Rendering Environmental Voice Reverberation for Large-scale Distributed Virtual Worlds

Micah Taylor*
University of North Carolina

Nicolas Tsingos†
Dolby Laboratories

Dinesh Manocha‡
University of North Carolina

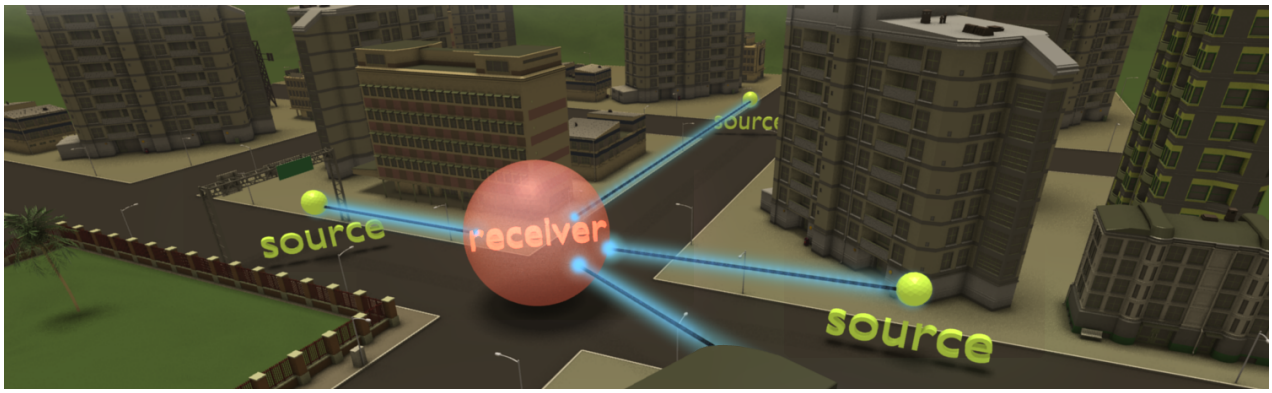


Figure 1: *Our algorithm can efficiently generate environmental acoustic effects in large virtual worlds. The city scene above is 600 meter × 980 meter in size. Our algorithm can precompute the acoustics for this space in less than 15 minutes, with the resulting data consuming only 173MB of storage. At runtime, over 690 sound streams can be rendered in realtime on a single core.*

Abstract

We present an algorithm that can render environmental audio effects for a large number of concurrent voice users immersed in a large distributed virtual world. Our approach uses an offline step, efficiently computing acoustic similarity measures based on average path length, reflection direction and diffusion throughout the environment. The similarity measures are used to adaptively decompose the scene into acoustic regions. Sound propagation simulation is performed on the acoustic regions; the resulting acoustic response data can be used efficiently at runtime to enable reverberation effects. We show that adaptive sampling based on our similarity metric correlates well with errors in perceptual acoustical metrics, contrary to naive subsampling. We demonstrate real-time, plausible sound rendering of large number of voice streams in virtual environments encompassing areas of tens of square kilometers at a fraction of the authoring and memory cost of previous acoustical precomputation approaches.

1 Introduction

For networked virtual environments, such as social communities or massively multiplayer on-line (MMO) games, meaningful interaction through voice conversation with other participants is a valuable feature [Williams et al. 2007; Wadley et al. 2007; Sallnäs 2005]. First adopted through side-clients that enabled telephone-quality, walkie-talkie style communication, voice services are becoming more integrated and are now connecting hundreds of millions of users on PCs, game consoles and cell phones. For instance, group voice chat is integral to gaming services such as Microsoft Xbox LIVE, Sony Playstation Network, and Valve Steam, and is also directly integrated into such games as Blizzard's World of Warcraft, CCP Games' EVE Online, Electronic Arts's Need for Speed World, and Linden Lab's Second Life.

There has been much work on voice communication over the internet, typically called VoIP (Voice over Internet Protocol) [Goode 2002]. Some of the key issues include latency, voice coding efficiency, network error resilience, and endpoint voice cleaning and processing [Markopoulou et al. 2002; Benesty et al. 2000]. While most work in scalable VoIP focuses on the infrastructure system, there has also been significant research on improving the immersive effects of the voice communication, typically through spatialized rendering [West et al. 1992; Hollier et al. 1997]. Studies have shown [Halloran 2009; Gibbs et al. 2006] that, while voice communication helps users coordinate in virtual environments, a lack of environmental effects can make it difficult for users identify sound sources.

Modeling the effects of sound propagation, such as occlusion and echoes, can help convey scenes where participants communicate from different rooms or areas. For example, the direction of the direct sound path and early sound reflection can help a user spatialize the sound source position (i.e. localization), while the time it takes for the late reverberation echoes to decay conveys the scale of the environment and the materials present (i.e. immersion). There has been considerable progress in interactive sound propagation and many techniques based on geometric and numeric methods have been proposed [Funkhouser et al. 1998; Raghuvanshi et al. 2010; Vorländer 1989; Mehra et al. 2013; James et al. 2006; Taylor et al. 2012; Yeh et al. 2013; Savioja 2010; Schissler et al. 2014; Tsingos et al. 2004; Tsingos et al. 2007; Raghuvanshi and Snyder 2014]. They have been used for interactive sound propagation and rendering for indoor and outdoor environments on high-end multi-core workstations and also integrated with game engines, such as Half-Life 2 and Unity. However, these techniques may not work well in large-scale VoIP systems. VoIP algorithms have unique aspects:

- Restricted data: Each client is constantly recording voice data that is known only to that individual client.
- Communication cost: The voice must be transferred between clients leading to bandwidth and latency costs. In client-server arrangements, thousands of clients need to be handled by a single server.

*e-mail: taylormt@cs.unc.edu
†e-mail: nicolas.tsingos@dolby.com
‡email: dm@cs.unc.edu

- Rendering cost: VoIP is commonly used in large MMOs, on the order of tens to hundreds of square kilometers in scale. This can significantly increase the computational cost of acoustic simulation.

- Thin clients: Cell phones and tablets are becoming increasingly popular and magnify the transfer and rendering costs in VoIP. Current cell phones often allocate only 12kbs for voice streams [ETSI 2014].

Since the voice mixing is generally performed on a remote server, network delivery cost restricts the amount of data that can be transferred. Moreover, a typical multi-core server platform must handle thousands of remote clients simultaneously [Dol 2012], strongly limiting the processing capabilities. Peer-to-peer VoIP systems [Berndt et al. 2011; Liang et al. 2013] remove the need for a server system to process all client audio, but push the communication and simulation cost to the clients. In these systems, implementing spatialized audio is possible [Zimmermann and Liang 2008; Jiang et al. 2014], but such effects are limited to binaural or distances attenuation effects.

Even VoIP systems that include effects beyond simple direct-path distance attenuation [Radenkovic et al. 2002; Boustead and Safaei 2004; Safaei 2005] only support direct line-of-sight occlusion modeling and simplified diffraction effects, which result in unrealistic proximity cues. This is because the storage or compute cost of propagation simulations (discussed below) is quite high. This load must be either borne by the clients (handling 3-4 simulations in real-time) or the server (handling hundreds of simulations in real-time).

For MMO games, where localizing teammates and enemies is of primary importance, rendering inappropriate distance cues could lead to a tactical disadvantage. Advances have been made in peer-to-peer infrastructures where bandwidth cost is reduced, but audio computation remains quite costly for clients to evaluate in real-time. In this paper, we focus on the immersive rendering of environmental reverberation effects on voice streams for such large-scale distributed virtual worlds with multiple sources and receivers.

Sound reverberation and environmental effects

In current video games, reverberation is either directly pre-rendered into the sound effects or implemented at run-time using dynamic artificial reverberation filters [Jot 1999]. Parameters of reverberation decay can be directly manipulated by the sound designer to achieve a desired effect without requiring any geometrical modeling. While simplifying the authoring process, traditional artificial reverberators suffer from a number of issues. They impose a "single room" model and constrain the shape of the decay profile (e.g., exponential). Because of their limited use of scene geometric representation, these methods also fail to convincingly model coupled or outdoor spaces. In [Bailey and Brumitt 2010], distance histograms extracted from a cube map rendered at a position of interest are used to select the parameters for such an artificial reverberator. However, distances to the camera origin alone fail to capture local variations of the surfaces, which have a strong influence on the scattering properties [Tsingos et al. 2007].

Client-server solutions have been proposed to dynamically compute sound propagation paths using the actual scene representation [Funkhouser et al. 1999]. But even the most recent geometrical acoustic (GA) approaches, which can model dynamic sound reflections and diffraction interactively [Taylor et al. 2012; Schissler et al. 2014; Schissler and Manocha 2014], use high-end workstations with multiple CPU cores or GPUs to compute propagation paths and are not designed for client-server environments, i.e. handling thousands of remote clients simultaneously, very large number of sources, etc. Other geometric acoustic methods [Funkhouser et al. 1998; Chandak et al. 2009] take advantage of the static nature of the scene to precompute a visibility tree. However, these methods assume that either the source or the receiver position is fixed.

One practical approach to simulating the acoustics of virtual environments is to pre-compute the acoustical impulse response (IR) at several locations throughout the environment in an off-line process; the results can then be efficiently re-used to process the audio signals at run-time by querying the response database and reconstructing a reasonable solution for a given source and microphone pair [Pope et al. 1999; Tsingos 2009; Siltanen et al. 2007; Antani et al. 2012; Raghuvanshi et al. 2010; Mehra et al. 2013; Yeh et al. 2013; Raghuvanshi and Snyder 2014] . The main benefit of the off-line computation is that both early and high-order sound scattering (reflection/diffraction) can be simulated, providing improved proximity cues and distance perception. Moreover, the numerical methods can also accurately compute the low-frequency sound propagation effects. However, most of these techniques have been designed for small indoor or outdoor acoustic spaces with only a few sources or objects and may not scale to large virtual worlds with a high number of sources and receivers. As a result, these methods are not well-suited for large-scale VoIP systems.

Overview

Given our goal of supporting environmental acoustic effects in very large virtual environments, we wish to reduce the time and storage cost required to support such scenes. One way to accomplish this goal is to reduce the number of acoustic responses that must be precomputed and stored, while still maintaining the accuracy of acoustic effects. We can sample the scene at a finite number of locations and compute the acoustic response based on those sample positions. The precomputation time and storage cost can be lowered by reducing the number of samples needed. This can be achieved by combining nearby sample points that have similar acoustic impulse responses. This process requires sampling the acoustic field at each point. However, combining sample points in this manner only reduces the final *storage cost*, not the *time cost* required to precompute all the acoustic responses.

