PRIORITIZED COMPUTATION FOR NUMERICAL SOUND PROPAGATION

John Drake

Center for Human Modeling and Simulation,
University of Pennsylvania
Philadelphia, PA, USA
drake@seas.upenn.edu

Maxim Likhachev

The Robotics Institute,
Carnegie Mellon University
Pittsburgh, PA, USA
maxim@cs.cmu.edu

Alla Safonova

Center for Human Modeling and Simulation,
University of Pennsylvania
Philadelphia, PA, USA
alla@seas.upenn.edu

ABSTRACT

The finite difference time domain (FDTD) method is commonly used as a numerically accurate way of propagating sound. However, it requires extensive computation. We present a simple method for accelerating FDTD. Specifically, we modify the FDTD update loop to prioritize computation where it is needed most in order to faithfully propagate waves through the simulated space. We estimate for each potential cell update its importance to the simulation output and only update the N most important cells, where N is dependent on the time available for computation. In this paper, we explain the algorithm and discuss how it can bring enhanced accuracy and dynamism to real-time audio propagation.

1. INTRODUCTION

Faithful propagation of sound through arbitrary environments is a computationally complex problem. Audio propagation solutions must be re-evaluated as the source position(s), listener position(s), and environment geometry change over time. If the recomputation can be done very quickly, the method might be useful in real-time applications like virtual environment audio simulations.

In this paper we present a method to accelerate the finite difference time domain numerical sound propagation method so that it might be used in real-time applications under broader configurations. We prioritize computation where it is needed most to most accurately propagate a wave, eliminating computation where it would have little effect on the output.

2. PREVIOUS WORK

Sound propagation can broadly be split into two groups: geometric methods and numerical methods. Geometric methods often take advantage of analytic solutions to wave equation problems directly in terms of the geometry of the environment and assume that sound waves travel in straight lines. Numerical methods discretize and solve wave equation problems with numerical analysis.

Geometric methods include such techniques as image methods [1], ray tracing [2], beam tracing [3], and acoustic energy transfer methods [4]. In the early image method presented in [1], virtual image sources are created from the true sound source to represent the acoustical contribution of sounds reflected from geometry in the environment. Similar to graphics research on geometric tracing techniques, ray [2] and beam [3] tracing methods have been developed for audio propagation. Ray tracing samples an environment with a multitude of rays reflecting from surfaces. Errors are introduced in ray tracing methods when samples miss important features in the environment [5]. Beam tracing methods improve on this by sampling continuous areas of the environment with each cast beam and splitting each cast beam where the environment is discontinuous, such as at the edge of a beam-intersecting wall. However, it is hard to accurately handle effects like wave diffraction in arbitrary environments using these methods.

Numerical methods include finite element [6] and finite difference [7] methods. The finite difference time domain method (FDTD) is an especially popular numerical method in acoustics, though originally developed for electricity and magnetism [7]. In the FDTD method, a finite simulation lattice is overlaid on the environment to discretize it in space, and the output is computed at discrete time steps over many iterations. The method naturally allows the modeling of arbitrary environment configurations and captures wave phenomena like diffraction. However, it suffers from high memory and computation time demands, especially as a simulation's time or space discretization is refined. We focus on FDTD in this paper and look at it in greater detail in Section 3.

Several hybrid methods have been developed, merging geometric and numerical methods [8], [9]. Hybrid methods are good for real-time applications because different wave properties can be efficiently represented by different methods. A recent example is the "Wave-Ray Coupling" presented by Yeh et al. 2013 [10], where a geometric technique handles long-distance wave propagation in a large environment and a numerical method captures important wave diffraction effects (which the geometric technique cannot handle) close to the listener position. Acceleration of computationally expensive numerical methods like FDTD is necessary for the numerical components of hybrid methods to function well in real-time applications. For example, the adaptive rectangular decomposition method [11] is used instead of plain FDTD in [10]. Any additional acceleration to the numerical component of a

This work was supported by NSF Grant IIS-1018486.

hybrid system can critically provide better real-time performance over a broader range of configurations, e.g. a larger numerical simulation zone capturing a more complete simulation of diffracted waves in the environment. This paper offers one such acceleration technique.

