KleinPAT: Optimal Mode Conflation For Time-Domain Precomputation Of Acoustic Transfer

JUI-HSIEN WANG, Stanford University
DOUG L. JAMES, Stanford University

We propose a new modal sound synthesis method that rapidly estimates all acoustic transfer fields of a linear modal vibration model, and greatly reduces preprocessing costs. Instead of performing a separate frequency-domain Helmholtz radiation analysis for each mode, our method partitions vibration modes into *chords* using optimal *mode conflation*, then performs a single time-domain wave simulation for each chord. We then perform *transfer deconflation* on each chord's time-domain radiation field using a specialized QR solver, and thereby extract the frequency-domain transfer functions of each mode. The precomputed transfer functions are represented for fast far-field evaluation, e.g., using multipole expansions. In this paper, we propose to use a single scalar-valued Far-field Acoustic Transfer (FFAT) cube map. We describe a GPU-accelerated vector wavesolver that achieves high-throughput acoustic transfer computation at accuracy sufficient for sound synthesis. Our implementation, KleinPAT, can achieve hundred- to thousand-fold speedups compared to existing Helmholtz-based transfer solvers, thereby enabling large-scale generation of modal sound models for audio-visual applications.

CCS Concepts: • **Computing methodologies** → *Modeling and simulation*; *Physical simulation.*

Additional Key Words and Phrases: Computer animation, sound synthesis, GPU, modal models.

ACM Reference Format:
Jui-Hsien Wang and Doug L. James. 2019. KleinPAT: Optimal Mode Conflation For Time-Domain Precomputation Of Acoustic Transfer. *ACM Trans. Graph.* 38, 4, Article 122 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322976

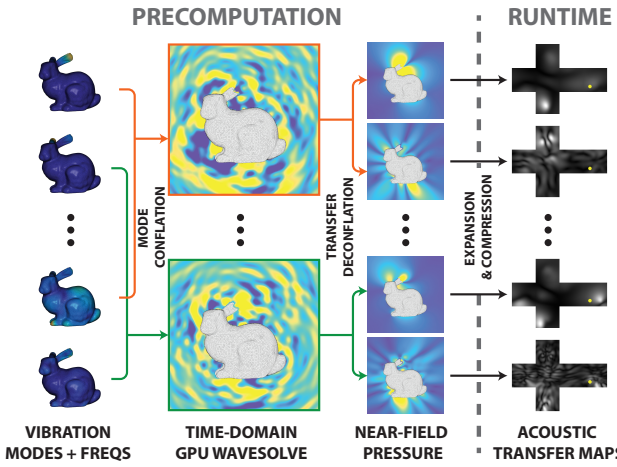


Fig. 1. **KleinPAT Overview:** Our method can evaluate acoustic transfer for 292 modes of this plastic bunny using only 6 time-domain wave simulations constructed by optimally conflating modes into 6 chords. The resulting sounds fields are deconflated to estimate the 292 transfer fields, then approximated with far-field acoustic transfer (FFAT) cube maps for real-time evaluation. This precomputed acoustic transfer (PAT) preprocess is over 2000x faster than traditional BEM-based approaches for the bunny.

PROLOGUE

Fritz Heinrich Klein created the first known all-interval twelve-tone row, the *Mutterakkord* (Mother chord), for his chamber-orchestra composition Die Maschine in 1921. Mother chord (see inset figure) is arranged so that it contains one instance of each interval within the octave [Wikipedia 2019]. Our KleinPAT algorithm optimally arranges different modal *tones* of a vibrating 3D object into *chords*, which are then *played* together by a time-domain vector wavesolver in order to efficiently estimate all acoustic transfer fields. All together now...

Authors' addresses: Jui-Hsien Wang, Stanford University, jui-hsien.wang@stanford.edu; Doug L. James, Stanford University.

0730-0301/2019/7-ART122 $15.00
https://doi.org/10.1145/3306346.3322976

1 INTRODUCTION

Physics-based modal sound synthesis is an effective and popular method for synthesizing plausible rigid-body contact sounds in computer animation and interactive virtual environments. Modal sound models use linear vibration modes to effectively characterize the surface vibrations of struck objects. The sound amplitude of each mode is described by its *acoustic transfer function*, whose spatial structure accounts for perceptually important wave effects like diffraction and radiation efficiency.

Unfortunately obtaining the acoustic transfer functions is a computational drag, since it involves solving the frequency-domain wave equation (Helmholtz equation) once for each vibration mode—there can be hundreds of audible modes even for simple objects. Precomputed Acoustic Transfer (PAT) methods were introduced to accelerate the otherwise costly *runtime* evaluation of transfer by precomputing each mode's exterior radiation field, and representing it in a form optimized for real-time evaluation. However the precomputation of acoustic transfer is still a computational bottleneck, even when using modern Helmholtz BEM solvers; hours to days of transfer precomputation can be required on large computing clusters, for objects of even small to moderate size, with higher frequency modes being more expensive. Currently, the high cost of

transfer precomputation makes high-quality modal sound synthesis impractical for many users and applications.

In this paper, we show how to accelerate frequency-domain transfer precomputation by orders of magnitude using a novel time-domain algorithm (see Figure 1) well-suited for GPU acceleration. We observe that it is possible to optimally pack the vibrational modes into several partitions for fast time-domain radiation solves, a process we call *mode conflation*. We refer to each group of conflated modes as a *chord*. The conflation algorithm runs in linear time, and guarantees the resulting wave field can be accurately decomposed back into separate radiation fields through a process of *transfer deconflation*. This decomposition is performed using a lightweight structure-exploiting QR solver specifically designed for this purpose. To further increase the throughput, we describe a novel GPU vector wavesolver that attempts fine-grain load balancing by interleaving the different solves. For fast runtime transfer representation, we propose a simple scalar-valued Far-field Acoustic Transfer (FFAT) map that approximates transfer amplitudes at accuracy suitable for sound rendering, and can be computed, stored, and evaluated efficiently.

Our proposed pipeline is designed with high-throughput acoustic transfer precomputation in mind, and can precompute an all-frequency modal sound model for typical 3D objects in a matter of minutes. We observe that the approach is a hundred- to thousand-fold faster than precomputation using modern BEM solvers, with negligible audible differences in the authors' experience.

2 RELATED WORK

Modal Sound Synthesis. Realistic modal sound synthesis has been a long-held goal in computer music, animation and HCI [Gaver 1993; Morrison and Adrien 1993; Takala and Hahn 1992; van den Doel and Pai 1996]. Related examples include audio-domain spectral synthesis techniques [Cook 2002; van den Doel and Pai 1996], and audio-domain methods based on recordings that "measure, analyze then synthesize" modal sounds [Pai et al. 2001; Ren et al. 2013; Serra and Smith 1990; van den Doel et al. 2001], and physics-based sound simulation methods based on linear modal analysis [Shabana 2012, 2013] of 3D discrete elastic models [O'Brien et al. 2002]. This work concerns the latter category for which 3D vibration models are available, and acoustic transfer techniques are applicable.

Precomputed Acoustic Transfer. While it is possible to display the modal oscillator values directly as sound [Bonneel et al. 2008; O'Brien et al. 2002; van den Doel et al. 2001; van den Doel and Pai 1996], they lack modal amplitude variations and spatialization effects due to acoustic wave radiation. Unfortunately simulating the wave equation is expensive, and incompatible with real-time sound rendering. Precomputed Acoustic Transfer methods were introduced to solve this problem [James et al. 2006]; they allowed frequency-domain radiation fields that satisfy the Helmholtz wave equation to be precomputed, e.g., by standard Boundary Element Method (BEM) solvers, then approximated by efficient representations (such as equivalent multi-point multipole (e.g., dipole) sources) that can be evaluated for real-time rendering. Later works often used a single-point multipole expansion with higher-order sources [Langlois et al. 2014; Zheng and James 2010, 2011] that was simpler to compute (since it avoids optimizing multipole placement). This method can limit the reconstruction accuracy in the near field, especially at higher frequencies where many multipole coefficients are required for nominal accuracy, thereby increasing runtime evaluation costs. Even faster runtime evaluation using texture mapping was first addressed by Chadwick et al. [2009]; they proposed Far-Field Acoustic Transfer Maps (FFAT Maps) as a way to use singular $1/r$ expansions to encode complex-valued angular radiation fields in textures precomputed using accurate fast Helmholtz multipole methods (FastBEM solver [Liu 2009]). FFAT Maps enable constant-time transfer evaluation, but can use hundreds of megabytes, something we improve upon in this work by using compressed scalar-valued amplitude maps. Various improvements have been made for modal sound models over the years, such as better object coupling, contact damping [Zheng and James 2011], compression of the modal matrix [Langlois et al. 2014], and the use of proxy soundbanks for non-object-specific sound modeling [Chadwick et al. 2012a; Zheng and James 2010]. However, the core PAT radiation precomputation remains mostly the same; because of the Helmholtz solvers required, it is by far the most expensive step in the pipeline.

Recently Li et al. [2015] proposed a method to enable interactive acoustic transfer evaluation to support interactive modal sound parameter editing–changing material parameters can change modal frequencies and necessitate transfer recomputation. The runtime speedup is achieved at the cost of additional precomputation: for *each* vibrational mode, they precomputed several BEM solves at different frequencies, and use these solutions to interpolate the radiation fields when the parameters are changed. Our work is complementary, in that we propose a fast alternative to slow per-mode BEM solves. Given the high speed of our method, sound models could be recomputed after object shape edits, enabling a much larger space for sound design.