To address this challenge, we introduce three main contributions:

1. Geometric acoustic similarity measure: We introduce a geometric measure based on the properties that influence the acoustic field. The measure can be computed quickly using the local neighborhood of a given point location in the environment. This enables us to perform scene decomposition and sampling in $O(p)$ time for p sample points (section 2).

2. Scene subdivision: We use the similarity measure to sample the virtual environment and then segment the scene into regions. These regions enclose portions of the scene that are similar based on the properties of our similarity measure, and our algorithm only needs to sample the full acoustic responses in each region. This results in a reduction of both the precomputation time and the storage overhead, and thereby enables us to handle very large scenes which span kilometers in virtual space (section 3).

3. Efficient response storage: We present an efficient approach that scales in both time and space complexity to accommodate large acoustic scenes. Our storage algorithm compresses redundant data while supporting fast inserts and constant average time retrieval. This allows for efficient storage of the tens of billions of acoustic responses needed for kilometer-sized scenes (section 4) .

Figure 2 offers an overview of the proposed approach in the context of a large-scale VoIP system as used in multi-player online games. We have evaluated our algorithm in large indoor and outdoor scenes

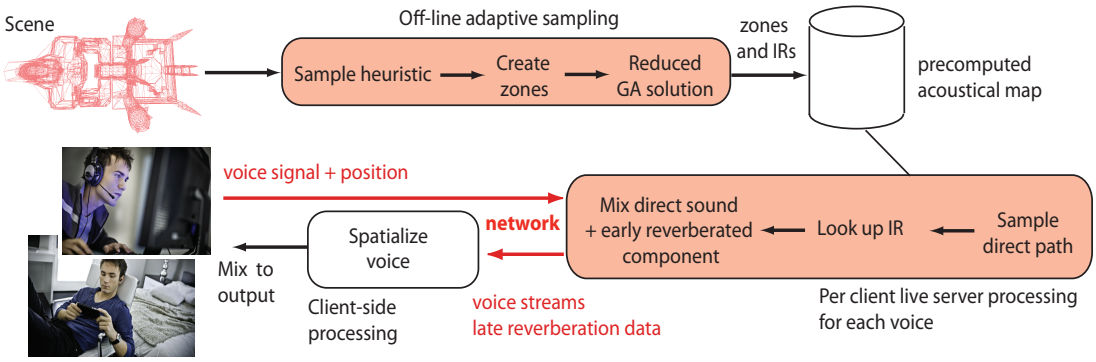


Figure 2: Example integration into a VoIP system: *Our efficient pre-computation and reverberation algorithm enables real-time rendering of environmental audio effects in large-scale environments with many connected clients. In this paper we focus on the components highlighted in red boxes.*

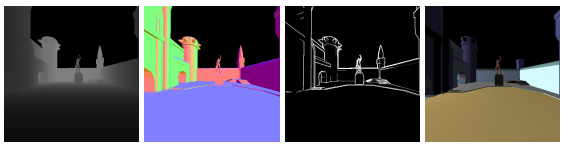


Figure 3: Sample signature for FPS scene in Figure 10(b): *The components of the similarity measure: (a) distance with black being near and white far, (b) direction with vectors shown as RGB components, (c) discontinuities in depth and direction, and (d) materials with three frequency bands shown in RGB components.*

that correspond to game maps and virtual worlds, with specular reflections, diffuse reflections and edge diffraction. We show that adaptive sampling based on our geometrical similarity metric correlates well with errors in perceptual acoustic criteria, contrary to naive subsampling (section 5). Our approach can be used to automatically create environmental reverberation maps up to 5 times faster than previous solutions on indoor scenes and up to 80 times faster on large outdoor scenes. In addition, storage costs are reduced by a similar factor, and our optimized runtime processing allows interactive rendering of reverberation effects for scenes comprising hundreds of connected clients (section 6) and we highlight the benefits over prior approaches (section 7).

2 Geometric Acoustic Similarity Measure

In this section, we introduce our metric and present a GPU-based algorithm for fast computation. In Section 3, we use this metric to decompose the scene into regions.

Our goal is to sample the acoustic field and form a database of acoustic responses, using as few samples as possible, while still generating a reasonable set of responses. The final output of any acoustic simulation is derived from the source/receiver configuration, the location of obstacles and the material properties of the scene. This can be expressed using the room acoustic rendering equation (RARE) [Siltanen et al. 2007], which describes the acoustic radiance from a point in the scene as

$$\ell(x', \Omega) = \ell_0(x', \Omega) + \int_G R(x, x', \Omega)\ell(x, \Gamma)dx, \qquad (1)$$

where the outgoing acoustic radiance ℓ for point x' and angle Ω is a sum of the emissive radiance $\ell_0(x', \Omega)$ and the integral of all reflected radiance from point x. The reflected radiance is scaled by the function R

$$R(x, x', \Omega) = V(x, x')\rho(x', \Theta, \Omega)g(x, x'), \qquad (2)$$

which accounts for the visibility V between points, the reflectance function ρ, and the distance and orientation of the points g. The function $g()$ can be expressed as:

$$g(x, x') = \left\lfloor \mathbf{n}(x) \cdot \frac{x' - x}{|x' - x|} \right\rfloor \left\lfloor \mathbf{n}(x') \cdot \frac{x - x'}{|x - x'|} \right\rfloor \frac{S_{|x-x'|}}{|x - x'|^2}, \qquad (3)$$

where $\mathbf{n}$ represents the surface normal at a point and S represents the effects of propagation over a distance, i.e. delay and attenuation.

Our goal is to quickly sample the scene and find locations where the acoustic field has a high gradient to guide the sampling. One possibility is to use the RARE to sample the scene, but this can be rather expensive for a large environment. Instead, we compute a first-order approximation that can be evaluated quickly using standard rasterization algorithms such as *cube mapping* [Greene 1986]. Using axis-aligned cube maps, we sample geometric and spatial data from the scene. From the cube map data, we extract several values related to the physical properties that influence the propagation of sound waves in the scene: surface distance, surface orientation, surface discontinuities, and surface absorption. Figure 3 shows the visual data that is sampled.This spatial data corresponds to some of the physical properties that influence first-order reflections: path length, reflection direction, surface reflectance properties, and diffusion.

2.1 Distance

Acoustic waves that arrive at the receiver as reflections must first bounce off an object in the scene. The path the sound travels results in delay, creating temporal effects that humans use to determine environment properties. In media that absorbs sound, the distance the sound travels modifies the signal further, as some energy is lost to absorption. The S term in RARE (Equation 3) accounts for effects that are related to the propagation distance. However, in order to measure this effect exactly, one must perform an acoustic simulation that computes both early reflections and late reverberations between the source and the receiver. It can be expensive to perform this computation for each source-receiver pair. Instead, we use a simpler approximation, and only sample first-order geometric properties to compute the reflection paths.

We note, using the law of cosines, that the length of the reflection path, p, can be computed by forming a triangle between the source, the reflector, and the receiver (Fig. 4). This is given as

$$p = f + \sqrt{f^2 + v^2 - 2fv\cos\phi},$$

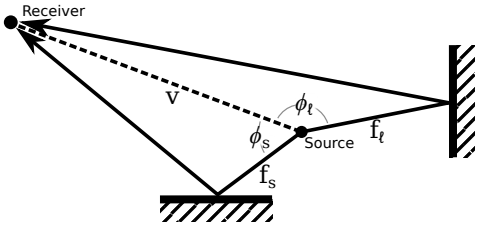


Figure 4: Surface distance: *The first-order propagation distance p is directly related to nearby reflectors. v is the direct path to the receiver. f_s represents the shortest first-order reflection path as ϕ_s goes to 0. f_ℓ represents the longest first-order reflection path as ϕ_ℓ goes to π. Our algorithm measures the f terms of p.*

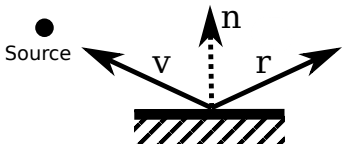


Figure 5: Surface orientation: *Reflection direction r varies as a property of the incoming vector v and the surface normal orientation n.*

where f represents the path to the reflector, v is the direct path to the receiver position, and ϕ is the angle between these vectors. As ϕ approaches zero, the path approaches the shortest possible reflection path between the source and the receiver; when ϕ is π, the reflection path is the longest possible path between the source and the receiver. Figure 4 illustrates the shortest path as f_s and the longest path as f_ℓ. The value of ϕ, and consequently p, are only known at runtime, so we sample the f component and use it to approximate the distance in the final acoustic response.

2.2 Surface orientation

The direction of earliest incoming sound paths to a receiver is highly indicative of the direction of the sound source, and is a key element in sound localization. The $g()$ function in Equation 3 describes how the direction of an object's surface normal directly influences the direction of any reflected sound paths off the surface. As in our handling of arrival delay, we avoid computing the full acoustic simulation by sampling only the first-order geometric properties.

Reflection direction $\mathbf{r}$ can be determined based on the view direction $\mathbf{v}$ and the surface normal direction $\mathbf{n}$ (Fig. 5). We note that $\mathbf{v}$ is fixed by the cube map sample location and the reflection direction is a function of $\mathbf{n}$, given as:

$$\mathbf{r} = 2(\mathbf{v} \cdot \mathbf{n})\mathbf{n} - \mathbf{v}.$$

Since the surface normal directly influences the direction of first-order reflection, computing the difference in first-order normals between two scene locations provides an approximation of the difference in reflection direction, and the actual difference in early onset direction.

2.3 Surface discontinuities

The amount of diffraction and diffuse scattering from surfaces can significantly influence the final acoustic response at the receiver's location. The RARE does not directly account for edge diffraction effects, but the reflectance function $\rho()$ (Equation 2) can model such scattering. Scattering occurs at the boundaries, which correspond to edges in 3D scenes. Simulating scattering effects using geometric acoustics is time-consuming and challenging, as compared to reflection effects [Calamia and Svensson 2007; Taylor et al. 2012], and may require an accurate wave-based solver [Raghuvanshi et al. 2010]. Instead, we use an approximation based on first-order geometric properties that influences these effects, including depth and orientation discontinuity.

We measure the surface gradient with respect to surface normal and depth to estimate diffraction and diffuse properties. When sampling our cube maps, we record the depth values and surface normals. Next, we use a series of 2D operators over the cube face to find discontinuities in depth gradient and normal gradient. These discontinuities represent potential regions where diffraction and diffusion effects are significant. First, a 2D gradient is computed from the depth information:

$$\nabla_x = [1, -1]; \quad \nabla_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

This results in a two-component image that represents depth changes in x and y directions. We look for discontinuities in this image to estimate where scattering is likely to originate from. An edge detection kernel is applied to the components of the gradient:

$$k = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The maximum value, u, of the two components is retained and clamped to the range $[0, 1]$, resulting in a black and white map used to detect depth and normal discontinuities. Since the depth and normal discontinuities influence the scattering of sound in the scene, we use these discontinuities as a component in our measure.

