997), 2072–2082. https://doi.org/10.1121/1.419657

R. Mehra, N. Raghuvanshi, L. Antani, A. Chandak, S. Curtis, and D. Manocha. 2013. Wave-based sound propagation in large open scenes using an equivalent source formulation. *ACM Transactions on Graphics (TOG)* 32, 2 (2013), 19.

R. Mehra, N. Raghuvanshi, L. Savioja, M. C. Lin, and D. Manocha. 2012. An efficient GPU-based time domain solver for the acoustic wave equation. *Applied Acoustics* 73, 2 (2012), 83 – 94.

P. Micikevicius. 2009. 3D Finite Difference Computation on GPUs Using CUDA. In *Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units* (Washington, D.C., USA) *(GPGPU-2)*. ACM, New York, NY, USA, 79–84. https://doi.org/10.1145/1513895.1513905

R. Mittal and G. Iaccarino. 2005. Immersed Boundary Methods. *Annual Review of Fluid Mechanics* 37 (2005).

W. Moss, H. Yeh, J.-M. Hong, M. C. Lin, and D. Manocha. 2010. Sounding Liquids: Automatic Sound Synthesis from Fluid Simulation. *ACM Trans. Graph.* 29, 3 (2010).

J. F. O'Brien, P. R. Cook, and G. Essl. 2001. Synthesizing Sounds From Physically Based Motion. In *Proceedings of SIGGRAPH 2001*. 529–536.

J. F. O'Brien, C. Shen, and C. M. Gatchalian. 2002. Synthesizing sounds from rigid-body simulations. In *The ACM SIGGRAPH 2002 Symposium on Computer Animation* (San Antonio, Texas). ACM Press, 175–181.

Dinesh K Pai, Kees van den Doel, Doug L James, Jochen Lang, John E Lloyd, Joshua L Richmond, and Som H Yau. 2001. Scanning physical interaction behavior of 3D objects. In *Proceedings of the 28th annual conference on Computer Graphics and Interactive Techniques*. 87–96.

Ante Qu and Doug L. James. 2019. On the Impact of Ground Sound. In *Proceedings of the 22nd International Conference on Digital Audio Effects (DAFx-19)* (Birmingham, UK).

Nikunj Raghuvanshi, Rahul Narain, and Ming C Lin. 2009. Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Transactions on Visualization and Computer Graphics* 15, 5 (2009), 789–801.

N. Raghuvanshi and J. Snyder. 2014. Parametric wave field coding for precomputed sound propagation. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 38.

Nikunj Raghuvanshi and John Snyder. 2018. Parametric directional coding for precomputed sound propagation. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Nikunj Raghuvanshi, John Snyder, Ravish Mehra, Ming Lin, and Naga Govindaraju. 2010. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. In *ACM SIGGRAPH 2010 papers*. 1–11.

Lauri Savioja and U Peter Svensson. 2015. Overview of geometrical room acoustic modeling techniques. *The Journal of the Acoustical Society of America* 138, 2 (2015), 708–730.

C. Schissler, R. Mehra, and D. Manocha. 2014. High-order diffraction and diffuse reflections for interactive sound propagation in large environments. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 39.

Carl Schissler, Gregor Mückl, and Paul Calamia. 2021. Fast diffraction pathfinding for dynamic sound propagation. *ACM Trans. Graph.* 40, 4, Article 138 (jul 2021), 13 pages. https://doi.org/10.1145/3450626.3459751

C. Schreck, D. Rohmer, D. James, S. Hahmann, and M.-P. Cani. 2016. Real-time sound synthesis for paper material based on geometric analysis. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2016)*.

J. O. Smith. 1992. Physical modeling using digital waveguides. *Computer music journal* 16, 4 (1992), 74–91.

T. Takala and J. Hahn. 1992. Sound rendering. In *Computer Graphics (Proceedings of SIGGRAPH 92)*. 211–220.

N. Tsingos, T. Funkhouser, A. Ngan, and I. Carlbom. 2001. Modeling acoustics in virtual environments using the uniform theory of diffraction. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, New York, NY, USA, 545–552.