Methods for water sound synthesis have used frequency-domain acoustic transfer functions to approximate air-domain amplitudes of thousands of harmonic bubbles [Langlois et al. 2016; Zheng and James 2009]. Recent work by Wang et al. [2018] showed that using a time-domain wave solver had the benefits of computing all bubble contributions in a single air-domain wave solve at less cost, while preserving the transient details. In a similar vein, we exploit the fact that many frequency-localized contributions can be evaluated, and decoded with a single time-domain wave solve.

BEM Acoustics. Boundary Element Methods for the Helmholtz wave equation are widely used in computational acoustics to obtain high-accuracy solutions for radiation and scattering problems [Ciscowski and Brebbia 1991; von Estorff 2000; Wu 2000]. Classical BEM populates a dense matrix and therefore has $O(N^2)$ memory requirements, and $O(N^3)$ solution cost using dense matrix solvers. To alleviate this cost, iterative solvers are used with fast approximate matrix-vector multiplication kernels, for which two popular competing methods with quasi-linear complexity exist: the fast multipole method (FMM), and hierarchical matrices ($\mathcal{H}$-matrices). Fast multipole methods rely on hierarchical gather/scatter operations on the Helmholtz fundamental solutions and related series expansions [Cheng et al. 2006; Gumerov and Duraiswami 2005; Liu 2009], whereas the hierarchical matrix approach is purely algebraic

and computes low-rank approximations of matrix partitions (see e.g., [Bebendorf 2008]). Exactly which one is faster and more memory efficient is problem and implementation dependent, but the $\mathcal{H}$-matrices approach with adaptive cross approximation [Bebendorf 2000] is known to be more parallel, and can perform faster approximate matrix-vector multiplications than FMM-based approaches [Bebendorf and Kriemann 2005; Brunner et al. 2010]. The latter is crucial for Krylov-subspace methods when solving large linear systems [Golub and Loan 2013]. We therefore chose an $\mathcal{H}$-matrices-based BEM solver as a baseline comparison to analyze the performance and accuracy of our method.

GPU Finite-Difference Time-Domain (FDTD). GPU-based time-domain wavesolvers have been studied and used in various applications, especially for wide-bandwidth problems, for example in computational room acoustics and musical instrument design [Bilbao 2009, 2011; Bilbao 2013; Bilbao and Webb 2013], head-related transfer-function computation [Meshram et al. 2014; Prepelita et al. 2016], seismic (elastic waves) and electrodynamics (electromagnetic waves) modeling [Komatitsch et al. 2010; Taflove and Hagness 2005]. Hardware-specific optimization techniques are available for different types of wavesolvers, e.g. in [Mehra et al. 2012; Micikevicius 2009]. The GPU is a great target for finite-difference time-domain (FDTD) simulations [Wang et al. 2018], and can reach real-time performance on bandlimited signals or 2D problems [Allen and Raghuvanshi 2015]. Inspired by many of these solvers, we design a highly specialized acoustic transfer GPU wavesolver that utilized new domain-specific algorithms to achieve high-throughput, many-mode PAT computation.

3 BACKGROUND

Background on modal sound synthesis and acoustic transfer are available in the SIGGRAPH physically based sound course [James et al. 2016]. We now describe notation and background required to explain the KleinPAT method.

3.1 Modal Vibrational Model

Linear modal analysis is standard in engineering and acoustics, and we use the same discretization and solution approaches of prior modal sound synthesis works in graphics [Zheng and James 2010, 2011]. The resulting modal model can describe the displacement of N surface vertices using a linear superposition of m vibrational mode shapes (restricted to the surface), $\hat{\boldsymbol{u}}_i \in \mathbb{R}^{3N}$, each associated with a vibration frequency $\omega_i = 2\pi f_i$. The surface displacement is given by the matrix-vector multiplication

$$\boldsymbol{u}(t) = [\hat{\boldsymbol{u}}_1\ \hat{\boldsymbol{u}}_2\ \cdots\ \hat{\boldsymbol{u}}_m]\boldsymbol{q}(t) = U\boldsymbol{q}(t), \tag{1}$$

where $U \in \mathbb{R}^{3N\times m}$ is the surface eigenmode matrix, and $\boldsymbol{q}(t) \in \mathbb{R}^m$ is the vector of modal coordinates $q_i(t) \in \mathbb{R}$. The dynamics of the modal coordinates are governed by m decoupled simple harmonic oscillators,

$$\ddot{q}_i(t) + (\alpha + \beta\omega_i^2)\dot{q}_i(t) + \omega_i^2 q_i(t) = \hat{\boldsymbol{u}}_i \cdot \boldsymbol{f}(t), \tag{2}$$

where (α, β) are Rayleigh damping parameters, and $\boldsymbol{f}(t)$ is a vector of applied surface forces. The oscillator dynamics for $\boldsymbol{q}(t)$ can be time-stepped efficiently and stably using an IIR digital filter [Hamming 1998] in a way suitable for real-time sound synthesis. We use the notation $\hat{\boldsymbol{u}}_i(\boldsymbol{x})$ to denote the i^{th} mode's surface displacement interpolated at $\boldsymbol{x}$, and, in the following, we use $u_{n,i}(\boldsymbol{x}) = \boldsymbol{n}(\boldsymbol{x})\cdot\hat{\boldsymbol{u}}_i(\boldsymbol{x}) \in \mathbb{R}$ to denote the normal component of displacement of mode i at $\boldsymbol{x}$.

3.2 Acoustic Transfer Function

Surface vibrations of an object O cause pressure fluctuation in the surrounding air, which radiates outwards as acoustic waves. The acoustic transfer function characterizes this radiation efficiency. More specifically, let Ω be the air medium containing O, and Γ be the boundary of O. Then the acoustic pressure p caused by the undamped, harmonically vibrating i-th mode $\hat{\boldsymbol{u}}_i$ is governed by the acoustic wave equation with the following Neumann boundary condition dependent on the normal surface acceleration $a_{n,i}$:

$$\frac{\partial^2 p(\boldsymbol{x},t)}{\partial t^2} - c^2\nabla^2 p(\boldsymbol{x},t) = 0, \quad \boldsymbol{x} \in \Omega \tag{3}$$

$$\partial_n p(\boldsymbol{x},t) = -\rho\, a_{n,i}(\boldsymbol{x},t) = \rho\,\omega_i^2\, u_{n,i}(\boldsymbol{x})\,\cos\omega_i t, \quad \boldsymbol{x} \in \Gamma, \tag{4}$$

where c is the speed of sound in air. The steady-state solution has only harmonic components

$$p(\boldsymbol{x},t) = c_i(\boldsymbol{x})\,\cos\omega_i t + d_i(\boldsymbol{x})\,\sin\omega_i t, \tag{5}$$

$$= \sqrt{c_i^2(\boldsymbol{x}) + d_i^2(\boldsymbol{x})}\,\cos(\omega_i t + \varphi_i(\boldsymbol{x})) \tag{6}$$

where (c_i, d_i) describes the pressure wave's amplitude $\sqrt{c_i^2 + d_i^2}$ and phase φ_i (which we ignore) at any point $\boldsymbol{x}$, and are effectively the acoustic transfer function. In the following, we will use the shorthand notation $p_i(\boldsymbol{x}) = \sqrt{c_i^2(\boldsymbol{x}) + d_i^2(\boldsymbol{x})}$ for the acoustic transfer amplitude of mode i, when it is not ambiguous.

Complex-valued form. In the literature, the acoustic transfer function for an harmonic vibration, $u_n(\boldsymbol{x})e^{i\omega t}$, is usually described using the complex value, $\bar{p}(\boldsymbol{x}) \in \mathbb{C}$ [James et al. 2006]. In this case, the time-domain pressure solution (5) is given by $p(\boldsymbol{x},t) = \Re(\bar{p}(\boldsymbol{x})e^{i\omega t})$, from which it follows that $\bar{p}(\boldsymbol{x}) \equiv c(\boldsymbol{x}) - i\,d(\boldsymbol{x})$, and the amplitude of the acoustic transfer function is the modulus, $|\bar{p}| = \sqrt{c^2 + d^2}$.

3.3 Runtime Sound Rendering

The runtime sound at any point $\boldsymbol{x}$ (in O's frame of reference) is approximated by the linear superposition of modal contributions,

$$\text{sound}(\boldsymbol{x},t) = \sum_{i=1}^{m} p_i(\boldsymbol{x})\, q_i(t) \tag{7}$$

where $p_i(\boldsymbol{x}) = |\bar{p}_i(\boldsymbol{x})|$ is mode i's transfer amplitude, that is used to scale the $q_i(t)$ modal coordinate. PAT methods usually precompute fast representations for evaluating $\bar{p}(\boldsymbol{x})$, then compute its modulus, whereas later we will approximate the transfer amplitude directly using nonnegative representations.

4 KLEINPAT MACHINERY

We have seen that a frequency-domain wave solver can compute a single harmonic acoustic transfer function, and so can a time-domain wave solver. However, by linearity of the wave equation, a time-domain wave solver can also, unlike its frequency-domain

counterpart, compute multiple *conflated* acoustic transfer functions of different frequencies at the same time in a *single* run. Let us exploit this fact.