2.4 Surface material

The material properties govern the absorption and scattering effects. The RARE accounts for materials in the bidirectional reflectance distribution function (BRDF) term, $\rho(\Omega_i, \Omega_e; x')$ (Equation 2). A commonly used BRDF segments the reflectance into pure absorption, diffuse reflection, and specular reflection components using an absorption factor α and a diffusion factor δ. These factors are defined over several frequency bands to define absorption, diffuse, and specular coefficients:

$$\underbrace{\alpha}_{absorption} + \underbrace{\delta(1-\alpha)}_{diffusion} + \underbrace{(1-\delta)(1-\alpha)}_{specular} = 1.$$

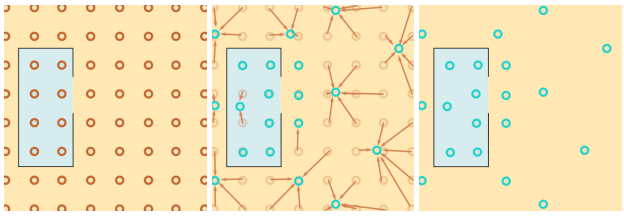
We directly sample the material of the first-order surface and use that to approximate the resulting effect. Note that α is a vector over several frequency bands.

2.5 Overall similarity measure

The components of the measure are stored on each face of the cube map and constitute a first-order response measure related to the surfaces near the sampling point. Storing all cube-face images for each sample point requires large amounts of storage. In order to reduce the memory overhead, we perform an integration step to compute the mean of the geometric properties on each cube face. Given a geometric value q (representing one of the four data values) across s cube map samples, we compute the mean $q_{avg} = \sum_{i=0}^{s} \frac{q_i}{s}$.

After the integration step, the cube map data for each axis is reduced to a four tuple of mean surface distance f_{avg}, mean surface orientation $\mathbf{r}_{avg}$, mean surface absorption α_{avg}, and mean number of pixels that lie on possible scattering edges u_{avg}. The final similarity measure $\mathbf{G}$ is a six row matrix composed of this four tuple for each of the six axis-aligned sample directions. Each direction is referred to as a superscript (shown as x below):

$$\mathbf{G} = [f^x_{avg}, \mathbf{r}^x_{avg}, \alpha^x_{avg}, u^x_{avg}].$$



(a) (b) (c)

Figure 6: Adaptive sampling: *(a) Regular grid sampling creates a very high number of samples; (b) we remove redundant samples; (c) adaptive sampling of the scene with fewer samples.*

3 Scene decomposition and sampling

In this section, we present our scene-decomposition algorithm, which greatly reduces the time and storage cost of the acoustic precomputation while incurring only small error in response accuracy. The similarity measure, $\mathbf{G}$, described above is simple and fast to compute. As a result, our algorithm first evaluates the similarity measure on a dense grid of sample points over the scene. Once the surface properties near each sample point have been measured, sample points with similar properties are merged, which reduces the total sample point count in the scene. We first discuss the method to compute the difference between two samples, followed by the techniques used to segment the scene and compute acoustic responses.

3.1 Similarity comparison

In the previous sections, we described our method for extracting geometric properties that correspond to expected reflection-path length, expected reflection direction, and expected diffuse energy. We assume that if two sample points have similar values for these parameters, it is likely that the acoustic field measured around the sample positions will be similar. The difference $\mathbf{S}$ between sample points $\mathbf{a}$ and $\mathbf{b}$ can be computed by taking the component-wise difference of the quantities described above:

$$\mathbf{S} = |\mathbf{G}_a - \mathbf{G}_b|.$$

When calculating the difference in the mean surface distance, f_{avg}, the percent difference distances is used. This results in a difference value in the range $[0, 1]$, which can account for changes in both short and long distances. Similarly, for direction, the dot product is taken and scaled to the range $[0, 1]$. Since the absorption vector, α, is already bounded $[0, 1]$, we use the maximum scalar absorption difference for any sub band. The discontinuity measure is also naturally bounded $[0, 1]$. The result is that $\mathbf{S}$ is a 6×4 matrix with the components being scalars in the range $[0, 1]$.

$\mathbf{S}$ will be used to determine if two locations in the scene are similar. In addition to the $\mathbf{S}$ parameters, line of sight is also used to restrict similarity; we assume that there is a line of sight (LOS) between the sample points considered similar. This forms the *basis* of our similarity metric: sample points are likely to have a similar acoustic response if the early response (i.e., cube map measured data) is similar and if the points are visible to each other (LOS restriction).

3.2 Sample merging

Once the similarity properties between all neighboring sample points have been measured, the scene-decomposition algorithm is used to compute adaptive sampling. We note that the similarity measure $\mathbf{S}$ is useful for evaluating whether two sample positions are similar, but not directly useful for eliminating sample positions. If a sample point is removed, the nearby acoustic field will be sampled sparsely, which may lead to more error at reconstruction when a receiver is placed near that sample point.

The sample points are merged in an adaptive quad-tree like manner. This is done by recursively subdividing the sample points into regions. The corners of regions are evaluated for similarity. Most quad trees divide the working set by four at each level. Our reduction algorithm also attempts to subdivide the four half regions (two along each dimension). This reduces the number of subdivision steps by a small amount.

The maximum norm is taken for each property in $\mathbf{S}$ across all dimensions (e.g. column vectors), and the result is used to form a weighted average error, s_p. This error is compared against the similarity threshold s_{thr}:

$$s_p = \frac{1}{4}(||\mathbf{S}_0||_\infty + ||\mathbf{S}_1||_\infty + ||\mathbf{S}_2||_\infty + ||\mathbf{S}_3||_\infty),$$

$$s_p < s_{thr}.$$

If the error criterion is satisfied, all points bounded by the corners may be combined into a single region. If the corners do not satisfy the metric, the region is further subdivided and the tests are repeated for all sub-regions. Each resulting region then contains similar geometric properties based to the degree specified in s_{thr}.

3.3 Acoustic regions

The output of the refinement stage is an irregular arrangement of merged samples (see Figure 6). The process of sample merging also gives us acoustic regions: if two sample points are merged, they were classified to be similar by the error criteria, and their original positions in the 2D grid are marked as belonging to the same region. As points are merged, the acoustic regions grow until the edges at the boundary of the region no longer satisfy the error criteria. At runtime, the acoustic region of the source point and the receiver point must be selected based on the spatial position of the source and the receiver.

Since the source and the receiver will not lie exactly on the sample points used during precomputation, the appropriate acoustic region is computed by checking the four nearest grid locations. The simplest case is when the four nearest grid locations belong to a single acoustic region. This region can be immediately identified. If not, LOS queries with $O(\log t)$ time cost are performed from the four points against the objects in the scene. The closest point with a clear LOS is selected as the closest grid node.

3.4 Simulation of sound propagation

Our method treats sound propagation simulation as a black box, and its only requirement is with respect to the input: the scene representation, along with the positions of source and receiver. From this input, it computes an impulse responses as the output. We use geometric acoustics (GA) simulation in order to compute responses on large scenes, since the complexity of wave-based simulators is a linear function of the scene volume. Our simulator traces specular, diffuse, and diffraction rays [Vorländer 1989; Taylor et al. 2012] and computes paths between a source-receiver pair; it then outputs an acoustic impulse response that represents environmental effects on sound waves traveling from the source to the receiver. This impulse response can be convolved with any input sound signal to render an output signal with the appropriate effects.

We identify the center of each region obtained from the decomposition process; this point becomes a sample point for the propagation simulation. Each sample is modeled as a source from which rays are traced into the scene. Every sample is also considered a receiver, and propagation paths are recorded at each sample. In other words, each sample position acts as a source and a receiver. The acoustic field is measured by simulating an acoustic emitter at each sample position and measuring the resulting IR at all sample position. Considering a emitter at one of the sample positions, sound waves

leave the emitter and are recorded at all other sample positions, resulting in an IR at each receiving sample position. For each sample position, a full acoustic simulation is run, resulting in the acoustic propagation paths between all sample positions. In our implementation we use a ray-based GA simulator tracing uniform random rays from the source and perform specular, diffuse, and diffracting transport. Each ray carries a portion of the emitted sound energy into the scene, modeling sound propagation. We model the receivers as spheres at the sample position and use ray tracing methods to compute the acoustic field in the scene. Since modern ray acceleration structures can achieve $O(logt)$ performance for a scene composed of t primitives (often triangles), model size in terms of primitives is not a restricting factor. However, in addition to the scene intersection, each ray requires intersection with p receivers at the sample points. Even with region segmentation, there are still a high number of source-receiver pairs and each sample point also acts as a source, this results in $O(p^2)$ ray tests using naive methods. We include receiver locations in a ray-tracing acceleration structure to reduce the receiver intersection cost to $O(p \log p)$. We take additional steps to reduce the size of p, as described in Section 4.1.

4 Storage and reconstruction

This section describes our storage algorithm. We have designed a method to efficiently compress, store, and retrieve audio-response data. Since our method is designed for large scenes, our storage database must scale in time and space. An acoustic response is a signal over time or frequency. If spatialization effects are desired, direction data must also accompany the signal. For our purposes, three portions of the response data are recorded: decay profile, incident direction, and diffusion. However, the methods described below can encode any type of data.

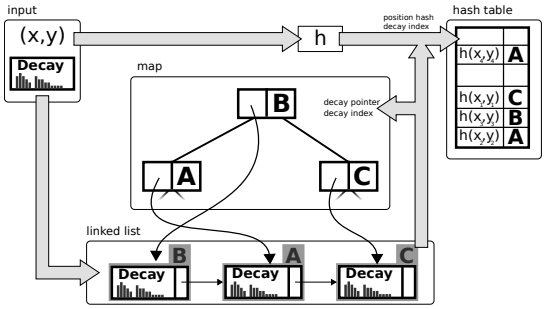
4.1 Response representation

The paths computed in the trace step form the acoustic response between the two samples and represent the environment's filter of the acoustic signal between the two locations. Such filters can be represented compactly by sampling the energy decay profile through time [Merimaa and Pullki 2004; Tsingos 2009]. The decay profiles are built by integrating the energy in the impulse response over small time-steps and a number of frequency sub-bands. In this way, both temporal and frequency resolution can be controlled by the user.