K. van den Doel, P. G. Kry, and D. K. Pai. 2001. FoleyAutomatic: Physically-based Sound Effects for Interactive Simulation and Animation. (2001), 537–544. https://doi.org/10.1145/383259.383322

Jui-Hsien Wang and Doug L James. 2019. KleinPAT: Optimal mode conflation for time-domain precomputation of acoustic transfer. *ACM Trans. Graph.* 38, 4 (2019), 122–1.

Jui-Hsien Wang, Ante Qu, Timothy R. Langlois, and Doug L. James. 2018. Toward wave-based sound synthesis for computer animation. *ACM Trans. Graph.* 37, 4, Article 109 (jul 2018), 16 pages. https://doi.org/10.1145/3197517.3201318

Silvin Willemsen, Stefan Bilbao, Michele Ducceschi, and Stefania Serafin. 2021. A Physical Model of the Trombone Using Dynamic Grids for Finite-Difference Schemes. In *2021 24th International Conference on Digital Audio Effects (DAFx)*. 152–159. https://doi.org/10.23919/DAFx51585.2021.9768286

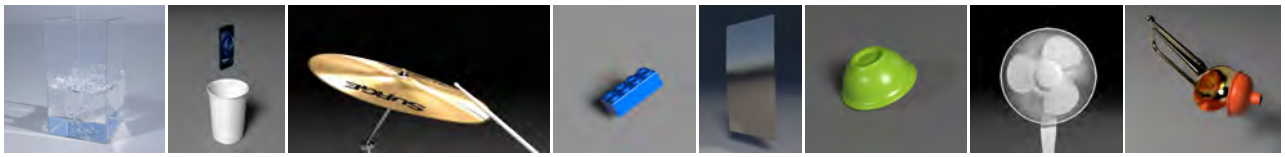
Silvin Willemsen, Stefan Bilbao, Michele Ducceschi, and Stefania Serafin. 2022. The Dynamic Grid: Time-varying Parameters for Musical Instrument Simulations Based on Finite-difference Time-domain Schemes. *Journal of the Audio Engineering Society* 70 (September 2022), 650–660. Issue 9.

Kangrui Xue, Ryan M Aronson, Jui-Hsien Wang, Timothy R Langlois, and Doug L James. 2023. Improved Water Sound Synthesis using Coupled Bubbles. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–13.

H. Yeh, R. Mehra, Z. Ren, L. Antani, D. Manocha, and M. Lin. 2013. Wave-ray Coupling for Interactive Sound Propagation in Large Complex Scenes. *ACM Trans. Graph.* 32, 6, Article 165 (Nov. 2013), 11 pages. https://doi.org/10.1145/2508363.2508420

C. Zheng and D. L. James. 2009. Harmonic Fluids. *ACM Transactions on Graphics (SIGGRAPH 2009)* 28, 3 (Aug. 2009).

C. Zheng and D. L. James. 2010. Rigid-Body Fracture Sound with Precomputed Soundbanks. *ACM Transactions on Graphics (SIGGRAPH 2010)* 29, 3 (July 2010).



Pouring Faucet Cup Phone Cymbal Lego Drop Metal Sheet Spolling Bowl Talk Fan Trumpet

Fig. 8. **Reference images for comparisons** with [Wang et al. 2018]. See the supplementary video for other examples.