3. BACKGROUND

Sound is a wave phenomena dependent on the wave equation. Analytic solutions to the equation exist for simple configurations, but no complete analytic solutions exist for complex environments.

The FDTD method provides a way to solve the equation in complex environments in a discrete way. The environment is discretized in space into a regular grid. The sizes of the grid cells in each dimension do not have to match but for simplicity they will be equal here, represented by h. $p^n(i,j,k)$ represents the pressure at location i,j,k at time n. Throughout, we use the speed of sound $c \approx 340$m/s.

Following is a brief derivation of an FDTD update equation, based on the description in [11].

$$\frac{\partial^2 p}{\partial t^2} - c^2 \nabla^2 p = F(x,t) \tag{1}$$

$F(x,t)$ is a forcing term representing sound inputs. It is zero without inputs.

∇^2 represents the 3D Laplacian operator, $\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$, the divergence of the gradient at a point in the pressure field.

The Laplacian operator can be discretized for a grid representation in many ways[12]. We use the L_2 discretization for speed and simplicity:

$$\nabla^2 p^n(i,j,k) \approx$$

$$lap(i,j,k) = \frac{1}{h^2}\begin{pmatrix} -6p^n(i,j,k) + \\ p^n(i-1,j,k) + p^n(i+1,j,k) + \\ p^n(i,j-1,k) + p^n(i,j+1,k) + \\ p^n(i,j,k-1) + p^n(i,j,k+1) \end{pmatrix} \tag{2}$$

Or more succinctly with K as a Discrete Laplacian Matrix and P a long vector of all pressure values in the grid:

$$\nabla^2 p \approx \frac{1}{h^2} KP \tag{3}$$

yielding

$$\frac{\partial^2 P}{\partial t^2} - \frac{c^2}{h^2} KP = F(t) \tag{4}$$

Discretizing in time with time step $\triangle t$ and using the leapfrog integrator yields the following update equation.

$$P^{n+1} = 2P^n - P^{n-1} + \left(\frac{c\triangle t}{h}\right)^2 KP^n + \triangle t^2 F^n(t) \tag{5}$$

Time step size $\triangle t$ depends, for the sake of numerical stability in the simulation, on the grid resolution according to Courant-Friedrichs-Lewy condition $\triangle t < \frac{h}{c\sqrt{3}}$

To model the interface of air and environment surfaces, any of many absorbing boundary conditions (ABCs) can be used. Please see [13], based on the original Perfectly Matched Layer (PML) work [14] for a very helpful derivation of the PML ABC for the single field parameter "scalar" FDTD context presented above. The works [15],[16] present the formulation of a simple surface ABC which also may be used in this context.

4. PRIORITIZED FDTD

Full FDTD simulation demands the evaluation of a large number of computations. There are many time steps needed, and in each one, potentially every grid cell representing the simulation space needs to be updated. Each FDTD time step depends on the previous time step, but within each time step, every cell update computation is independent. Also, if one cell is updated, the effect of that update is only relevant to its neighboring cells whose discretized Laplacian estimations in the subsequent time step include a term reading the value of that previously-updated cell. Areas of uniform pressure remain static until disturbed by impinging waves and also become static again after those waves pass by. These properties together allow us to accelerate FDTD by prioritizing computation where it is needed most and omitting it elsewhere.

We incorporate these properties into one tunable system by the introduction of a cell update importance function and prioritized selection of which updates to execute. We concentrate on accelerating FDTD impulse response simulation of low frequency diffracting waves. Higher frequency reflecting components can be simulated in real time with other methods, together forming a hybrid system as in [10]. Our method of acceleration is orthogonal to other approaches like parallelization, so we believe it can be applied on top of other methods for further improved results.