The response is stored in three parts: pressure values, a direction value in 2D, and a diffuse coefficient to indicate how strong the directionality is. Similarly to other methods [Tsingos 2009], we store pressure values for several spectral sub-bands quantized in the time domain. An energy-weighted average direction of incidence is stored at time-step resolution; all pressure frequency bands share the same direction. Similarly, a directional-to-diffuse energy value is also computed [Merimaa and Pullki 2004].

In the theoretical worst case, p^2 responses could be generated (p sources to p receivers), this does not happen in practice. Since there exists a maximum sound energy level for each source (e.g. human voices), there must be a bound on the number of responses generated. Moreover, the energy emitted is diminished by air absorption as it travels. For a given grid of n sample points, only k samples will have audible response to the human voice. The k factor is based on the distance between the sampling points, dissipation by the air attenuation model, the maximum pressure level that is simulated, and the minimum pressure level that can be sensed.

This bounds the number of responses that can be be inserted in the database, i.e. kn. This also bounds the time cost of the simulation, since the simulation can be confined to a region large enough to enclose the k sample points. As the scene size increases, the fact that k is constant allows the time and space required by the simulation to be bounded.



(a)

Figure 7: Data insertion: *The decay data A, B, C (example response data) is appended to a linked list and the length of the list is the data index. The index is stored with a pointer to the decay data as a pair in a map. The index is paired with a position hash and stored in a hash table. The insertion can be performed in average $O(\log n)$ time. After all the data is stored, the linked list is converted to a linear array, for a total time complexity of $O(n)$ and average storage cost of $O(n)$.*

4.2 Storage data structures

Since there is response data for each source-receiver pair, it is logical to store the data in a linear vector with an index table to query by source-receiver. This approach allows $O(1)$ insert and retrieval time. However, since the number of source-receiver pairs can be very large, the $O(p^2)$ scaling of the table index dominates the storage cost. For example, a scene of 35K sample points would result in nearly 5GB of storage for the table itself. On very large scenes, the size of the index can exceed the storage needed for the acoustic data. It is thus desirable to reduce the size of the actual data stored and the size of the index needed to query the data.

4.2.1 Storing acoustic data

Based on acoustic reciprocity, we assume that acoustic responses from swapping source position and receiver position should be similar. We capitalize on this factor to reduce the amount of acoustic data stored. To perform this step, we detect if an insertion will result in duplicate data. We can impose strict weak ordering on all the acoustic data, so a map offers fast insertion and query time. However, the ordering of map nodes is not fixed until all the insertions are completed. Since the index must be built at the same time, we use a linked list to store the actual acoustic data, and the map references this data. The ordering of this list is fixed through the insertion process, and the index is built against the list. This allows insertions to take place in $O(1)$ time for the list and $O(\log p)$ time for the map. Moreover, space complexity for both structures is $O(p)$. Refer to the supplementary materials for algorithm details.

In our algorithm, we store decay profile, direction, and diffusion separately, and store only unique responses to take advantage of reciprocity. This allows very efficient duplicate storage, and, most importantly, sparse storage of the empty response. Since we are simulating very large scenes, the most common output from the simulator is an empty acoustic response.

4.2.2 Efficient data indexing

For the acoustic data, both insertions and queries are performed during the simulation. At runtime, only the acoustic data is queried. Thus, when all insertions are complete, the map is discarded and the list is converted to a vector for efficient storage. The separate index is used to query acoustic data from this vector. Since the vector representation supports random access in $O(1)$ time, the cost of

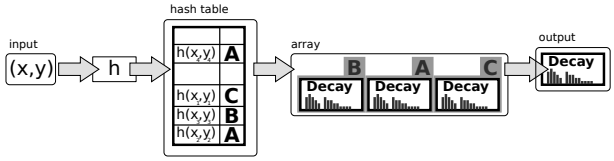


Figure 8: Data access: *For runtime access, the query position is hashed and the decay index is found in average $O(1)$ time. The data array is then queried for the final decay data.*

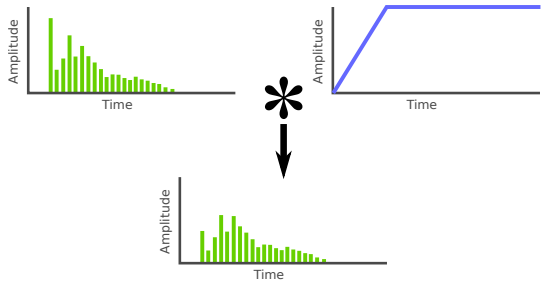


Figure 9: Early response attenuation: *The early response pressure is attenuated for source/receiver pairs in the same region.*

querying acoustic data is thus dominated by index lookup.

The data stored in the index is a key-value pairing of a hash with the index to the acoustic data in the previously discussed list. When the list is converted to a vector, the ordering is preserved. Combining the source-receiver indices results in a good hash function; it can be represented as a single integer that is guaranteed to not collide with any other hashed values. We also note that we expect many responses to be empty and thereby, represent such a quiet response (i.e. no sound). Based on these assumptions, we do not store keys that reference empty acoustic data. Instead, if a key is not found at query time, it is assumed that the data for the key is the empty response. This significantly reduces the size of the index.

This data is queried often at run time to lookup the acoustic data vector locations. Therefore, during we construct a hash map from the unordered vector to represent the final index for $O(1)$ query time and $O(p)$ storage cost on average. We also found that sorted vectors performed equally well, even with $O(\log p)$ query time due to the constant values associated with each data structure. The responses can be accessed in constant time using our data structure (see Figure 8). The source-receiver hash map can be queried in $O(1)$ time on average.

4.3 Response reconstruction and rendering

When sampling an acoustic region, the source and receiver positions for response sampling are centered in the region. However, at runtime, the source and the receiver may be in the same region separated by the full width of the the region. Since our algorithm has only computed a single response for the entire region, we approximate the distance attenuation by scaling the early portion of the decay response. For example, we define the early portion as the first 140ms of the response. We scale each pressure value in the early portion by $\frac{1}{d}$, where d is the distance that leads to a response during that time step. This attenuation factor is linearly reduced for each consecutive time step, so that no attenuation is applied to the final early time sample. This process allows the early field to be attenuated without altering the standing late reverberation field (see Fig. 9). The direct path and early reflection rendering is performed on the server using a block-based convolution in the Modified Cosine Transform Domain [Tsingos et al. 2011]. This is efficient as audio is often coded in that same domain. The direct path and decay data are interpolated over time as the user moves through the

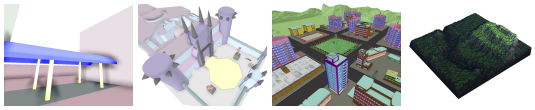


Figure 10: Example scenes: *Our algorithm can generate environmental acoustic effects in large virtual worlds and games. We show different benchmarks with their dimensions in meters: (a) simple outdoor scene ($33 \times 33 \times 10$); (b) first person shooter (FPS) game scene ($30 \times 60 \times 20$); (c) city scene ($600 \times 980 \times 33$); (d) canyon model ($4000 \times 4000 \times 100$). Our approach scales with the size of these models and can handle a large number of sources and receivers in multi-player environments at interactive rates.*

scene to avoid audible discontinuities in the output.

In many situations, the server can process the reverberation in realtime and operates in full mixing mode. This means all the processing is done on the server and stereo or multichannel results are generated and sent back to the client. Thanks to our representation of reverberation filters, it is also possible to send only a mono mixture and energy-weighted direction and direct/diffuse metadata to further reduce bandwidth and enable flexible spatialization on the client. No matter the rendering arrangement, all head-related filtering is performed at the client-side.

For environments with very long reverberations, which cannot be completely processed on the server, a further optimization would be to process only the direct + early part of the response (up to some number of blocks) on the server. An additional dry mono mixture and representative late reverb metadata would also be sent to the client so that it can provide local late reverb processing but this is not demonstrated in our video or results. This necessary metadata is the local late decay rate in the vicinity of the listener and an average late decay rate for all the sources.

5 Validation

In this section, we evaluate the accuracy of our approach using various perceptual metrics used in room-acoustic analysis. We used several example scenes, shown in Figure 10. These selected test scenes represent the likely use cases, and comprise both indoor and outdoor scenes, corresponding to game maps and virtual worlds. The details about these models and the underlying grid resolution are given in Table 1.

Scene	# Triangle	Size (m)	Grid spacing (m)	Sample count
Simple Outdoor	2k	33 x 33 x 10	4	81
FPS game	14k	30 x 60 x 20	4	128
Small city	2k	100 x 100 x 33	4	625
Large city	2k	247 x 168 x 33	4	4.9k
Canyon	540k	4k x 4k x 100	4	1,000k

Table 1: Example scenes: *Physical sizes for the indoor and outdoor scenes are given in meters (m). The sample count is for a regular grid at the given resolution.*

5.1 Error computation

We have measured various errors related to acoustic responses generated by our algorithms, including the similarity measure and the scene decomposition. We measured these errors in the reconstructed results based on certain properties of the impulse responses. In particular, we used some well-known *acoustic evaluation metrics* related to evaluating acoustic responses: onset time delay, initial onset direction, reverberation time in the form of RT60,

and signal definition in the form of D [Kuttruff 2007; ISO 3382 2009]. We performed GA simulation using our precomputation algorithm, computed impulse responses, and compared our results to a ground truth dataset. The ground-truth data is the full set of responses from the GA simulator with the source and the receiver positions at the same grid size, but without any sample reduction. We used a dense, uniformly-sampled grid to generate the ground-truth, as shown in Table 1. In both simulations, we traced 50K rays from each sample point, performing 50 orders of specular and diffuse reflections and 5 orders of edge diffraction. The relative error is calculated by dividing the absolute error by the maximum possible error for each metric. Below we give details related to error computation and evaluation and illustrate many error maps (as shown in Figure 2 and Figure 3). The low error in these maps validates our approach with respect to various acoustic evaluation metrics.

5.2 Similarity measure thresholds and validation

During the precomputation step, our method computes an acoustic similarity measure for positions in the environment. An error threshold is used to select similar regions during the scene decomposition. When merging the sample points and performing scene decomposition, the error between two sample points ranges in the interval $(0, 1]$. For the results presented in this paper, we set our error threshold s_{thr} to such that 75% of the original nodes are removed.

Since our approach reduces the number of acoustic responses based on our similarity measure, it is possible that the final acoustic map may not accurately represent the responses of the original acoustic environment. In order to show the accuracy of our decomposition algorithm, we compare the properties of a full ground-truth simulation to properties generated from our method in two scenes with the initial grid size set to $4m \times 4m$.