4.1 Making Chords with Mode Conflation

Consider simulating the first m vibration modes, ω_i, $i = 1 \dots m$, at the same time, like playing a chord with m frequencies or notes. By linearity of the wave equation and acceleration dependence in the Neumann boundary conditions, if the normal surface acceleration a_n is a sum of the m modes' accelerations,

$$a_n(\boldsymbol{x}, t) = \sum_{i=1}^{m} a_{n,i}(\boldsymbol{x}, t) \tag{8}$$

then the pressure solution is also the sum of m frequency components,

$$p(\boldsymbol{x}, t) = \sum_{i=1}^{m} c_i(\boldsymbol{x}) \cos \omega_i t + d_i(\boldsymbol{x}) \sin \omega_i t, \tag{9}$$

$$= \begin{bmatrix} \cos \omega_1 t & \sin \omega_1 t & \cdots & \cos \omega_m t & \sin \omega_m t \end{bmatrix} \begin{pmatrix} c_1(\boldsymbol{x}) \\ d_1(\boldsymbol{x}) \\ \vdots \\ c_m(\boldsymbol{x}) \\ d_m(\boldsymbol{x}) \end{pmatrix} \tag{10}$$

$$= \boldsymbol{\tau}(t)^T \, \boldsymbol{s}(\boldsymbol{x}), \tag{11}$$

where $\boldsymbol{\tau}(t) \in \mathbb{R}^{2m}$ is the vector of trigonometric basis functions, and $\boldsymbol{s}(\boldsymbol{x}) \in \mathbb{R}^{2m}$ stacks the unknown transfer coefficients, $(c_i(\boldsymbol{x}), d_i(\boldsymbol{x}))$.

Conflate, Simulate, and then Deconflate. We refer to the packing of modes into a single wave solve as *mode conflation*. Provided that the chord's modal frequencies are distinct, the temporal basis functions, $\cos \omega_i t$ and $\sin \omega_i t$, are all linearly independent functions in time. Therefore the $2m$ coefficients $\{(c_i, d_i)\}$, and thus the transfer amplitudes, $p_i(\boldsymbol{x}) = \sqrt{c_i^2 + d_i^2}$, can be least-squares estimated from simulated temporal samples of $p(\boldsymbol{x}, t) = \boldsymbol{\tau}(t) \cdot \boldsymbol{s}(\boldsymbol{x})$. We refer to this latter process as *transfer deconflation* and it is addressed in §4.2.

Distinct Frequencies. The assumption of distinct frequencies is not a good one since many objects have repeated frequencies, or very similar ones in practice. These (nearly) degenerate frequencies are common, and are associated with geometric (near) symmetries, such as those of mirror, rotational, or cylindrical types [Langlois et al. 2014]. Unfortunately, similar eigenfrequencies lead to similar basis functions that are numerically ill-conditioned, and will foil transfer deconflation. Therefore we must ensure that chords used in mode conflation only involve frequencies that satisfy a frequency separation condition: each pair of frequencies f_i and f_j in a chord satisfies $|f_i - f_j| > \varepsilon > 0$, where ε is a user-specified gap parameter, e.g., $\varepsilon = 50$Hz. In summary, this frequency gap will ensure that least-squares transfer deconflation will be possible for the simulated chords, and numerically well-conditioned.

We now address these two central problems: (1) transfer deconflation (§4.2), and (2) optimal mode conflation (§4.3).

4.2 Transfer Deconflation

Given a temporal signal sampled from a superposition of harmonics with distinct frequencies that are known, we can estimate the amplitude (and phase) of the harmonics using least squares [Kay 1988]. In our problem, we want to temporally sample $p(\boldsymbol{x}, t) = \boldsymbol{\tau}(t) \cdot \boldsymbol{s}(\boldsymbol{x})$ to reconstruct the amplitudes, with two additional concerns: (1) the amplitudes must be estimated at many positions, $\boldsymbol{x}$, and (2) repeated periodically, e.g., after time T, in order to monitor amplitude convergence while time stepping the wave equation. Fortunately, these can be done very efficiently as follows.

Least-Squares Estimation of Amplitudes. Consider n pressure samples at $\boldsymbol{x}$ taken at uniformly spaced times, $t_i = t_0 + i\,\delta t$, $i = 1, \dots, n$, where δt is the temporal spacing, and t_0 is a waiting period since the time-stepping of the wave equation began at $t = 0$; in our implementation we use $\delta t = 4\Delta t$, and $t_0 = 0$, where Δt is the wavesolver's timestep size set to $1/(8 f_{max})$. From (11), we obtain n conditions on the coefficients $\boldsymbol{s}(\boldsymbol{x})$,

$$\boldsymbol{\tau}(t_i)^T \, \boldsymbol{s}(\boldsymbol{x}) = p(\boldsymbol{x}, t_i), \quad i = 1, \dots, n. \tag{12}$$

In matrix form this becomes

$$\begin{bmatrix} \text{—} \; \boldsymbol{\tau}(t_1)^T \; \text{—} \\ \vdots \\ \text{—} \; \boldsymbol{\tau}(t_n)^T \; \text{—} \end{bmatrix} \boldsymbol{s}(\boldsymbol{x}) = \begin{pmatrix} p(\boldsymbol{x}, t_1) \\ \vdots \\ p(\boldsymbol{x}, t_n) \end{pmatrix} \quad \Leftrightarrow \quad A\,\boldsymbol{s} = \boldsymbol{p}. \tag{13}$$

This linear least-squares problem, $A\,\boldsymbol{s} = \boldsymbol{p}$, has a unique solution if we have enough temporal samples, $n \geq 2m$, and it will be well conditioned provided that the frequencies are well separated–the sampling must also have sufficient resolution to avoid aliasing, which is achieved by construction. Observe that only $\boldsymbol{p}$ and $\boldsymbol{s}$ depend on the position $\boldsymbol{x}$, and not the basis matrix A. Therefore we can construct A's QR factorization once at cost $O(nm^2)$, and reuse it to estimate transfer coefficients $\boldsymbol{s}$ at arbitrarily many points $\boldsymbol{x}$. In our implementation, the initial QR factorization is performed on the CPU, and the following periodic estimation and the least-squares solves are performed on the GPU on-the-fly; we will explain how to choose n in a later section.

Efficient Periodic Estimation using Basis Factorization. While time-stepping the wave equation and sampling pressures at key positions $\boldsymbol{x}$, we can periodically invoke the QR solver to obtain transfer amplitude values, e.g., to monitor convergence. Unfortunately, these periodic estimates have different basis matrices, A, and therefore also necessitate periodic QR factorizations. Fortunately, this can be addressed as follows.

Consider the trigometric basis matrix A and one constructed after a period of time T later, named A_T. Due to trigonometric properties, these matrices are related by a rotation matrix:

$$A_T = A \begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_m, \end{bmatrix} \equiv A\,B \tag{14}$$

where B is block diagonal and orthogonal, and $B_i \in \mathbb{R}^{2\times 2}$ are small rotation matrices

$$B_i = \begin{bmatrix} \cos(\omega_i T) & \sin(\omega_i T) \\ -\sin(\omega_i T) & \cos(\omega_i T) \end{bmatrix}. \tag{15}$$

Therefore, given the thin QR factorization of $A = QR$, the solution to the new least-squares problem is

$$\boldsymbol{s} = (A_T)^\dagger \boldsymbol{p} = (AB)^\dagger \boldsymbol{p} = B^T A^\dagger \boldsymbol{p}, \tag{16}$$

(where $(\ )^\dagger$ denotes the pseudoinverse) which amounts to back-substitution with the original QR factorization, followed by a rotation by the block-diagonal B^T matrix. Therefore periodic estimates of transfer amplitude can be performed with negligible additional overhead, and only one QR factorization per chord is ever needed.

Adaptive Stopping Criterion. The efficient periodic transfer estimation allows us to stop the simulation whenever a stopping criterion is met, and to keep iterating if the problem is more challenging. This adaptive strategy increases both the efficiency and the robustness of our deconflation solver. The stopping criterion we use is the relative ℓ^2-norm of the least-square error for (13). The solver returns the latest transfer field after the error falls below the specified tolerance. In our examples, we check for convergence in non-overlapping sliding windows (hop size $T = n\,\delta t$); our error tolerance is set to 1%. We found that the convergence is usually fast (see Figure 2), and thus the actual errors in our examples fall well below this tolerance. We confirmed that the plateau for the convergence is purely due to machine precision (see §7.1.2).

4.3 Mode Conflation Algorithm

We now consider how to construct K chords that simulate all modes, but avoid closely spaced frequencies, so as to ensure that the trigonometric basis matrix, A, used in transfer deconflation is well-conditioned. Furthermore, since we also want to minimize the number of time-domain wavesolves we perform–one for each chord–we also seek to minimize the number of chords, K.

Chord Optimization Problem. Given the frequency distribution $\mathcal{F}$ (in the form of a sorted list)

$$\mathcal{F} = \{f_i \mid \text{Frequency of the } i^{\text{th}} \text{ vibrational mode}\}, \tag{17}$$

we seek to partition $\mathcal{F}$ into a minimal number of K partitions (chords), subject to the frequency *separability constraint*:

$$\text{Separability Constraint:} \quad \min_{(i,j) \text{ in same chord}} |f_i - f_j| > \varepsilon, \tag{18}$$

for some ε. This parameter ε directly affects the accuracy and the performance of the algorithm: if ε is set too low, the least-squares problem will have a poorly conditioned A matrix, which will lead to inaccurate transfer amplitude estimates, but if ε is set too high, the number of chords K needed to partition $\mathcal{F}$ will become large, and result in too many wavesolves. We will discuss how to choose ε in a later section.