4.1. FDTD Setup

We initialize our FDTD simulation grid with a grid cell size, h appropriate for low frequency waves (e.g. simulation frequency= 1KHz, $h \leq \frac{c}{2KHz}$ guided by the Nyquist-Shannon sampling theorem). These low frequency waves have a greater tendency to diffract in a significant way in human-scale environments than do higher frequency waves.

We focus on the task of recording an impulse response, rather than continuously propagating an arbitrary source wave. Simulated impulse responses can be efficiently convolved with arbitrary source waves after simulation to auralize output. The impulse response context allows us accelerate computation more than we could if the source emitted an arbitrary wave. This is true because with an impulse, more of the pressure field is likely to be static and unimportant (in front of or behind the impulse wave) at any arbitrary time step than if the field were inundated with a continuous series of waves.

To record an impulse response, we use an input pulse defined by the following Gaussian function. Its parameters were chosen to make the pulse peak near the origin and to be short but not so short that it causes major ringing oscillations when played into the 1KHz grid simulation.

$$\begin{aligned} pulseCenterTime &= 0.0025 \\ spread &= 0.001 \\ f(t) &= e^{-\frac{(t-pulseCenterTime)^2}{2*spread^2}} \end{aligned} \tag{6}$$

Note that any impulse response recorded from $t = 0$ using this impulse must be shifted in time by $-pulseCenterTime$.

PML ABC zones are placed around the boundaries of the simulation space. It is suggested in [17] to make the PML thickness at least 65-70% of the longest wavelength of interest. Because we primarily focused on the recording of impulse responses and to aid computation times we use $n_a = 10$, which corresponds to a "longest wavelength of interest" of $0.65c/h/10 = 130$Hz.

The profile of our input pulse approximately corresponds to a sinusoidal wave section with a frequency around 200Hz, so the PML thickness is reasonable.

4.2. FDTD Computation

Described below is our method to accelerate FDTD computation in the context described in Section 4.1.

4.2.1. Importance Function

An importance function, notated as $importance(i, j, k)$, estimates the importance of a cell update during simulation. Before the first FDTD update pass, the $importance$ function is initialized for every cell to zero. At the moment of the start of the input pulse, the sound source location and its immediate neighbors are given artificially elevated $importance$ values to seed their evaluation when they are first important.

In the last moments of a simulation, many regions of the environment are so far from the listener position(s) that no wave leaving those regions and traveling at the speed of sound could reach the listener(s) before the end of the simulation. We include this in all importance functions by forcing importance to zero when a cell update can be omitted. Let L represent the listener position in grid units. Let dur be the duration of the simulation. $Omit$ determines when a cell update would be made to a cell too far from the receiver position to possibly have any effect on the output of the simulation.

$$Omit(i, j, k, t) = \frac{h \cdot |[i, j, k] - L|}{c} > (dur - t) \qquad (7)$$

Idealy, an importance function would look into the future and determine how much of an effect an update will have on the eventual output of the simulation. Since this is not possible to do in any less time than it would take to run the simulation to that future time or even in less time than it takes to do a single update, we approximate the importance function by estimating the effect an update could have on the immediate region surrounding it in the following time step. In FDTD simulation, the discretized Laplacian approximation (lap) is the only term in the update equation (Equation 5, inside K) which interacts with neighboring cells, so our importance functions incorporate the same values which lap uses. Importance functions we used take these forms:

$importance_1$ (m_1) is something like the Laplacian approximation, but the absolute value of each term is used and all coefficients are one. It is always non-negative.