By adjusting the error threshold in the region-segmentation step, higher accuracy in acoustic properties can be achieved. However, this increases the number of acoustic regions that need to be stored. We expect simulations with fewer regions to have greatly reduced computational overhead, but to also have a small increase in errors in the impulse response. Figures 11 highlights the relative variation in error and computation cost as the region merge threshold is adjusted. The errors in this figure are computed by taking the average of the error for every possible position of the receiver, similar to the error maps used in Figure 2.

5.3 Acoustic property error calculation

In order to compare our approach to a ground-truth simulation, we consider several acoustic evaluation metrics: Onset delay (Onset), Onset direction, reverberation (RT60), and Definition (D). More details about these metrics can be found in [Kuttruff 2007; ISO 3382 2009] and in the supplementary materials. Due to our adaptive decomposition, some positions in our precomputed solution may be associated with empty responses, while the corresponding solution in the ground-truth data may have a non-zero response (and vice versa). In this case, the error at those positions is undefined and we use a worst case error value at these locations for our error metrics.

5.4 Error analysis for reduction algorithms

We investigated other possible reduction algorithms: naive subsampling, greedy flood fill reduction, and greedy sorted merge, and compared them with our adaptive schemes. Subsampling is placing the sample points in a regular grid at lower resolution sampling rates. This is the most natural way to reduce sampling in a scene. Flood fill is performed by sorting edges according to the minimum error, then greedily merging the lowest error nodes to form a region. Any nodes connected to the region are merged, provided the resulting error is low enough. In greedy sort merge, the edges are sorted according to the minimum error. Edges are merged in a greedy fashion until all possible edges are merged. We investigated all these methods and observed that adaptive reduction is the most effective (see Figure 12).

6 Implementation and Performance

We describe our implementation and highlight the performance of our algorithm on game scenes and virtual worlds. We have implemented our precompute system on a PC with an NVIDIA GPU with 480 threads and 1.5 GB video memory. The precompute system CPU is an Intel Xeon with 48 threads and 256 GB main memory. The runtime system is a common Intel Core i7 CPU at 3.3 GHz with 16GB memory. The cube sampler and similarity metrics were implemented in OpenGL with GLSL shaders to compute the diffusion and integration steps. The precomputation algorithm is implemented in C++ with OpenMP threading.

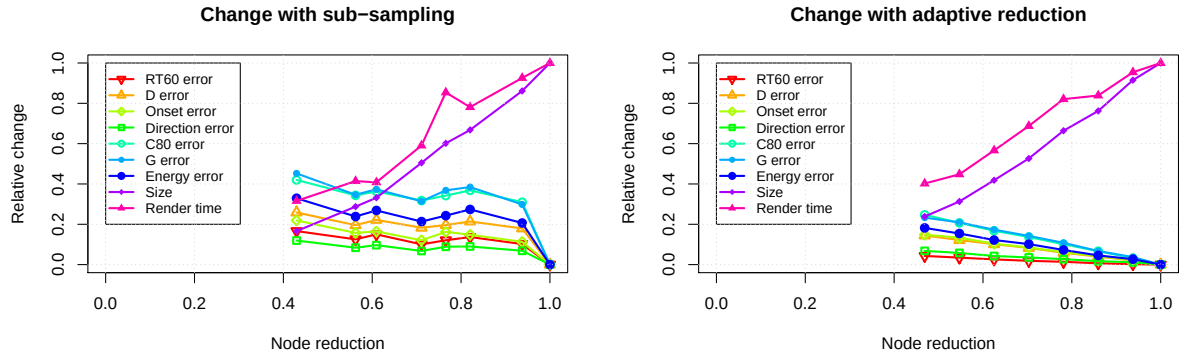
6.1 Similarity and reduction cost

Our similarity metrics are computed on cube faces using the GPU. We used GLSL shaders to sample the surface distance and normals. The distance is computed in the object space and recorded along with the normals. The diffusion metric is computed in one pass based on the gradient for each query using the edge-convolution kernel. The integration step is performed using a simple texture value summation kernel and scaled by the kernel size. In our benchmarks, we found that a kernel size of 4 pixels gives the best performance on our GPU. In the reduction and similarity comparison step, LOS queries are performed using a fast CPU-based BVH ray tracer. All edge-reduction and region computations are performed using custom data structures backed by STL containers. The reduction process has a low computational overhead. Given the appropriate error thresholds, we perform sampling and segmentation on our benchmark scenes. The time to compute the similarity measure at each sample point as well as the time cost to eliminate similar sample points are shown in Table 2. The results in this section are for specular and diffuse responses only. Diffuse reflection requires a trace for each frequency band in order to model per frequency scattering effects. Unless noted, we trace 50k rays for 50 specular and diffraction recursions and 5 diffraction recursions for 4 frequency bands. For our algorithm's reduced results, we selected a reduction of 75%. Depending on goals, an actual user may desire a different reduction amount to adjust the performance vs. accuracy tradeoff. Table 3 shows the cost of simulating diffuse reflections.

Scene	Full grid	Reduced			Time improvement
		Cubemap	Segmentation	Trace	
Simple outdoor	15.14s	1.3s	1ms	6.4s	2.0x
FPS game	141.0s	2.0s	1ms	41.6s	3.2x
Small city	8.8m	8.6s	3ms	4.6m	1.9x
Large city	68m*	26s	69ms	14.4m	4.7x
Canyon	52d*	1.24h	3s	174h	7.17x

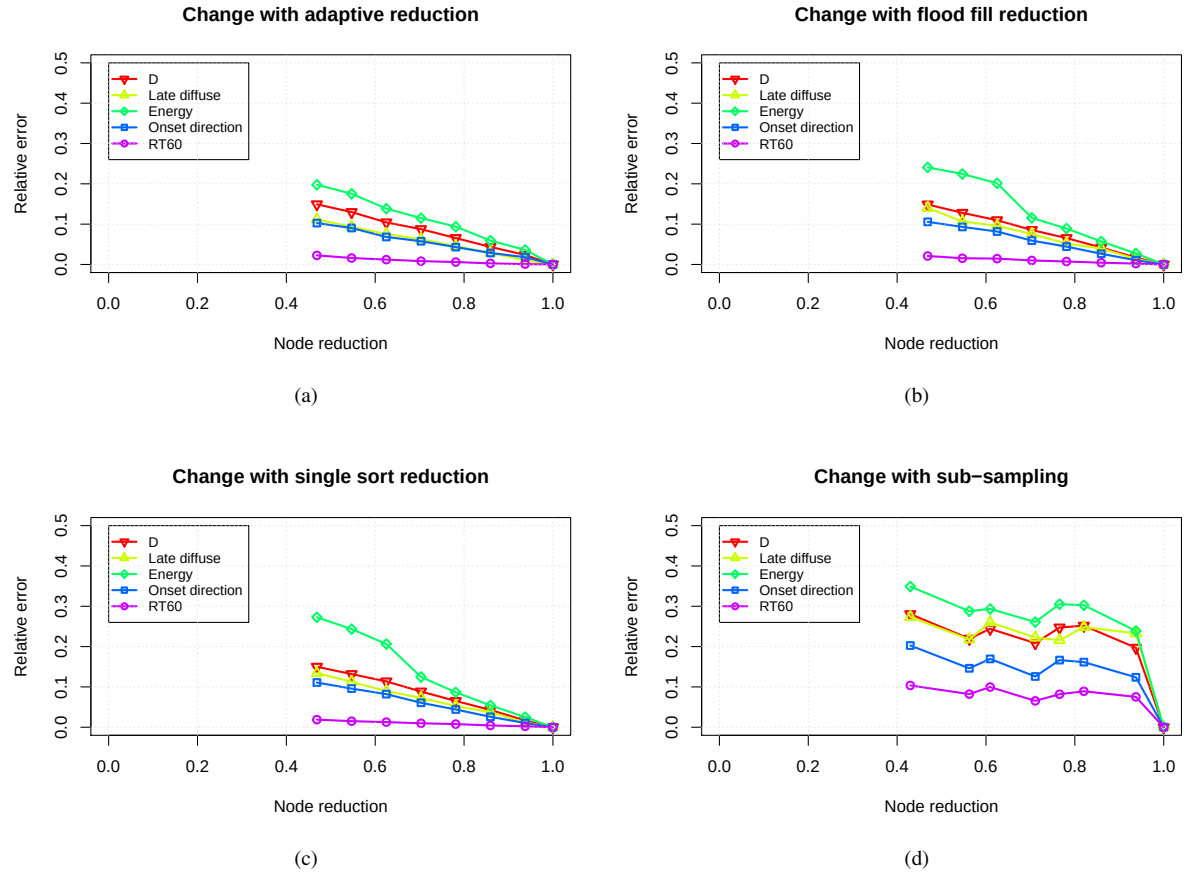
Table 2: Precomputation time cost: *Region segmentation using cube maps allows a significant reduction in precomputation time. The full grid data is generated based on the grid size given in Table 1 for each benchmark. Due to the high time and space cost, times marked with an * are based on partial simulation.*

Our ray based GA simulator is highly optimized for massive scenes. In Table 4, we compare our method to [Schissler et al. 2014]. The [Schissler et al. 2014] algorithm ran on an Intel i7 4770k and we ran our algorithm on an Intel i7 3720QM. Both methods sample the acoustic field at 44.1khz and trace 50k rays for 100 recursions for 4 frequency bands. By building a ray acceleration structure for our receiver detection spheres, we improve tracing performance compared by over an order of magnitude.



(a) (b)

Figure 11: Sampling accuracy vs. error and cost: *Naive subsampling (a) is the most common way of reducing time and storage cost. As the threshold error in our adaptive sampling algorithm changes (b), the overall error in the acoustic evaluation metrics increases, while overall storage, precomputation time, and number of samples decrease (FPS game scene).*



(a) (b)

(c) (d)

Figure 12: Error values for different reduction algorithms corresponding to different acoustic evaluation metrics: *The (a) adaptive algorithm performs better than any other, but the other node placement algorithms guided by our signature, (b) flood fill and (c) sorted merge, perform much better than naive (d) subsampling. These error plots demonstrate the benefit of using our geometric acoustic similarity criteria along with the adaptive scheme as compared to other approaches. For example, the error reduction over sub-sampling algorithms can be large, as compared to that over flood-fill and single sort reduction. These results are for the FPS scene.*

6.2 Precomputation: time and storage

By segmenting the scene and reducing the number of sample points during precomputation, the time and storage costs can be considerably reduced. Our sound propagation simulator is based on discrete ray tracing, accelerated by efficient BVH trees. The ray tracer is heavily multi-threaded, but not SIMD optimized, as high-order re-

	With diffuse		Without	
	full	reduced	full	reduced
FPS: time	554s	223s	141s	42s
FPS: memory	2.0MB	0.3MB	1.3MB	0.3MB
FPS: error	0%	18%	0%	18%
Small city: time	33.5m	14.9m	8.8m	4.6m
Small city: memory	39MB	3.3MB	39MB	5MB
Small city: error	0%	21%	0%	21%

Table 3: Diffuse reflection cost: *Diffuse reflections requires more simulation time and slightly more storage space. The reduced results use a subset of the full sample positions as selected by our algorithm. Reduction results are for 75% node reduction, although the user may select a different reduction amount to adjust performance vs. accuracy tradeoffs.*

Scene	[Schissler et al. 2014]	Ours
FPS (128 nodes)	2.6h	8.8m
Small city (625 nodes)	56h	52.5m

Table 4: Precompute time comparison: *Other simulation methods cannot scale massive scenes. Our reduction algorithm and efficient simulation allow sub-quadratic performance on massive scenes.*

flections are very incoherent. The simulator computes the response at each receiver in the scene from a single source in one simulation cycle. This results in n simulation cycles for n acoustic regions or source positions.