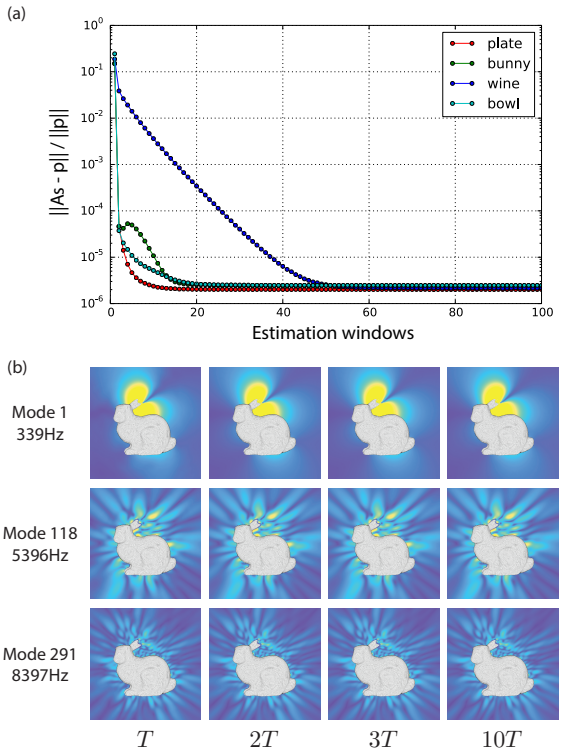


Fig. 2. **Convergence of transfer field in time**: The transfer fields of our test objects all converge rapidly. Here $T = n\,\delta t$ is the length and hop of the non-overlapping sliding window. (a) The overall least-square error converges quickly for all of our examples. (b) The estimated transfer fields for three modes in the same 100-mode chord for the "plastic bunny" (at the min, median, and max mode frequencies, respectively). Even though the first window (ending at time T) is polluted by pressure values starting at zero, surprisingly similar estimates are obtained, e.g., the average error between the first (T) and last (10T) windows are (0.4, 0.28, 0.09) dB for each of the three modes.

Linear-time Algorithm for Chord Optimization. We now show that it is possible to compute an optimal solution to the chord optimization problem (with minimal K chords) using a linear-time algorithm. The key observation is that this problem is an instance of a graph coloring problem for a special type of graph called the *indifference graph*. An indifference graph is an undirected graph where all vertices are assigned a real number (frequency), and an edge exists whenever two vertices are closer than a unit (ε). The coloring problem arises since any two vertices connected by an edge cannot be assigned the same color, where here colors represent chords. See Figure 3 for an illustration.

Fortunately, Looges and Olariu [1992] proved that a greedy coloring approach gives the optimal solution for indifference graphs. We provide an outline of the greedy algorithm here for completeness:

- Initialize colors $C = \{\}$.
- Scan through the vertices using the sorted order of $\mathcal{F}$.
- For each vertex v, find $c \in C$ not used by the neighbors of v; otherwise, color v with a new color c', and $C = C \cup \{c'\}$.

Table 1. **Mode/Chord Statistics**: Here, $\mathcal{F}$ is the list of modal vibrational frequencies in Hz; $\Delta\mathcal{F}$ is defined as the frequency difference between adjacent modes; kL is the non-dimensional wavenumber typically used to characterize the difficulty of the Helmholtz problem – k is the wavenumber and L is the object size. Conflation statistics include the parameters used in our algorithm and the distribution of modes in each chord and vector solve. For large objects (marked by *) we cannot interleave the chords due to memory constraints, and therefore the number of modes per solve is the "same" as per chord.

	Modal Model					Conflation Statistics						
Object	material	#modes	range $\mathcal{F}$	max kL	min $\Delta\mathcal{F}$	n	ε	min TBP	chords	#modes/chord	#vecsol	#modes/vecsol
Plate	ceramic	42	2.2kHz-14.9kHz	87.3	1.12Hz	512	50	0.59	4	[15, 15, 6, 6]	3	[15, 15, 12]
Wine glass	glass	22	2.1kHz-14.8kHz	51.5	1.29Hz	512	50	0.49	3	[10, 9, 3]	3	[10, 9, 3]
Bowl	wood	78	795Hz-14.7kHz	78.1	0.62Hz	512	50	0.57	3	[33, 32, 13]	3	[33, 32, 13]
Bunny	plastic	292	313Hz-8.4kHz	60.0	1.86Hz	512	50	0.82	6	[100, 90, 55, 30, 13, 4]	3	[100, 94, 98]
Backhoe Bucket*	iron	250	157Hz-10kHz	175.0	0.32Hz	128	270.6	1.0	13	[32, 32, 29, 29, 25, 24, 23, 19, 16, 11, 6, 3, 1]	13	same
Dragon*	poly.	642	337Hz-12.0kHz	133.0	0.21Hz	512	81.2	1.0	11	[120, 115, 105, 88, 69, 49, 40, 30, 15, 7, 4]	11	same
Engine Block*	steel	166	568Hz-9.9kHz	120.0	7.36Hz	128	270.6	1.0	9	[29, 27, 24, 24, 21, 19, 13, 5, 4,]	9	same

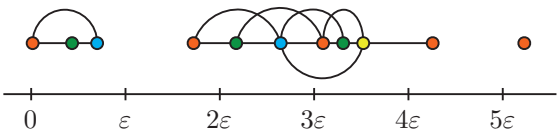


Fig. 3. **Computing the minimal number of chords** that satisfy the separability constraint can be done efficiently by (i) first encoding all the frequencies into the indifference graph, then (ii) running the greedy coloring algorithm, which is optimal for the indifference graph. Each of the colors represents a different chord comprised of well-separated modal frequencies in our optimal mode conflation.

This algorithm can be implemented efficiently with a stack that runs in linear time [Looges and Olariu 1992]. Using this simple algorithm, we observed negligible run time for computing the optimal conflation when compared to the wave solves and transfer estimation. We show the chord arrangement for our objects in Table 1.

Choosing the (de)conflation parameters using a Time-Bandwidth Product. Although we have three major parameters in our algorithm (n, δt, ε), their effects on the deconflation performance can be characterized using a single non-dimensional product, defined as

$$\text{TBP} = n\,\delta t\,\varepsilon = \frac{T_{\text{window}}}{T_{\text{beat}}}, \tag{19}$$

where $T_{\text{window}} \equiv n\,\delta t$ is the length of the sliding window, and $T_{\text{beat}} \equiv 1/\varepsilon$ is the inverse of the beat frequency caused by the two closest modal frequencies (See the inset of Figure 4). This time-bandwidth product (TBP) directly affects the stability of the least-square basis matrix A defined in (13). A basis with a high TBP has better conditioning and the resulting solves are more stable, whereas one with a low TBP (especially when lower than 1) can cause conditioning problems (see Figure 4). Empirically, we found that often the basis is stable enough for low TBP values, but for challenging problems it is better to use TBP = 1. Note that schedules with high TBPs are associated with higher computational costs (either with higher n, which increases memory usage for the QR solver, or with higher ε, which results in more chords to be solved and reduce the overall throughput); for example, see the numbers used in Table 1.

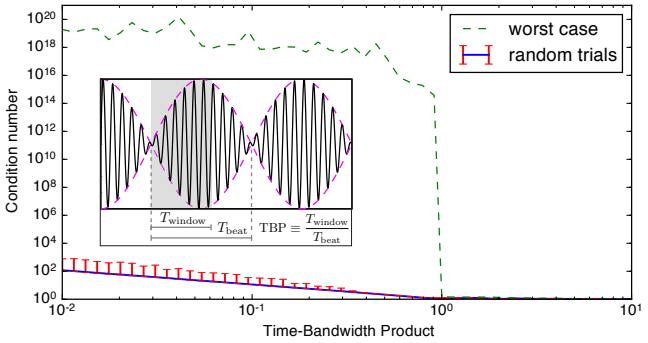


Fig. 4. **Time-Bandwidth Product**. We show that lower TBP increases the condition number of the least-square basis matrix on average. The experiments are randomized to have 4 – 50 modal frequencies randomly arranged in the basis before the condition number is computed; we performed in total 500 trials per TBP value; median, 10th, and 90th percentiles are reported (in blue line, and red bars, respectively). For each TBP, we also perform an analysis on the worst case scenario, where up to $n/2$ modes are inserted with uniform frequency gaps (at ε). This results in the densest modal frequency distribution. It is clear that for TBP < 1, the worst case scenario can result in unstable least-square solves for all TBPs; therefore, we advise against using values in this range without explicitly checking the rank of the basis . In this plot, we vary TBPs by varying ε; varying n and δt results in qualitatively similar behaviors. (Inset image) Here TBP directly measures the fraction of the beat (caused by two close frequencies) covered by the sliding-window samples $\boldsymbol{p}$ in (13).

5 GPU VECTOR WAVESOLVER

We now describe a novel GPU vector wavesolver that achieves high-throughput transfer computation. For example, this system allows us to efficiently estimate the dense sound field of the 292-mode plastic bunny (from Figure 1) in under 2 minutes—more than 2000x faster than a multi-threaded BEM solve—with accuracy suitable for sound rendering. We first introduce the idea of leveraging vector wavesolves for fine-grain load balancing in §5.1, then detail the defining features of our GPU wavesolver in §5.2.