$$m_1(i, j, k) = \begin{pmatrix} |p^n(i, j, k)| + \\ |p^n(i-1, j, k)| + |p^n(i+1, j, k)| + \\ |p^n(i, j-1, k)| + |p^n(i, j+1, k)| + \\ |p^n(i, j, k-1)| + |p^n(i, j, k+1)| \end{pmatrix}$$

$$importance_1(i, j, k) = \begin{Bmatrix} 0 & \text{if } Omit(i, j, k, t) \\ m_1(i, j, k) & \text{otherwise} \end{Bmatrix} \qquad (8)$$

$importance_2$ (m_2) is something like a gradient magnitude. m_2 yields the largest magnitude of a difference between any two nearby pressure values ($Nearby(i, j, k)$ denotes the set of nearby pressure values and includes the value of the cell itself). It is always non-negative.

$$Nearby(i, j, k) = \begin{Bmatrix} p^n(i, j, k)), \\ p^n(i-1, j, k)), p^n(i+1, j, k)), \\ p^n(i, j-1, k)), p^n(i, j+1, k)), \\ p^n(i, j, k-1)), p^n(i, j, k+1)) \end{Bmatrix}$$

$$m_2(i, j, k) = \max_{\substack{p_1 \in Nearby(i,j,k) \\ p_2 \in Nearby(i,j,k)}} (p_1 - p_2)$$

$$importance_2(i, j, k) = \begin{Bmatrix} 0 & \text{if } Omit(i, j, k, t) \\ m_2(i, j, k) & \text{otherwise} \end{Bmatrix} \qquad (9)$$

4.2.2. Most Important Cell Retrieval

In every FDTD update pass, a limited number of cells with the highest $importance$ values are recomputed. To efficently ascertain which cells have the highest $importance$, we keep a list of update candidate cells which we call $candidates$. An average case $O(n)$ time partial sorting algorithm is used to partially sort $candidates$ once per time step according to the importance function. It ensures that the first N cells in the list have greater importance than all others. Provided their importances are non-zero, these cells are updated in the usual FDTD manner to finish evaluation of a time step.

The number of cells to recompute, N, can be estimated according to the approximate volume of the pulse wavefront, defined here by the inner and outer radii, r_i and r_o, of an impulse wave in an open environment. t is the simulation time.

$$\begin{aligned} r_o &= c \cdot t \\ r_i &= c \cdot (t - 2 \cdot pulseCenterTime) \\ ApproxWaveVol(t) &\approx \frac{4}{3}\pi {r_o}^3 - \frac{4}{3}\pi {r_i}^3 \end{aligned} \qquad (10)$$

$$N(t) = ApproxWaveVol(t) \div h^3 \qquad (11)$$

4.2.3. Refreshing the Importance Function Efficiently

To avoid unnecessarily evaluating the $importance$ function for every cell at every time step, we keep a set of cell references which we call $refreshSet$. Once per time step, the $importance$ for every cell stored in $refreshSet$ is evaluated and all other cells are ignored. After this, $refreshSet$ is cleared. In a single time step of FDTD, the $importance$ values of *only* updated cells and their immediate neighbors can change. So, whenever a cell is updated in one time step, the cell and its neighbors are added to $refreshSet$ for $importance$ re-evaluation before updates at the next time step.

4.2.4. Pseudocode

Some specific implementation details have been simplified, such as code to avoid duplicate additions to $refreshSet$ and the special case for PML zones. See Algorithm 1 and UpdateSimState.

5. DISCUSSION

5.1. Analysis

Figure 1 shows how N changes during a simulation. The unshaded portion shows the amount of computation avoided by our method.

begin
  initialize;
  for $t = 0$...*dur at increments of* $\triangle t$ **do**
    UpdateSimState;
    Force pressure at source cells;
    Record IR output at listener cells;
    Prepare for next time step;

Algorithm 1: FDTD Outer Loop

begin
  initialize;
  $candidates$ =List of all sim cells;
  for $element \in refreshSet$ **do**
    Refresh $importance$ of $element$.
  $refreshSet = \emptyset$;
  N = compute as in Eq. 11;
  Partially sort $candidates$ as in 4.2.2;
  for $i = 1...N$ **do**
    updateCell($candidates[i]$);
    $refreshSet+ = candidates[i]$;
    $refreshSet+$ = All neighbors of $candidates[i]$;