We store our acoustic data as quantized decay filters. A four second reverberation decay profile can be efficiently encoded using 200 blocks, each containing 6 bytes of information, for a total of 1.2 KB. Our reduced sample count and efficient storage structure significantly reduce the storage overhead, as compared to grid based methods. Table 5 compares our storage overhead ("Reduced") to standard grid based methods ("Full").

We design our storage algorithms to reduce both time and space costs. For p sample positions, the time cost is $O(p \log p)$, which is dominated by map operations and sorting the index vector. Additionally, data is stored sparsely when possible. Sparse data, response quantization, and node reduction result in highly compressed acoustic databases (Table 6).

	Full grid		Reduced		Storage
Scene	inserts	storage	inserts	storage	improvement
Simple outdoor	5.5k	1.6MB	1k	0.3MB	5.3x
FPS game	7.3k	1.3MB	1.6k	0.3MB	4.3x
Small city	229k	39MB	7.3k	3MB	13x
Large city	*	*	722k	173MB	*
Canyon	*	*	11.6B	20.8GB	*

Table 5: Storage cost: *We compare storage cost of our reduction algorithm to a full grid, both stored in our efficient sparse data structure.We observe significant improvement for large scenes. Due to the high time and space cost, times marked with an * could not be computed.*

6.3 Runtime cost

The direct sound and early reflection computations are performed on the server for a viable VoIP application. A single server would be expected to support thousands of clients (e.g. Axon supports 8K clients on an 8-core server [Dol 2012]). The results for the maximum number of streams that can be mixed on a single core in realtime are given in Table 7. By implementing our runtime mixing using efficient GPU kernels, we can further scale the approach to thousands of streams in realtime using high-end GPUs [Tsingos et al. 2011].

Method	Error	Space savings
Value quantization	∼2%	75%
No duplicates	None	∼50%
Node reduction	∼20%	∼80%
Total	∼ 22%	∼99.99%

Table 6: Compression: *We combine several algorithms to produce highly compressed scenes*

Scene	# Mixes	Convolve (ms)	Setup (ms)
Walkway	764	16.2	5.0
Game FPS	731	13.7	4.8
Small city	800	15	5.6
Large city	692	14.8	5.2
Canyon	684	14.2	5.8

Table 7: Max realtime mixes per core: *This table shows mixing costs for 500ms of decay data per stream. The setup time includes data structure access and LOS traces. The realtime mixing step in our system is performed in less than 20ms.*

In traditional spatial VoIP systems, occluded voices that are not in direct LOS would fade into silence, and therefore would not be considered for mixing; this indirectly improves the computation and bandwidth scalability of the system. In contrast, we take into account occluded voices for clustering and mixing as they might be audible due to early scattering and reverberation effects. This can decrease the forwarding-to-mixing ratio and requires more runtime processing.

We measure the performance of our algorithm by changing the number of audio streams that can be rendered simultaneously. For our purposes, an audio stream is the data that must be mixed for each source to each receiver. For example, if a single source is within the range of three receivers, three streams would need to be mixed. If source or receiver is moving, the decays must be interpolated at region boundaries. This results in an extra multipy-add for each impulse, decreasing performance by a slight degree. All numbers shown in Table 7 are for moving sources and receivers.

7 Comparison and limitations

In this section, we compare our approach with other precomputation methods and highlight some limitations.

7.1 Comparison

Many precomputation techniques have been proposed for interactive sound propagation and rendering. We compare some features of our approach to other precomputation methods in Table 8.

Numerical propagation: Wave-based precomputation approaches [Raghuvanshi et al. 2010; Mehra et al. 2013; Yeh et al. 2013; Raghuvanshi and Snyder 2014] are more accurate than GA methods for low frequencies and can model higher order diffraction and scattering effects. However, the complexity of this simulation increases as a fourth power of frequency and is a linear function of the volume of the virtual world. As a result, these techniques are currently limited to small indoor scenes and are used only for low frequencies (e.g. less than 1000 Hz) . Although our current implementation is based on GA, we believe that our similarity-measure and decomposition method could also be combined with wave-based solvers [Raghuvanshi et al. 2010; Raghuvanshi and Snyder 2014].

Cell and portals: Many games and interactive applications use cell-and-portal scene decompositions, which can be utilized to precompute higher-order reflections of sound between moving sources

Algorithm	Ours	PART	Wave-grid	IS-gradient	DP-Cache	Reverb-graph
Mem. use	<Low	Low	High	Low	Medium	Low
Convolution	>Realtime	Interactive	Realtime	>Realtime	Realtime	Realtime
Regions	Acoustic	Spatial	None	Cell+Portal	Cell+Portal	Cell+Portal
Directionality	All samples	1st order	1st order	All samples	1st order	1st order
Decomposition	Automatic	Automatic	Automatic	Manual	Manual	Manual

Table 8: Comparison: *We compare some features of our approach with other precomputation methods, including PART [Siltanen et al. 2009], Wave-grid [Raghuvanshi et al. 2010], IS gradient [Tsingos 2009], DP Cache [Foale and Vamplew 2007], and Reverb graph [Stavrakis et al. 2008]. The compression scheme presented in [Raghuvanshi and Snyder 2014] can considerably reduce the memory overhead of Wave-grid [Raghuvanshi et al. 2010].*

and listeners using ray tracing [Foale and Vamplew 2007; Stavrakis et al. 2008; Tsingos 2009]. These approaches typically store IRs sampled at a single position for each cell and/or portal encountered along the paths between the source and the receiver. However, these approaches require significant manual intervention to define regions and portals, making them impractical for large-scale environments. Defining cells and portals suitable for acoustic rendering can often be unintuitive, especially for large outdoor scenes. In contrast, our approach can automatically partition any large environment into acoustic regions.

Frequency and time decomposition: Some techniques perform frequency-domain precomputation based on the acoustic rendering equation [Siltanen et al. 2009], which limits them to static sources. Recently, [Antani et al. 2012] extended the approach by precomputing acoustic transfer operators. However, this approach can only handle a few moving sources, since it performs ray tracing to compute early reflections at runtime. As a result, the runtime overhead can be very high for a large number of sources and receivers. Furthermore, its storage overhead is about $50 - 100$X higher than our approach.

7.2 Limitations

Our approach introduces several approximations with respect to reverberation computation. While our segmentation and precomputation algorithms are independent of frequency, our storage and runtime implementation is highly optimized for voice related frequencies. Speech can often be rendered at low frequency with acceptable results, for example, cellular phones often sample audio at 8 kHz [ETSI 2014]. Our implementation renders all audio at 16 kHz. In order to support high frequency 44.1 kHz audio, memory and runtime costs would increase by approximately 2.75 times (i.e. a linear function of the highest frequency).

Within regions, the direct path can be rendered with occlusion and distance attenuation. However, the decay data has been precomputed and cannot be adjusted within region. We attempt to mitigate this issue by restricting the region sizes and apply a heuristic scaling to the early response. Nonetheless, the interpolation within regions is not physically based and may lead to incorrect decay responses.

We render integrated decay profiles reconstructed with a random phase, rather than the original impulse response. This reduces the accuracy with which the early reflections can be rendered. In particular, flutter echoes might not be captured by our approach unless the number of sub-bands is increased. Since we use GA simulation, it cannot accurately simulate all wave effects (e.g. low frequency effects).

Since our similarity measure computation is a heuristic based on scene geometric representation, which only takes into account first-order reflections or responses, this formulation may fail in some cases to accurately estimate the late responses. In scenes where the depth variance is large, the early response time cannot be reliably estimated from the local geometric representation. Moreover, in scenes with high depth complexity or occlusion, the diffraction paths contribute significantly towards the early responses, and our approach may not work well in such scenes. Finally, our reduction metrics can be overly conservative in some scenes, resulting in less time and storage benefits.

8 Conclusion and future work

We present a new approach to generating environmental voice reverberation in large virtual worlds. Our algorithm scales with the size of the model and computes early and late acoustic responses complete with diffraction and reverberation effects. We use a local geometrical similarity metric to efficiently sample key positions and create zones with similar acoustical properties. Due to our sample reduction algorithm and efficient storage structures, we observe more than an order of magnitude improvement in precomputation time and storage overhead. We demonstrate results on kilometer-sized virtual worlds with a large number of sources and receivers. The size of the environment and the number of sources used in our simulation are much larger than the ones used in recent real-time propagation algorithms. In practice, our approach can generate plausible environmental audio effects and is targeted towards gaming and virtual environments.

There are many avenues for future work. In addition to overcoming the stated limitations, we plan to investigate techniques to update the acoustic response based on dynamic objects using precomputed filters. It would be useful to extend our acoustic similarity metric and adaptive similarity metric to wave-based propagation techniques [Raghuvanshi et al. 2010; Raghuvanshi and Snyder 2014] to improve our computation for low-frequency effects. We also would like to investigate possible perceptual reduction techniques to further reduce the number of samples that need to be stored. We need to perform more validation and error analysis, especially for large outdoor scenes. Our current error analysis is based on standard ISO-3382 parameters, which were primarily designed for room acoustics. We plan to integrate our approach in a large-scaled VoIP system and evaluate its performance. We believe that a user study to evaluate the accuracy of reduced sampling strategies and plausibility would be useful.

References

ANDŌ, Y. 1998. *Architectural Acoustics: Blending Sound Sources, Sound Fields, and Listeners*. Modern Acoustics and Signal Processing Series. Springer Verlag.

ANTANI, L., CHANDAK, A., SAVIOJA, L., AND MANOCHA, D. 2012. Interactive sound propagation using compact acoustic transfer operators. *ACM Trans. Graph. 31*, 1 (Feb.), 7:1–7:12.