5.1 Load Balancing Using Vector Wave Equation

Recall that given K chords, our goal now is to perform K separate time-domain wavesolves, each with the conflated boundary condition (BC) of form (8). We further observe that close frequencies

can occur in the high-frequency range, resulting in several chords having similar highest frequencies, and thus similar step rates. To utilize this property, we propose to group several of these solves together in one run when we can (see Figure 5), and use the same (min) step size and spatial discretization. This is mathematically equivalent to solving the discrete *vector* wave equation with BCs,

$$\frac{\partial^2 \boldsymbol{p}(\boldsymbol{x},t)}{\partial t^2} - c^2\nabla^2\boldsymbol{p}(\boldsymbol{x},t) = 0 \tag{20}$$

$$\partial_n \boldsymbol{p}(\boldsymbol{x},t) = -\rho \boldsymbol{a}_n(\boldsymbol{x},t). \tag{21}$$

Solving the vector wave equation on the GPU has several practical advantages over the scalar one: (1) it allows us to reuse potentially expensive geometry rasterization and other tedious bookkeeping overhead, such as cell index calculation, across vector components; (2) it increases per-thread compute density; (3) it reduces kernel launch overhead, which can be expensive; (4) it results in better caching behavior for fetching modal matrix, which can be sparse for a small chord. Although it is hard to isolate the gains for each factor, we observe an overall increased throughput when this strategy is applicable. Note that this is a *software* vectorization, while GPU has hardware vectorization such as Nvidia's Single Instruction Multiple Threads (SIMT) execution model. Our vectorization is more of an optimization that seeks to further balance the compute loads. Also note that we do not want to be overly aggressive about grouping, since it increases the memory stride, and burdens the GPU memory. Finally note that the vectorization here only affects the overall throughput, but not the accuracy of the solution at all, since all components of the vector $\boldsymbol{p}$ are solved independently.

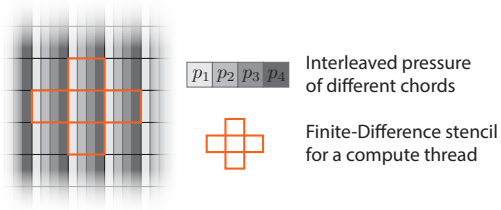


Fig. 5. **Interleaving the chords** into one vector wave solve can increase the amount of computation done by each thread, thereby increasing the throughput. It also helps us balance the loads for smaller chords.

In light of the goal of load balancing, we introduce a simple algorithm modified from the SortedBalance algorithm used for *makespan* minimization problems. It is a makespan problem because we can consider each machine an instance of wavesolve, each job a chord with conflated modes, and jobs assigned to the same machine will be run vectorized. We first sort the chords based on the number of modes conflated, which is our measure of compute load, e.g., it accounts for the higher cost of the dense matrix-vector multiply $U\boldsymbol{q}$ when computing the acceleration BC (8) for larger chords. We then loop through the list in order, and assign a job to the lightest machine *if the new processing time does not exceed the maximum process time of any machine at the time or the hardware resources.* Otherwise, we put it in a new machine. For completeness, we provide Algorithm 1 in the Appendix. The frequency distribution for two representative cases after this load-balancing procedure is given in Figure 6.

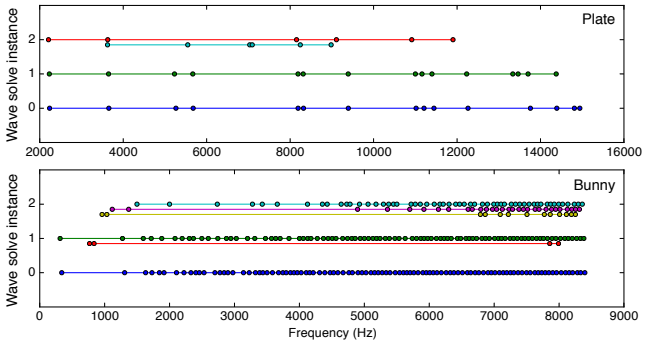


Fig. 6. **Chords after load-balancing** are solved in groups using vector wave solves. For both the Plate and the Bunny object, we show the frequency distribution after running the modified sorted balance algorithm. The colors correspond to the chords returned by the coloring algorithm.

5.2 GPU Finite-Difference Time-Domain Wavesolver

Our solver builds on the finite-difference time-domain (FDTD) discretization scheme with an absorbing boundary layer, similar to the one recently proposed for animation sound synthesis [Wang et al. 2018]. However, instead of aiming for a general system, we instead optimize the solver for GPU programming and for the transfer computation with a fixed object. For this purpose, we use a simple voxelized boundary discretization, and control the staircasing boundary error by sampling finer than the Shannon-Nyquist bound; in our results, we use at least 8 cells per shortest wavelength in any chord.

The key to efficient GPU implementation is to reduce the memory traffic while keeping all the threads busy. For example, it is usually preferred to have more work per thread, less off-chip memory read/write operations, regular/aligned/coalesced memory access, parallelizable workload, minimal branching instructions etc. Learned from the previous efficient GPU wavesolvers [Allen and Raghuvanshi 2015; Mehra et al. 2012; Micikevicius 2009], we further the efficiency of our system by introducing several new technical features suitable for fast transfer computation.

5.2.1 Hybrid discretization for improved PML. We use a hybrid finite-difference discretization to ensure fast wave simulation for the majority of the cells, and accurate grid-boundary absorption using perfectly matched layers (PML) in order to minimize artifacts caused by spurious wave reflections that might corrupt transfer deconflation. The inner domain, which contains the object, uses a pressure-only collocated grid with a lightweight 7-point stencil that is ideal for GPU SIMD/SIMT processing. For the outer domain, we use a pressure-velocity staggered grid to support accurate split-field PML [Liu and Tao 1997]. The split-field PML gives the better absorption than purely scalar pressure-only absorption models, because it damps waves in different directions separately. The inner and outer domains can be combined seamlessly and time-stepped separately. Note that if instead we use split-field staggered grid everywhere, such as in [Chadwick et al. 2012b; Liu and Tao 1997], the memory read/write operations required to update a single interior pressure

cell will increase by about 350%[1], and the memory allocation will increase by about 230%[2]. It is therefore beneficial to use a hybrid discretization (with only pressure values in the inner domain) despite the added implementation complexity.

5.2.2 Compact object bitset representation. During solver initialization the object is conservatively rasterized to the grid using a standard approach [Akenine-Möller 2002]. However, this rasterized representation will need to be accessed in various kernels to sample boundary conditions at every time step. It is therefore beneficial to use a compressed representation to reduce memory traffic.

To this end, we design a simple multi-pass algorithm to compress the rasterization into a bitset representation. The compression is fully parallel, and respects the SIMT execution model found in modern GPUs. Consider a list of size $|\mathcal{L}|$ of cell indices indicating rasterized cells. Our goal here is to compute a binary packing to $\lceil |\mathcal{L}|/\ell \rceil$ strings, each of ℓ-bit length. The naive approach might be to uniformly schedule elements of $\mathcal{L}$ to threads. However, due to the SIMT nature of the GPU execution, this scheduling will cause threads in different execution groups[3] to potentially process the same packed string, resulting in unnecessary communications and serialization. Instead, we first run a few passes to $\mathcal{L}$ to figure out the offset of each string, then schedule the processing accordingly. These passes consist of stream compaction, vectorized search, and adjacent differences, all of which can be done efficiently on the GPU, and the CUDA Thrust API provides related library support. After we have the optimal schedule, each thread execution group can process the binary representation independently and communication happens only within the group, which has low latency because they are typically scheduled on the same chip. An example of packing rasterization results of a 8-by-8 grid to 8-bit string can be found in Figure 7. In our implementation, we use $\ell = 32$ to leverage the 32-thread CUDA *warp*.

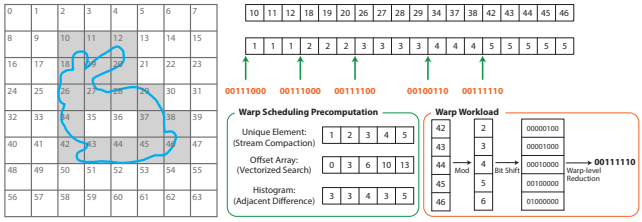


Fig. 7. **Rasterization bitmask** can be computed in parallel with a simple multi-pass algorithm. We first compute the optimal offset for each packed string, and schedule thread groups accordingly (green arrows). Once we have the schedule, we can process the binary representation for each cell index in parallel. The final reduction step happens at the thread-group level, which avoids slow inter-group communication. "Warp" here is synonymous with thread group.

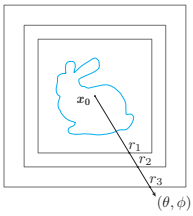
[1]Using the split-field, staggered grid to update a single pressure cell takes 15 (12R3W) + 16 (12R4W) = 31 global memory read/write operatons assuming the center cell value is cached; using the pressure grid takes 9 (8R1W) memory operations.
[2]Split-field PML technique damps pressure in each direction separately, therefore required the storage of 7 scalar fields $\{P_x, P_y, P_z, P_{\text{total}}, V_x, V_y, V_z\}$
[3]Threads that have the same instruction. This group is called a *warp* in Nvidia's CUDA programming model, and consists of 32 threads on most Nvidia GPUs.