Procedure UpdateSimState

The dashed line at the top marks the total number of cells in the FDTD grid. At the beginning of a simulation, computation is limited by N, and at the end of a simulation, it is limited by the $Omit$ function. The superimposed curves show the numbers of updates done on actual runs of our algorithm in our test environments. The configuration of the environment affects how many updates are done by affecting the number of zero-importance cells at different times. Zero-importance cells are not updated even if N is larger than the number of nonzero-importance cells). Figure 2 is similar to Figure 1, but the simulation duration is five times longer.

The first profile, Figure 1, demonstrates the context where our method is most useful: impulse response simulations of limited duration, such as the numerical simulation component in a hybrid system like Yeh et al.'s Wave-ray Coupling [10]. However, even in less ideal situations like Figure 2, our method always saves some computation at the beginning and end of the simulation and provides a pricipled approach to restricting computation in the middle of the simulation too, by limiting N.

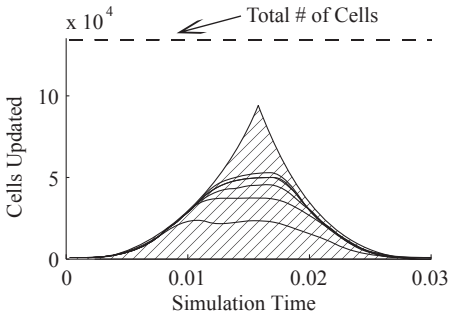

Figure 1: *With simulation parameters from our trials in Section 6, the shaded region is computed, the rest omitted.*

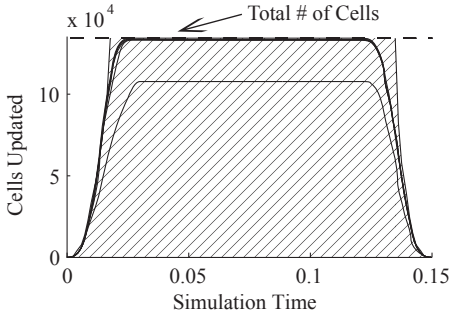

Figure 2: *Longer duration simulation worst-case behavior.*

If the simulation computes output for a continuous input instead of an impulse response or if multiple source positions are present, our method can still be used. $ApproxWaveVol$ would have to be changed, which would change the shape of the early portions of Figures 1 and 2 (for lesser performance), but the end portions of the simulation would remain the same, since the same distance-based cell omission can be done. Conversely, adding additional listener positions affects the later portions of simulation while not affecting the beginning.

5.2. N Approximation

In an environment with many absorbing surfaces (or areas open to the simulation boundary PML regions), a tighter bound on the wavefront volume can be made by observing that if there were no obstacles in the environment, an expanding spherical wave would eventually begin to leave the simulation grid. The portions of the wave which have left the space can be subtracted from the volume as calculated in 4.2.2.

Our presented approximation for N, estimating the wave volume in an empty environment, is not always quite enough to capture important updates at the front of a wave because the importance function is only an estimation of true update importance. In the "worst case" of a completely open environment while the propagated wave forms a spherical shell, we find it helps to inflate the very tight wavefront volume figure by up to 20% to ensure that important cells on the leading edge of the wavefront are not missed in simulation. In our experiment trials, we did not have to inflate the N value, because absorbers in our environments and the edges of the simulation space reduced the actual simulated wave volume below the estimated wave volume before significant deterioration took place. If, as discussed in the previous paragraph, a tighter volume bound were used, some inflation might indeed be needed in all cases.