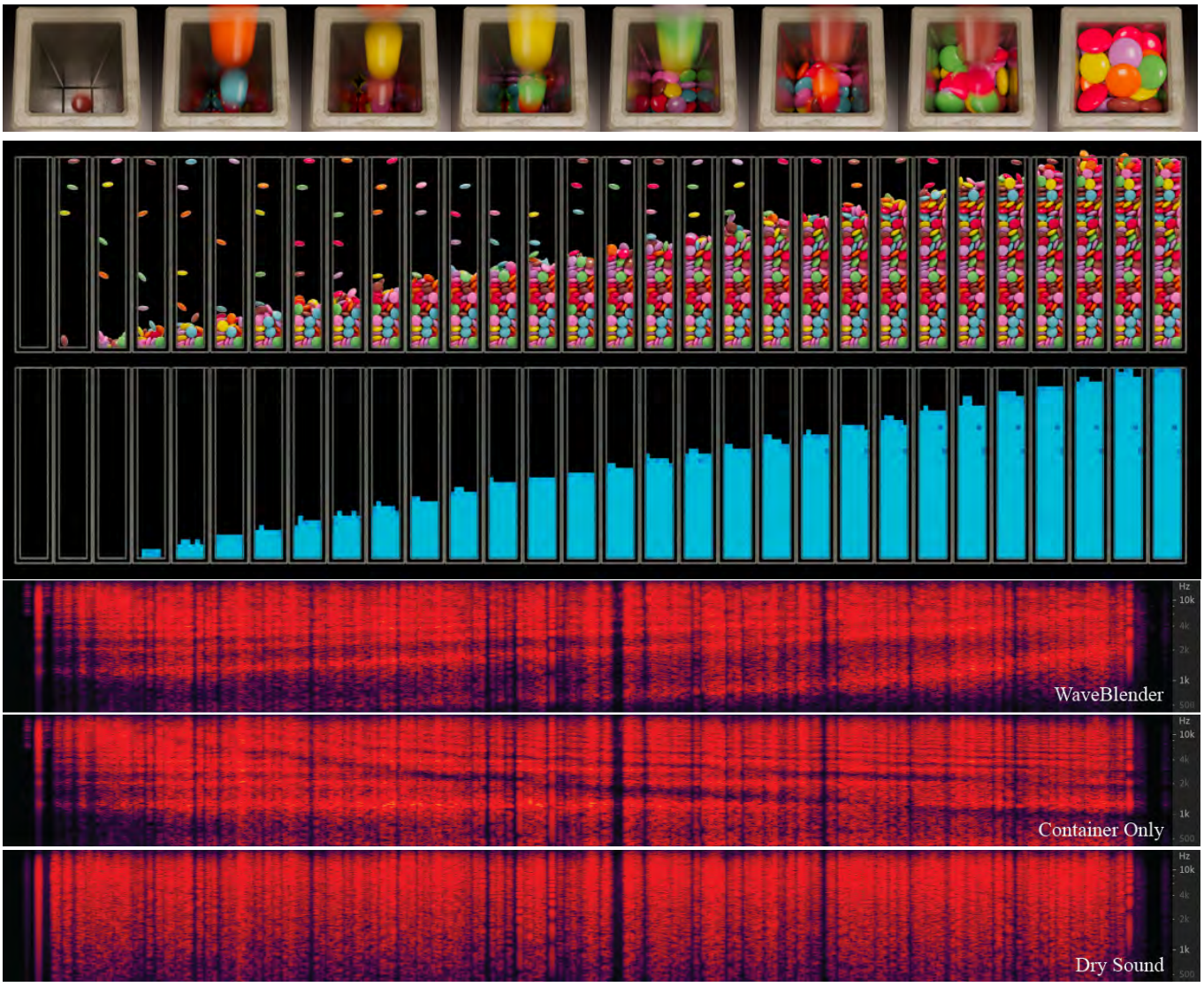



Fig. 9. **Fill'er Up!** A rigid-body simulation of 264 hard candies falling into a tube-like concrete container (3cm × 3cm × 20cm) generates 366832 contact impulses that are approximated as point-like acceleration-noise sources. Our WaveBlender acoustic wave simulation framework approximates such scenes on uniform grids but represents the changing air-domain shape using an auxiliary $\beta$ field, which can be used to model auxiliary scene geometry (in blue) in addition to the container. WaveBlender timesteps modified finite-difference time-domain (FDTD) equations and boundary conditions to approximate acoustic wave transport in smoothly blended domains. In addition to achieving robust, low-noise sound synthesis in dynamic scenes, its uniform grids ease GPU parallelization. The blended auxiliary geometry here allows us to efficiently capture the resonant character of the container filling up (see WaveBlender spectrogram). In contrast, without the auxiliary $\beta$ field (see "Container Only" spectrogram), only the empty container resonances appear. For reference, (Bottom) spectrograms of a "dry sound" show the wavesolver's response to point-source inputs without geometry and show strong impulse-like behavior.