6 FAR-FIELD ACOUSTIC TRANSFER MAPS

Given the acoustic transfer function computed in a small domain containing the object, we need a practical way to extend these solutions into the acoustic far field for sound rendering. Inspired by the complex-valued Far-Field Acoustic Transfer (FFAT) map introduced in Chadwick et al. [2009] to approximate $\bar{p}(\boldsymbol{x}) \in \mathbb{C}$, we ignore phase (since it is discarded anyway) and directly approximate the simpler squared transfer amplitude $p^2(\boldsymbol{x}) = |\bar{p}(\boldsymbol{x})|^2$, using a lightweight, real-valued expansion. Specifically, we approximate p^2 using a positive polynomial in $1/r$ with direction-dependent coefficients,

$$T(\boldsymbol{r}) \equiv \left(\sum_{i=1}^{\tilde{M}} \frac{\psi_i(\hat{\boldsymbol{r}})}{r^i} \right)^2 \approx p^2(\boldsymbol{r}), \tag{22}$$

where the radial expansion is evaluated with respect to the object's bounding box center point, $\boldsymbol{x}_0$. The functions ψ_i capture the directionality of the radiating fields, and the radial expansion has the correct asymptotic behavior as $r \to \infty$.



The coefficient ψ_i for a given angular direction is estimated using least-squares by taking $\tilde{N}$ samples at the intersection points between an emitted ray from $\boldsymbol{x}_0$, and concentric boxes that aligned with our solver grid (see inset figure). The boxes are geometrically expanded from the bounding box of the object: $R_i = 1.25^i$ for the expansion ratio of the i-th box. We found that $\tilde{M} = 1$ and $\tilde{N} = 3$ gives the most robust results and is efficient to compute, as the least-square solve per ray only involves two 3-vector dot products and a division. In contrast, higher $\tilde{M}$ values tend to overfit near-field fluctuations, and give worse far-field estimates. The directional parametrization of ψ_i is chosen to coincide with the largest box mesh, in order to reduce resampling issues. Pressure values on the other two boxes are bilinearly interpolated to the appropriate sample locations. Because of the chosen box parametrization, the resulting ψ_i field can be efficiently processed, compressed, stored, and looked up using standard image processing library. We found that the standard JPEG compression with medium quality setting (e.g., 65) works well with our data. The compressed 8-bit grayscale images of the 292-mode bunny FFAT maps take up only 3MB (see Figure 8). If desired, they can be spatially downsampled to provide more compression for lower fidelity use.

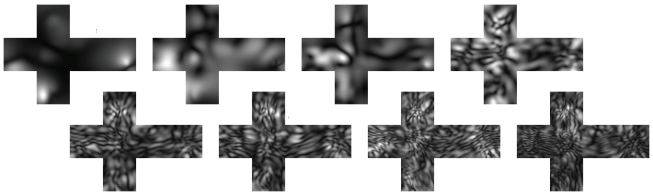


Fig. 8. **FFAT cube maps** are a convenient image-based representation for transfer amplitudes that exploit image compression. Maps shown are for the plastic Bunny model for mode 0 (0.31 kHz), 5 (1.03 kHz), 10 (1.50 kHz), 50 (3.48 kHz), 100 (5.00 kHz), 150 (6.14 kHz), 200 (7.07 kHz), 250 (7.82 kHz) (top to down, left to right)

Table 2. **Performance Statistics**: We list the geometric aspects of all our models, including the bounding sphere diameter L. All timings are given in terms of the total wall clock time for all-mode computations. The BEM solver runs on the CPU and uses all 48 possible threads, while our KleinPAT system utilizing conflated FDTD solves runs on a single consumer-grade GPU. Precomputation for BEM involves constructing and solving the integral equations (solve); the integrals are then evaluated at the field points for building the FFAT maps (eval); precomputation for FDTD incurs a single cost (total). Mode-averaged errors between BEM and FDTD solutions are provided along with the overall speedup. For the large objects (marked in *), we randomly sampled at least 10% of the modes for computing the BEM solutions. The average cost is then used to estimate the cost for computing all the modes.

	Geometry			BEM			FDTD	Comparison	
Object	#tri	#vtx	L	solve	eval	total	total	error	speedup
Plate	8186	4095	32cm	16.1 mins	385.0 mins	401.8 mins	0.6 mins	2.4dB	617.2x
Wine glass	20034	10019	19cm	206.8 mins	282 mins	234.7 mins	0.4 mins	2.3dB	592.6x
Bowl	32356	16180	29cm	258.7 mins	549.9 mins	808.6 mins	1.2 mins	0.86dB	676.1x
Bunny	36116	18060	39cm	2326.3 mins	1718.0 mins	4044.2 mins	1.85 mins	1.5dB	2186.8x
Backhoe Bucket*	99253	198506	96cm	31041.7 mins	19441.7 mins	50483.3 mins	27.71 mins	2.9dB	1821.9x
Dragon*	33267	66534	60cm	7383.0 mins	18018.8 mins	25401.8 mins	18.30 mins	2.1dB	1388.3x
Engine Block*	43817	88058	66cm	13916.3 mins	7829.7 mins	21746.0 mins	9.19 mins	0.64dB	2367.5s

7 RESULTS

Our KleinPAT solver is capable of delivering the full modal sound models end-to-end in a matter of minutes for all the objects we tested on, and reaches thousand-fold speedups compared to a modern parallel BEM solver. To make this number more concrete, we estimated all 292 distinct radiation fields of a 40cm plastic bunny within 2 minutes, while the BEM solver running in parallel on a high-end desktop finished after more than 2 days and 19 hours. There are no audible differences the authors could detect between the two solutions. The full performance and object statistics are summarized in Table 2. We note that comparable precomputation times can be found in other sound papers [Li et al. 2015; Zheng and James 2010] with similar objects but different BEM solvers.

We carefully validated our solver against a trustworthy BEM solver in several ways. First, we show that the sounds generated by FDTD and BEM solvers are similar in static impulse tests and in an animation sequence where there are numerous impacts on the object. Second, we systematically analyze the error introduced by the conflation/deconflation algorithm, by the FDTD solve, and by the far-field expansion model. Please see the accompanying video for the sound results.

7.1 Implementation Details

In this section, we provide implementation details including libraries used for the BEM solves, and computing hardware.

7.1.1 BEM solver. We use the well-maintained, open-source BEM solver `Bempp` [Śmigaj et al. 2015] for all our related calculation. It is based on the hierarchical matrix compression with Adaptive Cross Approximation (ACA) [Bebendorf 2000] to avoid populating a dense BEM matrix, and to speed up matrix-vector multiplications crucial in iterative solvers. Since the BEM matrix is not symmetric, we use the standard GMRES solver [Saad and Schultz 1986], and the recommended error tolerance 10^{-6} to ensure trustworthy solutions. Note that the BEM performance is only mildly sensitive to this tolerance (see Figure 9), since in many cases constructing the boundary integrals and evaluating the far-field solutions dominate the costs. We also found that the default maximum rank of 30 for the ACA approximation is insufficient and will introduce blocky artifacts, and therefore increased it to $60 - 100$ for our examples. `Bempp` is CPU based and runs in parallel. Due diligence was done to ensure fair comparisons.

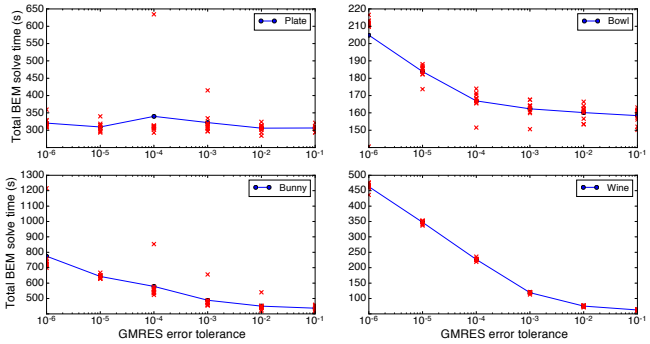


Fig. 9. **GMRES error tolerances:** To study the effects of the GMRES error tolerance on `Bempp` solve time, we ran multiple BEM solves at each of the tolerance settings. The averaging is needed since the ACA algorithm is randomized. Lower tolerance (e.g., 10^{-1}) introduces higher error in the solution, while only improving the solution time mildly compared to our default setting at 10^{-6} (speedups: 1.05 for Plate, 1.29 for Bowl, 1.77 for Bunny, and 7.37 for Wine). Blue lines go through the median data points for each tolerance level.

7.1.2 Computing hardware and implementation details. Our computing hardware for CPU is a single workstation equipped with dual Xeon E5-2690V3 2.6GHz 12-core processors (24 physical cores and can be hyperthreaded to run 48 threads). Our GPU hardware is a single GeForce GTX Titan X graphics card (1.08GHz clock speed, 3072 CUDA Cores, 12GB memory). Our implementation is written in C++ and CUDA. CUDA 9.2 is used in order to support the warp-level intrinsics we used to compute the rasterization bitmask. All BEM computation is done on the CPU using all hardware threads; the wavesolver runs on the GPU, and FFAT maps are computed and evaluated on the CPU.

7.2 Error Analysis

We evaluate the error of our method in this section. Since there are potentially multiple sources of error, we discuss them separately.