6. RESULTS

We computed average performance statistics on an Intel i7-860 machine. The code was compiled and linked from C++ source with the MSVC 2008 compiler. We ran the simulation thirty six times for our method and thirty six times for plain FDTD. The trials were divided evenly between six environments which together stress our algorithm and represent plausible hybrid simulation scenarios: four open artificial environments with various wall configurations between source and listener, one completely open

	Time Prioritized	Time FDTD	Speedup in Time
Open	1.26s	4.29s	3.41
Env. A	1.24s	4.28s	3.44
Env. B	1.04s	4.31s	4.13
Env. C	1.17s	4.27s	3.66
Env. D	1.27s	4.29s	3.38
Building	0.71s	3.59s	5.07
Column Averages	1.11s	4.17s	3.85

Table 1: *Performance results in terms of time.*

	Updates Prioritized	Updates FDTD	Speedup in Cell Updates
Open	15.0%	100%	6.66
Env. A	14.5%	99.1%	6.82
Env. B	12.3%	99.1%	8.05
Env. C	13.7%	99.4%	7.23
Env. D	15.5%	99.5%	6.86
Building	8.2%	80.8%	9.86
Column Averages	13.1%	96.3%	7.58

Table 2: *Performance results in terms of cell updates. "Updates FDTD" is not always 100% because updates are not made within environment obstacles.*

environment with no walls, and one generated from interior geometry of a real building. The environment dimensions were all fixed at 7m $\times$ 7m $\times$ 2.5m and the simulated duration (0.03s, long enough to receive all impulse wave diffractions in our trials) was also equal across all trials. These results are shown in Tables 1, 2, and 3. "Time" columns show average computation times, "Updates" columns show average percentages of updates performed out of the maximum possible, and "Speedup" columns show the relative performance of our method over plain FDTD. In all trials, FDTD updates were not made for cells within solid objects. The real building environment had the largest number of occluded cells (around 20%).

As seen in Table 1, our approach improves average simulation speeds by a factor of 3.85. When the real building environment trial is considered alone, the improvement was over a factor of 5. Memory usage with our method was around 70% greater than plain FDTD, to store the $candidates$ list, $refreshSet$, and other data to do things like avoid duplicate neighbor additions to $refreshSet$ efficiently.

Figure 3 shows response waveform comparisons of our method to full FDTD simulation for three environments, at three different inflation factors for N and one deflation factor for N. Each plot has a response from our method overlaid with the full FDTD response. The first two environments (Env. C and Env. D respectively in Table 1) are artificial and mostly open, so they exhibit a mild case of the problem explained in Section 5.2, where N is close to the actual wave volume and the importance function does not perfectly indicate which cells must be updated. Mild N inflation helps those results converge. The third environment is the real building environment and has many reflecting surfaces which the other tested environments do not have. Relative computation times as N inflation factors change are given in the caption of Figure 3.

	0.8 N Factor	1.0 N Factor	1.2 N Factor	2.0 N Factor
Open	42967.8	3014.0	271.3	69.7
Env. A	6148.4	517.6	138.7	49.2
Env. B	21.9	3.7	2.9	2.8
Env. C	76.5	5.9	1.9	1.3
Env. D	4343.0	241.8	25.3	12.7
Building	91.4	39.4	46.0	36.6

Table 3: *Sum of squared error between waveform outputs of the full FDTD and the prioritized method. Compare with Figure 3*

7. CONCLUSIONS & FUTURE WORK

Our prioritized computation method accelerates FDTD wave propagation. It is especially helpful in the context of a hybrid simulation where a method like FDTD captures the effect of an impulse wave diffracting in an environment. Our acceleration allows such an impulse response simulation to be repeated more rapidly, with better discretization, or larger environment size, to improve real time results.

Our implementation was not parallelized, but it also does not prevent the use of parallelization. In fact, preliminary results show that running our method on a single CPU thread is faster than a four-way parallel FDTD implementation we tested on four CPU threads, at least under the trial configurations tested in Section 6. It is future work for us to parallelize prioritized FDTD computation.