BAILEY, R., AND BRUMITT, B. 2010. Method and system for automatically generating world environment reverberation from a game geometry. Tech. rep., U.S. Patent Application US 2010/0008513 A1, January.

BENESTY, J., GAENSLER, T., AND ENEROTH, P. 2000. Multi-channel sound, acoustic echo cancellation, and multi-channel time-domain adaptive filtering. In *Acoustic Signal Processing for Telecommunication*. Kluwer Academic Publishers, 101–120.

BERNDT, P., HOVESTADT, M., AND KAO, O. 2011. Crowd buzz: Scalable audio communication for mmves using latency optimized hypercube gossiping. In *Haptic Audio Visual Environments and Games (HAVE), 2011 IEEE International Workshop on*, IEEE.

BOUSTEAD, P., AND SAFAEI, F. 2004. Comparison of delivery architectures for immersive audio in crowded networked games. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, ACM, New York, NY, USA, NOSSDAV '04, 22–27.

CALAMIA, P., AND SVENSSON, U. P. 2007. Fast time-domain edge-diffraction calculations for interactive acoustic simulations. *EURASIP Journal on Advances in Signal Processing*.

CHANDAK, A., ANTANI, L., TAYLOR, M., AND MANOCHA, D. 2009. Fastv: From-point visibility culling on complex models. In *Eurographics Symposium on Rendering*.

2012. Dolby axon surround sound chat for gamers. http://www.dolby.com/us/en/consumer/technology/gaming/dolby-axon.html.

ETSI, 2014. TS 126 071 V12.0.0; AMR speech Codec; General description.

FOALE, C., AND VAMPLEW, P. 2007. Portal-based sound propagation for first-person computer games. In *Proceedings of the 4th Australasian conference on Interactive entertainment*, IE '07, 9:1–9:8.

FUNKHOUSER, T., CARLBOM, I., ELKO, G., PINGALI, G., SONDHI, M., AND WEST, J. 1998. A beam tracing approach to acoustic modeling for interactive virtual environments. In *Proc. of ACM SIGGRAPH*, 21–32.

FUNKHOUSER, T. A., MIN, P., AND CARLBOM, I. 1999. Real-time acoustic modeling for distributed virtual environments. In *Proc. of ACM SIGGRAPH*, 365–374.

GIBBS, M., WADLEY, G., AND BENDA, P. 2006. Proximity-based chat in a first person shooter: using a novel voice communication system for online play. In *Proceedings of the 3rd Australasian conference on Interactive entertainment*, Murdoch University, Murdoch University, Australia, Australia, IE '06, 96–102.

GOODE, B. 2002. Voice over internet protocol (voip). In *Proceedings of the IEEE*, 1495 – 1517.

GREENE, N. 1986. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications 6*, 11 (Nov.).

HALLORAN, J. 2009. It's talk, but not as we know it: Using voip to communicate in war games. In *Proceedings of the 2009 Conference in Games and Virtual Worlds for Serious Applications*, VS-GAMES '09, 133–140.

HOLLIER, M. P., RIMELL, A. N., AND BURRASTON, D. 1997. Spatial audio technology for telepresence. *BT Technology Journal 15*, 4, 33 – 41.

ISO 3382, 2009. Measurement of room acoustic parameters.

JAMES, D. L., BARBIC, J., AND PAI, D. K. 2006. Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. In *Proc. of ACM SIGGRAPH*, 987–995.

JIANG, J.-R., WU, J.-W., FAN, C.-W., AND WU, J.-Y. 2014. Immersive voice communication for massively multiplayer online games. *Peer-to-Peer Networking and Applications*, 1–13.

JOT, J.-M. 1999. Real-time spatial processing of sounds for music, multimedia and interactive human-computer interfaces. *Multimedia Systems 7*, 1, 55–69.

KUTTRUFF, H. 2007. *Acoustics: An Introduction*. Taylor and Francis, New York.

LIANG, K., SEO, B., KRYCZKA, A., AND ZIMMERMANN, R. 2013. Idm: An indirect dissemination mechanism for spatial voice interaction in networked virtual environments. *Parallel and Distributed Systems, IEEE Transactions on 24*, 2, 356–367.

MARKOPOULOU, A., TOBAGI, F., AND KARAM, M. 2002. Assessment of voip quality over internet backbones. In *IEEE INFOCOM 2002*, 150 – 159.

MEHRA, R., RAGHUVANSHI, N., ANTANI, L., CHANDAK, A., CURTIS, S., AND MANOCHA, D. 2013. Wave-based sound propagation in large open scenes using an equivalent source formulation. *ACM Trans. on Graphics 32*, 2, 19:1–19:13.

MERIMAA, J., AND PULLKI, V. 2004. Spatial impulse response rendering. *Proc. of the 7th Intl. Conf. on Digital Audio Effects (DAFX'04)* (Oct.).

POPE, J., CREASEY, D., AND CHALMERS, A. 1999. Realtime room acoustics using ambisonics. *Proc. of the AES 16th Intl. Conf. on Spatial Sound Reproduction*, 427–435.

RADENKOVIC, M., GREENHALGH, C., AND BENFORD, S. 2002. Deployment issues for multi-user audio support in cves. In *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, VRST '02, 179–185.

RAGHUVANSHI, N., AND SNYDER, J. 2014. Parametric wave field coding for precomputed sound propagation. *ACM Transactions on Graphics (TOG) 33*, 4, 38:1–38:11.

RAGHUVANSHI, N., SNYDER, J., MEHRA, R., LIN, M., AND GOVINDARAJU, N. 2010. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. In *ACM Trans. on Graphics*, vol. 29, 68:1 – 68:11.

SAFAEI, F. 2005. Dice: Internet delivery of immersive voice communication for crowded virtual spaces. In *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, IEEE Computer Society, Washington, DC, USA, VR '05, 35–41.

SALLNÄS, E.-L. 2005. Effects of communication mode on social presence, virtual presence, and performance in collaborative virtual environments. *Presence: Teleoper. Virtual Environ. 14*, 4, 434–449.

SAVIOJA, L. 2010. Real-Time 3D Finite-Difference Time-Domain Simulation of Mid-Frequency Room Acoustics. In *13th International Conference on Digital Audio Effects (DAFx-10)*.

SCHISSLER, C., AND MANOCHA, D. 2014. Interactive sound propagation and rendering for large multi-source scenes. Tech. rep., Department of Computer Science, University of North Carolina. Submitted for publication.

SCHISSLER, C., MEHRA, R., AND MANOCHA, D. 2014. High-order diffraction and diffuse reflections for interactive sound propagation in large environments. *ACM Trans. Graph. 33*, 4 (July), 39:1–39:12.

SCHROEDER, M. R. 1965. New method of measuring reverberation time. *The Journal of the Acoustical Society of America 37*, 409.

SILTANEN, S., LOKKI, T., KIMINKI, S., AND SAVIOJA, L. 2007. The room acoustic rendering equation. *The Journal of the Acoustical Society of America 122*, 3 (September), 1624–1635.

SILTANEN, S., LOKKI, T., AND SAVIOJA, L. 2009. Frequency domain acoustic radiance transfer for real-time auralization. *Acta Acustica 95*, 1, 106–117.

STAVRAKIS, E., TSINGOS, N., AND CALAMIA, P. 2008. Topological sound propagation with reverberation graphs. *Acta Acustica/Acustica - the Journal of the European Acoustics Association 94*, 921–932.

TAYLOR, M., CHANDAK, A., MO, Q., LAUTERBACH, C., SCHISSLER, C., AND MANOCHA, D. 2012. Guided multiview ray tracing for fast auralization. *IEEE Transactions on Visualization and Computer Graphics 18*, 1797–1810.

TSINGOS, N., GALLO, E., AND DRETTAKIS, G. 2004. Perceptual audio rendering of complex virtual environments. *ACM Trans. Graph. 23*, 3, 249–258.

TSINGOS, N., DACHSBACHER, C., LEFEBVRE, S., AND DELLEPIANE, M. 2007. Instant sound scattering. In *Proceedings of the Eurographics Symposium on Rendering*, 111–120.

TSINGOS, N., JIANG, W., AND WILLIAMS, I. 2011. Using programmable graphics hardware for acoustics and audio rendering. *Journal of Audio Engineering Society 59*, 9, 628–646.

TSINGOS, N. 2009. Pre-computing geometry-based reverberation effects for games. *35th AES Conference on Audio for Games*.

VORLÄNDER, M. 1989. Simulation of the transient and steady-state sound propagation in rooms using a new combined ray-tracing/image-source algorithm. *The Journal of the Acoustical Society of America 86*, 1, 172–178.

WADLEY, G., GIBBS, M., AND BENDA, P. 2007. Speaking in character: using voice-over-ip to communicate within mmorpgs. In *Proceedings of the 4th Australasian conference on Interactive entertainment*, IE '07, 24:1–24:8.

WEST, J., BLAUERT, J., AND MACLEAN, D. 1992. Teleconferencing system using head-related signals. *Applied Acoustics 36*, 3-4, 327 – 333.

WILLIAMS, D., CAPLAN, S., AND XIONG, L. 2007. Can You Hear Me Now? The Impact of Voice in an Online Gaming Community. *Human Communication Research 33*, 4, 427–449.

YEH, H., MEHRA, R., REN, Z., ANTANI, L., MANOCHA, D., AND LIN, M. 2013. Wave-ray coupling for interactive sound propagation in large complex scenes. *ACM Trans. Graph. 32*, 6, 165:1–165:11.

ZIMMERMANN, R., AND LIANG, K. 2008. Spatialized audio streaming for networked virtual environments. In *Proceedings of the 16th ACM international conference on Multimedia*, ACM, New York, NY, USA, MM '08, 299–308.

Rendering Environmental Voice Reverberation for Large-scale Distributed Virtual Worlds

2 Appendix

In this supplementary material we include details on various aspects of our algorithm and its error analysis. We first describe the error metrics and how the error of each acoustic property is computed (Section 2.1). In Section 2.2 we highlight the error maps for some benchmarks to evaluate the accuracy of our adaptive sampling algorithm. Section 2.3 shows the errors for all the metrics used in our similarity analysis. We highlight the errors for each metric separately as well all four combined metrics. We show that all four metrics are need for robust segmentation. Section 2.4 gives the details (and pseudo-codes) for various algorithm described in the paper.

2.1 Error metrics

We evaluate the accuracy of our simulation by comparing properties of the resulting acoustic field. Below, we describe the details of error calculation for each metric. Refer to Section 5.3 for a general overview of how the metrics are compared. Figure 1 is similar to Figure 12 in the paper and shows the error of some of these properties for various reduction strategies.