7.2.1 Error metrics. To quantify the error in the perceived loudness, we use the spatial average of the difference between two sound pressure amplitudes, $p(\boldsymbol{x})$ and $p'(\boldsymbol{x})$, in decibels (dB),

$$\text{error} = \text{avg}_{\boldsymbol{x}} \left| 20 \log \left(\frac{p(\boldsymbol{x})}{p'(\boldsymbol{x})} \right) \right|. \tag{23}$$

Note that the just noticeable difference (JND) in amplitude for a single tone in an A/B comparison could be on the order of 1 dB, however it becomes much harder for people to hear that same amplitude difference for a tone when there are many tones played [Zwicker and Fastl 2013].

7.2.2 Mode conflation error analysis. The core idea of our paper is to amortize the wave solve cost by conflating vibrational modes into chords. Non-harmonic parts of the signal for a mode, such as numerical dispersion error or transient noises, can leak into other modes in the same chord and cause transfer estimation error. We avoid this problem by ensuring a proper frequency separation in each chord. We show that this strategy introduced small errors (see Figure 10). The error is measured by comparing the conflated solves (with many modes per solve), and single-mode solves on the per-mode basis. We found that the error is less than 0.4dB for all modes, and the radiation fields visually identical. We have shown the rapid temporal convergence of deconflated transfer estimates earlier in Figure 2.

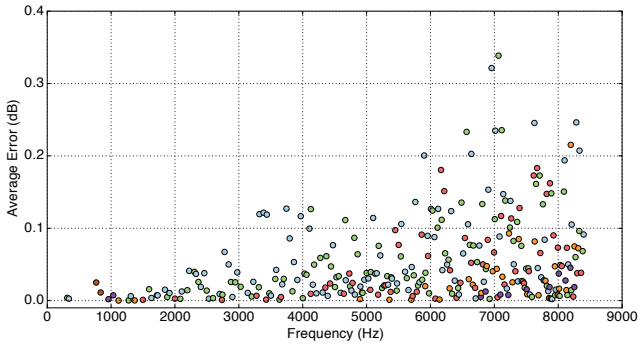


Fig. 10. **Errors introduced by the mode conflation process:** For the bunny, all the modes can be conflated into 6 chords (each with a different color). The conflation/deconflation introduces error well below 0.5dB, while making the transfer estimation 36 times faster than a per-mode FDTD solve.

7.2.3 FDTD/BEM error analysis. We compare the transfer values estimated from the conflated FDTD solves and BEM solves on the same mesh placed inside the FDTD grid. The mode-average error is shown in Table 2, and is generally found to be within 1-3dB. Although the pointwise error is not negligible, our conflated solve captures the distinctive radiation fields well. Some representative FFAT Maps for the ceramic Plate object are shown in Figure 11. The resulting sounds rendered using FDTD/BEM FFAT maps have inaudible differences for all examples in the authors' experience; this is supported by the spectrogram and the waveform comparison shown in Figure 12 for the bunny drop scene.

7.2.4 Far-field evaluation error analysis. We proposed to estimate the far-field transfer amplitude using a FFAT cube map based on a positive polynomial expansion. We compared this approach to the more widely used single-point multipole expansion [Langlois et al. 2014; Li et al. 2015; Zheng and James 2010, 2011], and found that generally our approach not only runs faster due to the $O(1)$ cost (for both evaluation and data fetching), but it is also more accurate (see Table 3). One potential downside for using the FFAT cube maps is the larger memory cost. However, we demonstrate that with the existing image compression techniques, we are able to store the FFAT cube maps very compactly. The 8-bit JPEG compressed at a modest quality setting has less memory requirement than the multi-pole model, and results in almost no audible degradation compared to the uncompressed version in the authors' experience. In total, the FFAT cube map takes up no more than a few MB per object, making it of similar memory requirement compared to high-quality visual textures.

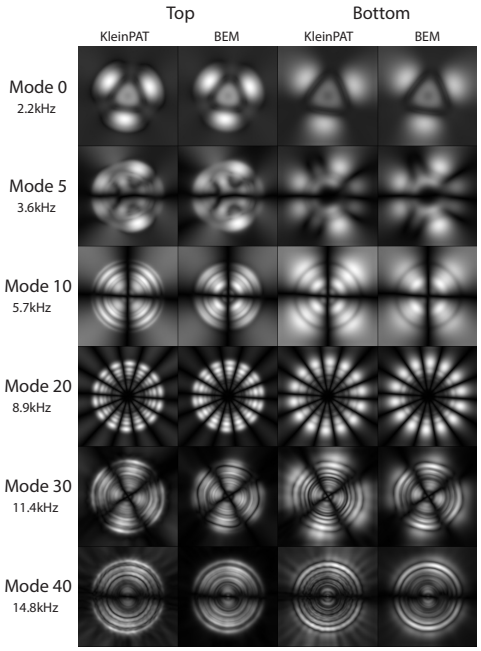


Fig. 11. **Transfer fields of the Plate** captured by our KleinPAT solver (left) and the 617x slower BEM solver (right) exhibit very similar structure, shown here as FFAT maps near the top/bottom of the plate model.

8 CONCLUSION

The KleinPAT solver enables rapid generation of acoustic transfer models suitable for real-time sound rendering. Our GPU-accelerated FDTD transfer solver can leverage mode conflation and transfer de-conflation that outperforms traditional BEM-based transfer pipelines.

Limitations and Future Work. Our work has many limitations and opportunities for future improvement. First, our approach has many parameter values (e.g., n, ϵ, δt, compression settings, etc.), and better performance and/or accuracy may be possible with further optimization. Despite good performance on several objects, the volumetric FDTD method is not a panacea for acoustics as it still suffers from poor scaling at high frequencies and/or for physically large

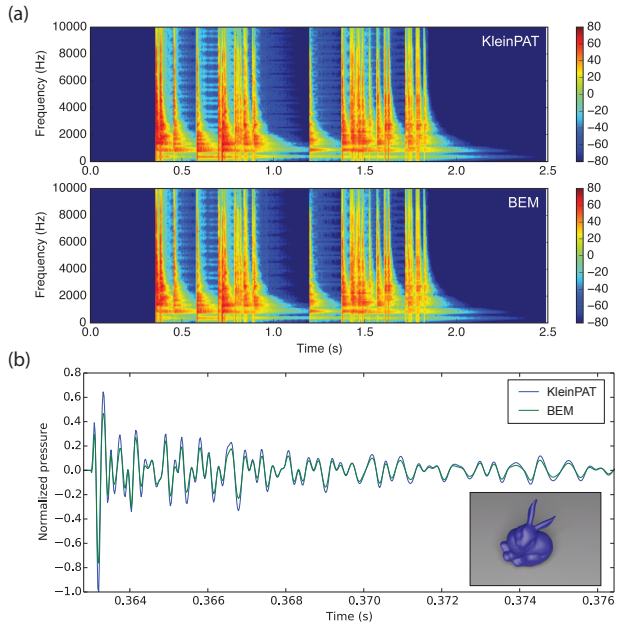


Fig. 12. **Sound rendering comparison for the dropped bunny**. We show that sounds generated by the two algorithms have very similar spectral and temporal characteristics. Both sounds were generated using FFAT Maps, which have the same angular resolution and discretization. Maps for the KleinPAT solution were generated by the procedures described in §6. BEM solutions were first evaluated on the same expansion boxes; the corresponding FFAT Maps for BEM were then computed and stored for fast runtime queries. (a) shows the spectrogram for the full sequence; (b) shows the sound waveforms following an impact event.

Table 3. **Far-field Transfer Models**: We compare the multipole-based transfer models with our FFAT cube map approach using the bunny object. The ground truth solution is given by BEM model directly evaluated using a Kirchhoff integral at far-field points–centroids of 20k triangles uniformly sampled on a sphere 10 times bigger than the bunny. The storage space is calculated based on the minimal number of bytes necessary to evaluate a far-field point. The storage size for the FFAT map is computed using 8-bit JPEG compression with "65" quality settings. All numbers are averaged over the number of modes.

	Accuracy	Performance			Storage
Model	err	precomp	eval	total	memory
BEM	0.0dB	N/A	29.1s	29.1s	5.0MB
Multipole	5.9dB	1.99s	0.79s	2.78s	2.45KB
FFAT Map	2.8dB	0.15s	0.01s	0.16s	1.86KB

sound models that necessitate high-resolution voxel grids and increased time-stepping costs. Chords with very many modes m have an increased $O(nm^2)$ QR factorization cost, which might require huge chords to be split in extreme cases. Conflated FDTD solves can timestep the K modes of a chord all together, however evaluating the Neumann acceleration BC still requires a weighted sum of K surface mode displacements at N surface points, which has $O(KN)$ cost and can be expensive for small timesteps. Chords can have a wide range of frequencies, which means that the highest frequency mode determines the finest spatial grid resolution required, and also time-step size.

We observe that amplitude-based FFAT cube maps (with a single map per mode) can achieve good far-field accuracy, speed, and memory performance trade-offs compared to traditional single-point multipole expansions. Future work should investigate optimal perceptually based tolerances for compressing FFAT cube maps. Least-squares estimation of FFAT cube maps require a sufficiently large domain to avoid over-fitting near-field waves, especially for higher-order FFAT map expansions.

Table 4. **Material parameters**: Material parameters used in the study. ρ: density; E: Young's Modulus; ν: Poisson ratio; α and β are the Rayleigh damping parameters. All parameters are in SI units.