We kept the complete set of simulated cells in the $candidates$ list, but this list could instead be grown, starting with just the sound source cell, by appending the contents of $refreshSet$ at each time step. Old irrelevant cells could likewise be removed over time. If $candidates$ were made shorter like this or if special consideration were given to the high temporal coherence of $candidates$, the running time of the partial sorting algorithm could be reduced.

The partial sorting of $candidates$ creates an overhead over plain FDTD in the middle of long simulations (like ones with reflections which must be simulated), when N approaches the full volume of the space. In these situations, because we do not change the pressure field representation, computation can easily be switched between our method and plain FDTD to avoid the overhead. Computation savings would still be found at the beginning and end of the simulation time. Alternatively, an upper bound can be placed on the size of N to guarantee that performance is always better than ordinary FDTD while still maintaining simulation quality in a pricipled way. While potentially deleterious in the "worst case," an upper bound like this could ordinarily be used because the wave volume in a simulation typically does not approach the total environment volume.

Finally, future work also includes testing the effectiveness of prioritized computation in numerical techniques other than FDTD.

8. REFERENCES

[1] B. M. Gibbs and D. K. Jones, "A simple image method for calculating the distribution of sound pressure levels within an enclosure," *Acustica*, vol. 26, pp. 24–32, 1972.

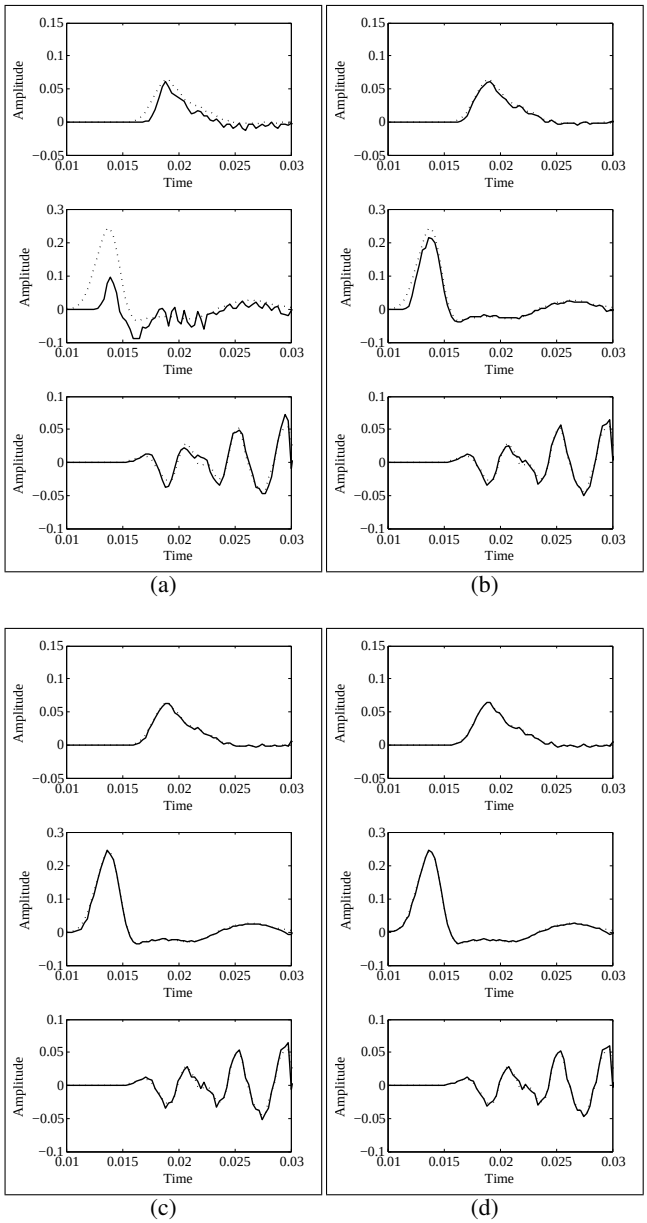

Figure 3: *The broken lines are the full FDTD ground truth output and the solid lines are the prioritized computation output. The three waveforms correspond to Env. C, Env. D, and Building, respectively, in Tables 1, 2 and 3.*
(a): 20% N deflation (10% shorter computation time than (b), but quality suffers in some environments)
(b): No N inflation (quality suffers a little in worst-case open environments – see Section5.2)
(c): 20% N inflation (5% longer computation time than (b))
(d): 100% N inflation (12% longer computation time than (b))

[2] A. Krokstad, S. Strøm, and S. Sørsdal, "Calculating the acoustical room response by the use of a ray tracing technique," *J. Sound Vib.*, vol. 8, no. 1, pp. 118–125, 1968.

[3] Norm Dadoun, David G. Kirkpatrick, and John P. Walsh, "The geometry of beam tracing," *Proc. of the first annual symposium on Computational geometry*, pp. 55–61, 1985.

[4] Nicolas Tsingos, *Simulating High Quality Virtual Sound Fields for Interactive Graphics Applications*, Ph.D. thesis, Universite J. Fourier, Grenoble I, December 1998.

[5] Hilmar Lehnert, "Systematic errors of the ray-tracing algorithm," *Applied Acoustics*, vol. 38, pp. 207–221, 1993.

[6] Emmanuel Granier, Mendel Kleiner, Bengt-Inge Dalenbäck, and Peter Svensson, "Experimental auralization of car audio installations," *J. Audio Eng. Soc*, vol. 44, no. 10, pp. 835–849, 1996.

[7] Kane S. Yee, "Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. AP-14, no. 3, pp. 301–307, May 1966.

[8] Ying Wang, Safieddin Safavi-Naeini, and Sujeet K. Chaudhuri, "A hybrid technique based on combining ray tracing and fdtd methods for site-specific modeling of indoor radio wave propagation," *IEEE Transactions on Antennas and Propagation*, vol. 48, pp. 743–754, 2000.

[9] S. Hampel, S. Langer, and A. P. Cisilino, "Coupling boundary elements to a raytracing procedure," *International Journal for Numerical Methods in Engineering*, vol. 73, pp. 427–445, 2008.

[10] Hengchin Yeh, Ravish Mehra, Zhimin Ren, Lakulish Antani, Ming C. Lin, and Dinesh Manocha, "Wave-ray coupling for interactive sound propagation in large complex scenes," *Proc. of ACM SIGGRAPH Asia (TOG)*, 2013.

[11] Nikunj Raghuvanshi, Rahul Narain, and Ming C. Lin, "Efficient and accurate sound propagation using adaptive rectangular decomposition," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 5, 2009.

[12] Steve Schaffer, "Higher order multi-grid methods," *Mathematics of Computation*, vol. 43, no. 167, 1984.

[13] D. Zhou, W. P. Huang, C. L. Xu, D. G. Fang, and B. Chen, "The perfectly matched layer boundary condition for scalar finite-difference time-domain method," *IEEE Photonics Technology Letters*, vol. 13, no. 5, pp. 454–456, May 2001.

[14] Jean-Pierre Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 114, pp. 185–200, 1994.

[15] Tapio Lokki, Alex Southern, and Lauri Savioja, "Studies on seat dip effect with 3d fdtd modeling," *Proc. of Forum Acusticum*, 2011.

[16] Konrad Kowalczyk and Maarten van Walstijn, "Room acoustics simulation using 3-d compact explicit fdtd schemes," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 1, pp. 34–46, January 2011.

[17] Yotka S. Rickard, Natalia K. Geogieva, and Wei-Ping Huang, "Application and optimization of pml abc for the 3-d wave equation in the time domain," *IEEE Transactions on Antennas and Propagation*, vol. 51, no. 2, pp. 286–295, February 2003.