Materials	ρ	E	ν	α	β
Ceramic	2700	7.2E10	0.19	6	1E-7
Glass	2600	6.2E10	0.20	1	1E-7
Wood	750	1.1E10	0.25	60	2E-6
Plastic	1070	1.4E9	0.35	30	1E-6
Iron	8000	2.1E11	0.28	5	1E-7
Polycarbonate	1190	2.4E9	0.37	0.5	4E-7
Steel	7850	2.0E11	0.29	5	3E-8

A MODIFIED SORTED BALANCE ALGORITHM FOR LOAD BALANCING

In this section, we list the modified sorted balance algorithm used for vectorize the wavesolver in Algorithm 1. For our simulations, we use $\gamma = 0$ unless otherwise specified.

Algorithm 1: Modified SortedBalance

Function groupPartitions()
Input : Number of partition K. Sorted cardinality for each partition t_i, where $t_1 \geq t_2 \geq \ldots \geq t_K$. Elastic parameter $\gamma \geq 0$.
Output: Job assignments A for m machines
Initialize one machine M_1 and add the first job to it
Set $T_1 = t_1$, $T_{\max} = t_1$, $m = 1$, and $A(1) = \{1\}$
for $i = 2$ **to** K **do**
  Let M_j be the machine that achieves the minimum $\min_k T_k$
  if $T_j + t_i > (1 + \gamma)T_{max}$ **then**
    Add one new machine
    $m = m + 1$
    Assign job i to this new machine M_m
    $A(m) = \{i\}$
    $T_m = t_i$
  else
    $A(j) = A(j) \cup \{i\}$
    $T_j = T_j + t_i$
  $T_{\max} = \max(T)$
return A

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. Toyota Research Institute ("TRI") provided funds to assist the authors

with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

REFERENCES

T. Akenine-Möller. 2002. Fast 3D Triangle-box Overlap Testing. *Journal of Graphics Tools* 6, 1 (2002).

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).

M. Bebendorf. 2000. Approximation of boundary element matrices. *Numerical Mathematics* 86, 4 (01 Oct 2000), 565–589. https://doi.org/10.1007/PL00005410

Mario Bebendorf. 2008. Hierarchical Matrices - A Means to Efficiently Solve Elliptic Boundary Value Problems. In *Lecture Notes in Computational Science and Engineering*.

M. Bebendorf and R. Kriemann. 2005. Fast parallel solution of boundary integral equations and related problems. *Computing and Visualization in Science* 8, 3 (01 Dec 2005), 121–135. https://doi.org/10.1007/s00791-005-0001-x

S. Bilbao. 2009. *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. John Wiley and Sons.

S. Bilbao. 2011. Time domain simulation and sound synthesis for the snare drum. *Journal of the Acoustical Society of America* 131, 1 (2011).

S. Bilbao. 2013. Modeling of Complex Geometries and Boundary Conditions in Finite Difference/Finite Volume Time Domain Room Acoustics Simulation. *IEEE Transactions on Audio, Speech, and Language Processing* 21, 7 (July 2013), 1524–1533. https://doi.org/10.1109/TASL.2013.2256897

S. Bilbao and C. J. Webb. 2013. Physical modeling of timpani drums in 3D on GPGPUs. *Journal of the Audio Engineering Society* 61, 10 (2013), 737–748.

N. Bonneel, G. Drettakis, N. Tsingos, I. Viaud-Delmon, and D. James. 2008. Fast Modal Sounds with Scalable Frequency-Domain Synthesis. *ACM Transactions on Graphics* 27, 3 (Aug. 2008), 24:1–24:9.

D. Brunner, M. Junge, P. Rapp, M. Bebendorf, and L. Gaul. 2010. Comparison of the Fast Multipole Method with Hierarchical Matrices for the Helmholtz-BEM. *Computer Modeling in Engineering & Sciences* 58 (03 2010).

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* (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).

H. Cheng, W. Y. Crutchfield, Z. Gimbutas, L. F. Greengard, J F. Ethridge, J. Huang, V. Rokhlin, N. Yarvin, and J. Zhao. 2006. A wideband fast multipole method for the Helmholtz equation in three dimensions. *J. Comput. Phys.* 216, 1 (2006), 300–325.

D. Ciscowski, R. and C. A Brebbia. 1991. *Boundary Element methods in acoustics*. Computational Mechanics Publications and Elsevier Applied Science, Southampton. UK.

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

W. W. Gaver. 1993. Synthesizing auditory icons. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. ACM, 228–235.

Gene H. Golub and Charles F. Van Loan. 2013. *Matrix Computations* (4th ed.). The Johns Hopkins University Press.

N. A. Gumerov and R. Duraiswami. 2005. *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. Elsevier Science.

R. W. Hamming. 1998. *Digital filters*. Courier Corporation.

Doug James, Changxi Zheng, Timothy Langlois, and Ravish Mehra. 2016. Physically Based Sound for Computer Animation and Virtual Environments. In *ACM SIGGRAPH 2016 Courses*. ACM, 22.

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.

S. M. Kay. 1988. *Modern Spectral Estimation: Theory and Application*. Prentice Hall.

D. Komatitsch, G. Erlebacher, D. Göddeke, and D. Michéa. 2010. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. *Journal of computational physics* 229, 20 (2010), 7692–7714.

T. R. Langlois, S. S. An, K. K. Jin, and D. L. James. 2014. Eigenmode Compression for Modal Sound Models. *ACM Transactions on Graphics (TOG)* 33, 4, Article 40 (July 2014), 9 pages. https://doi.org/10.1145/2601097.2601177

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

D. Li, Y. Fei, and C. Zheng. 2015. Interactive Acoustic Transfer Approximation for Modal Sound. *ACM Transactions on Graphics (TOG)* 35, 1 (2015). https://doi.org/10.1145/2820612

Q.-H. Liu and J. Tao. 1997. The perfectly matched layer for acoustic waves in absorptive media. *The Journal of the Acoustical Society of America* 102, 4 (1997), 2072–2082.

Y. J. Liu. 2009. *Fast Multipole Boundary Element Method: Theory and Applications in Engineering*. Cambridge University Press, Cambridge.

P. J. Looges and S. Olariu. 1992. Optimal greedy algorithms for indifference graphs. In *Proceedings IEEE Southeastcon '92*. 144–149 vol.1. https://doi.org/10.1109/SECON.1992.202324

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.

A. Meshram, R. Mehra, H. Yang, E. Dunn, J.-M. Frahm, and D. Manochak. 2014. P-hrtf: Efficient personalized hrtf computation for high-fidelity spatial sound. *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on* (2014).

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

J. D. Morrison and J.-M. Adrien. 1993. Mosaic: A framework for modal synthesis. *Computer Music Journal* 17, 1 (1993), 45–56.

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*. ACM Press, 175–181.

D. K. Pai, K. van den Doel, D. L. James, J. Lang, J. E. Lloyd, J. L. Richmond, and S. H. Yau. 2001. Scanning physical interaction behavior of 3D objects. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 87–96.

S. Prepelita, M. Geronazzo, F. Avanzini, and L. Savioja. 2016. Influence of voxelization on finite difference time domain simulations of head-related transfer functions. *The Journal of the Acoustical Society of America* 139, 5 (2016), 2489–2504. https://doi.org/10.1121/1.4947546

Z. Ren, H. Yeh, and M. C. Lin. 2013. Example-guided physically based modal sound synthesis. *ACM Transactions on Graphics (TOG)* 32, 1 (2013), 1.

Y. Saad and M. H. Schultz. 1986. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Statist. Comput.* 7, 3 (July 1986), 856–869. https://doi.org/10.1137/0907058

X. Serra and J. Smith. 1990. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal* 14, 4 (1990), 12–24.

A. A. Shabana. 2012. *Theory of Vibration: An Introduction*. Springer Science & Business Media.

A. A. Shabana. 2013. *Dynamics of multibody systems*. Cambridge university press.

W. Śmigaj, T. Betcke, S. Arridge, J. Phillips, and M. Schweiger. 2015. Solving Boundary Integral Problems with BEM++. *ACM Trans. Math. Software* 41, 2, Article 6 (Feb. 2015), 40 pages. https://doi.org/10.1145/2590830

A. Taflove and S. C. Hagness. 2005. *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Artech House.

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

K. van den Doel, P. G. Kry, and D. K. Pai. 2001. FoleyAutomatic: Physically-based Sound Effects for Interactive Simulation and Animation. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (ACM Transactions on Graphics (Proceedings of SIGGRAPH 2001))*. ACM, New York, NY, USA, 537–544. https://doi.org/10.1145/383259.383322

K. van den Doel and D. K. Pai. 1996. Synthesis of shape dependent sounds with physical modeling. *International Conference on Auditory Display* 28 (1996).

O. von Estorff. 2000. Boundary Elements in Acoustics: Advances and Applications. WIT Press, Southhampton, UK.

J.-H. Wang, A. Qu, T. R. Langlois, and D. L. James. 2018. Toward Wave-based Sound Synthesis for Computer Animation. *ACM Transactions on Graphics (TOG)* 37, 4, Article 109 (July 2018), 16 pages. https://doi.org/10.1145/3197517.3201318

Wikipedia. 2019. All-interval twelve-tone row. https://en.wikipedia.org/wiki/All-interval_twelve-tone_row

T. W Wu. 2000. Boundary Element Acoustics: Fundamentals and Computer Codes. WIT Press, Southhampton, UK.

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

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

C. Zheng and D. L. James. 2011. Toward High-Quality Modal Contact Sound. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)* 30, 4 (Aug. 2011).

E. Zwicker and H. Fastl. 2013. *Psychoacoustics: Facts and models*. Vol. 22. Springer Science & Business Media.