Onset delay: This is the time it takes for the earliest first-order path (reflection or diffraction) to reach the receiver. This is very similar to the Initial Time Delay Gap [Andō 1998], but we cannot guarantee the presence of a direct contribution; so we measure the delay from the initial simulation time. We compute this parameter by finding the delay of the first impulse in the acoustic response. Moreover, we assume that a missing value indicates that the onset occurred outside the measurement range (4 seconds in our implementation). In these cases we set the error to the difference between the maximum measurable onset and the known delay. Otherwise, this error is computed as a time difference between the two response onset delays.

Onset direction: This is the average direction of all contributions for the initial impulse. It is the first directional sound that a listener hears in the absence of direct sound and is important for localization. We compute this parameter by averaging the directions of all contributing paths that contribute the first impulse. This error is computed as the dot product of the normalized onset directions of the responses. We scale the error over the $[0, 1]$ interval, where an error of 1 represents the maximum error of 90°. Since highly diffuse signals will have nearly random direction, the direction error is scaled by the strength of the spatial response. This means that highly diffuse signals will produce low directional error, while signals with less diffusion could produce more error.

Reverberation (RT60): This corresponds to the time it takes for sound waves to decay past a certain threshold. In particular, RT60 is the time needed for the sound to decay -60dB. The reverberation decay helps the listener to determine the size of the space. We compute this using the Schroeder method [Schroeder 1965; ISO 3382 2009] by matching a least-squares fit of the pressure decay in log space. We compute the time to decay to -60 dB by fitting a least square line to both the decay responses; we report the error as the time difference in decay time. For reverberation, we assume that a missing value indicates that the signal decayed immediately, so the error is the decay time from the other signal.

Definition (D) This value represents the ratio between energy levels in the early and late portions of the acoustic response. We use D, the ratio of the integral of the first 50 milliseconds of energy to the total energy in the response, as defined in [Kuttruff 2007; ISO 3382 2009]. D represents the intelligibility of speech signals. If both responses have a value, the error is the difference between the measured D values. If the response is empty, we assume it has a D value of 0.

Clarity (C80) This value is similar to D, except it is the ratio between the energy in the first 80 milliseconds of the response to the rest of the energy [Kuttruff 2007]. In short responses, this can lead to a division by 0 if there is no energy past the initial 80 milliseconds. We compute error in this value as a percent difference in decibel values. If there is no energy in the later portion of the response, we avoid dividing by 0 and assign a default value of 20 decibels.

Strength (G) This value represents the strength of the response as a ratio of response energy to the direct path at 10m [Kuttruff 2007]. We compute error in this value as a percent difference in decibel values.

Total energy (Energy) This value represents the sum of the sound pressure levels over the length of the response. Before the sum, the frequency band values are averaged into a single time domain response. We compute relative error in this value as a percent difference in decibel values.

Diffuseness (Diffuse) We compute the diffuseness of the response as a function of the direction of incoming sound at a sample position over time. If majority of the sound arriving at a sample position for a given time index is from a similar direction, that response time sample will have low diffuseness. If the sound arrives from many different directions, the time sample will have high diffuseness. This value is computed by finding the mean direction of the contribution paths arriving at a receiver. The length of the resulting vector is the ratio of diffuseness. To compute the error, we find the average diffuseness for the signal over time, then taking the difference from the known diffuseness. If a diffuseness value is missing, we assume a maximum error of 1.

2.2 Example maps

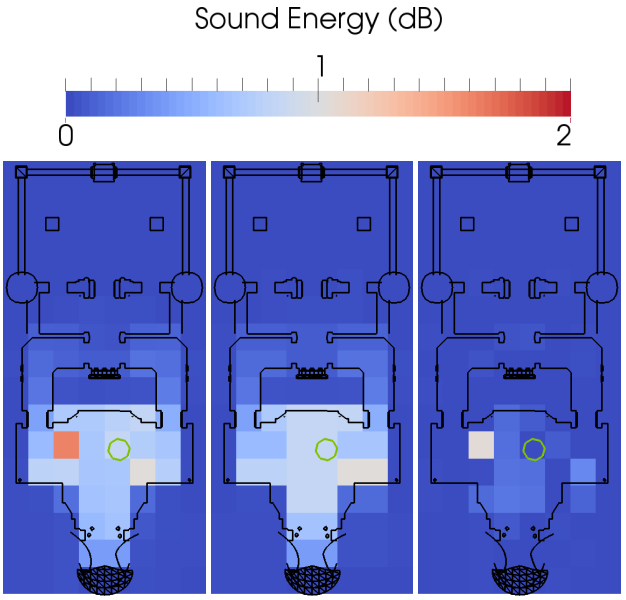


Figure 2: Energy error maps: *We compute sound energy for the (a) full dataset and (b) our reduced dataset. The details of these datasets are given in Table 2. The difference between these datasets represents the error in our solution (c). The total energy values for the source position outlined in green are shown.*

Since it is difficult to report the thousands of error values from a sin-

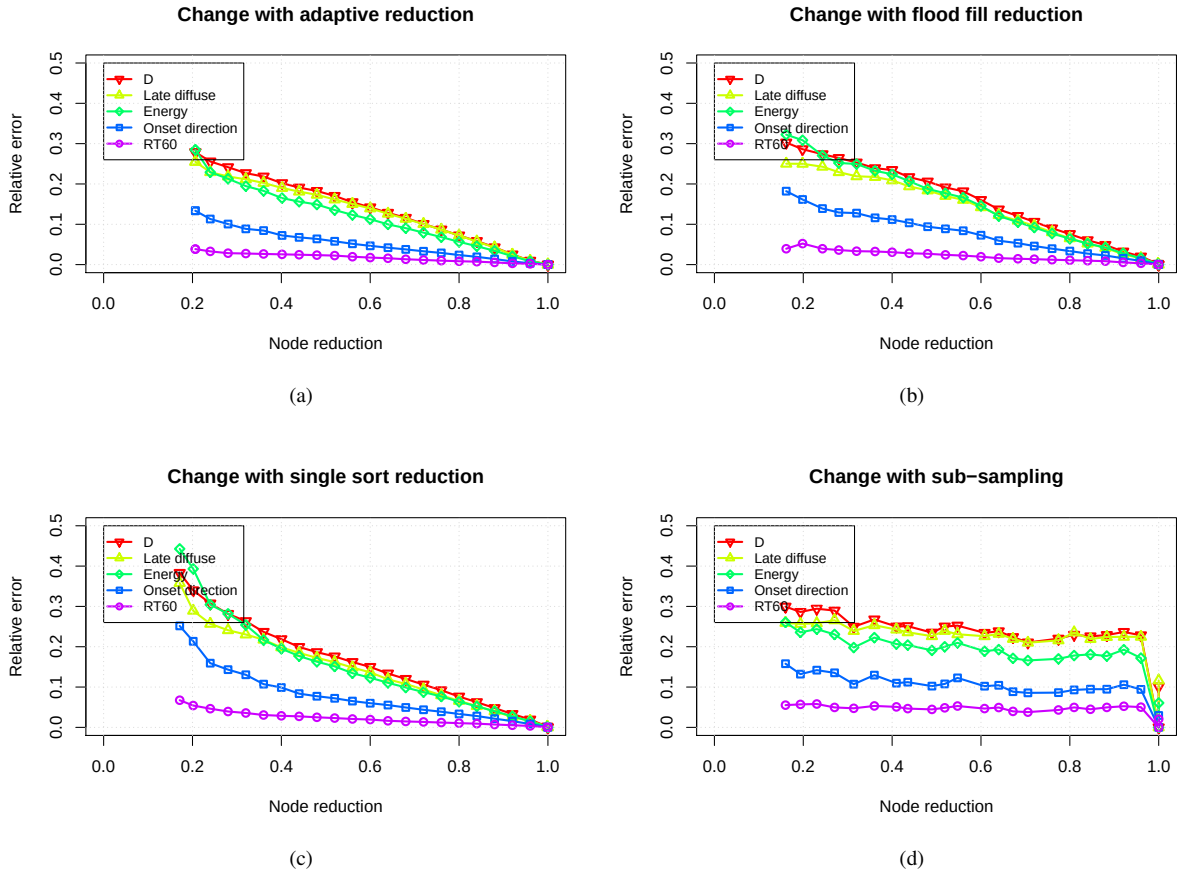


Figure 1: Error values for different reduction algorithms corresponding to different acoustic evaluation metrics (City): *This is similar to Figure 12 in the paper, but for a different benchmark. It highlights the error values for different reduction algorithms corresponding to different acoustic environment metrics. The (a) adaptive algorithm performs better than any other, but the other node placement algorithms guided by our signature*

gle scene, we compute various maps that average the error values over the entire scene. All of these calculations are performed on the acoustic evaluation metrics defined above. The most intuitive error is the difference between a property f from a ground-truth acoustic response and a property c from our region-segmented acoustic response. For a given source position, we measure the acoustic property difference of all receiver positions over the entire map as $|f - c|$. This differencing process is shown in Figure 2.

We weight the acoustic property error relative to audibility a, which we define as the maximum decibel value in the acoustic response. The error for a response is scaled over the range 0 to 2 for audibility of -60 to 0 dB. Thus, the error in quiet responses is weighted less, while error in loud responses is weighted more.

Using these difference maps, we compute the relative error in acoustic evaluation metric over the range of that metric (g) and scale it by the audibility factor a as $\frac{|f-c|}{g}a$. The average relative error A for a single source position s to any of n receiver positions can be given as

$$A = \frac{\sum_{s=0}^{n} \frac{|f_s - c_s|}{g} a_s}{\sum_{s=0}^{n} a_s}$$

Figure 3 shows the value A visually. The average relative error M is computed for all positions: $\sum_{s=0}^{n} A_s/n$. This is the value shown in Figure 11. We note that for large maps, many responses are silent, as the sound decays before reaching distant receivers. These responses are trivially equal (i.e. empty) and are not included in any error calculations.

We have computed the ground-truth for the FPS Game scene using our simulator with no segmentation. The results in Figure 4 are the relative error for an initial sampling of 4 meters with a merge threshold S_{thr} set such that the simulated node count is 75% of the original grid sample count.

2.3 Metric analysis

We use four metrics in our similarity measure: distance, direction, diffuseness, and material. We weight the metrics identically since it is difficult to know which metric will perform best for a given scene. For example, the material metric may be very useful in an office environment with many different materials, but less useful in a stone cathedral where all materials share similar properties. In Figure 6 we show the results of reduction with some metrics disabled. We have highlighted these results for Small City benchmark in the paper and for the City benchmark and FPS benchmark here. We can only get low error and significant node reduction when all four metrics are used (i.e. a robust solution).

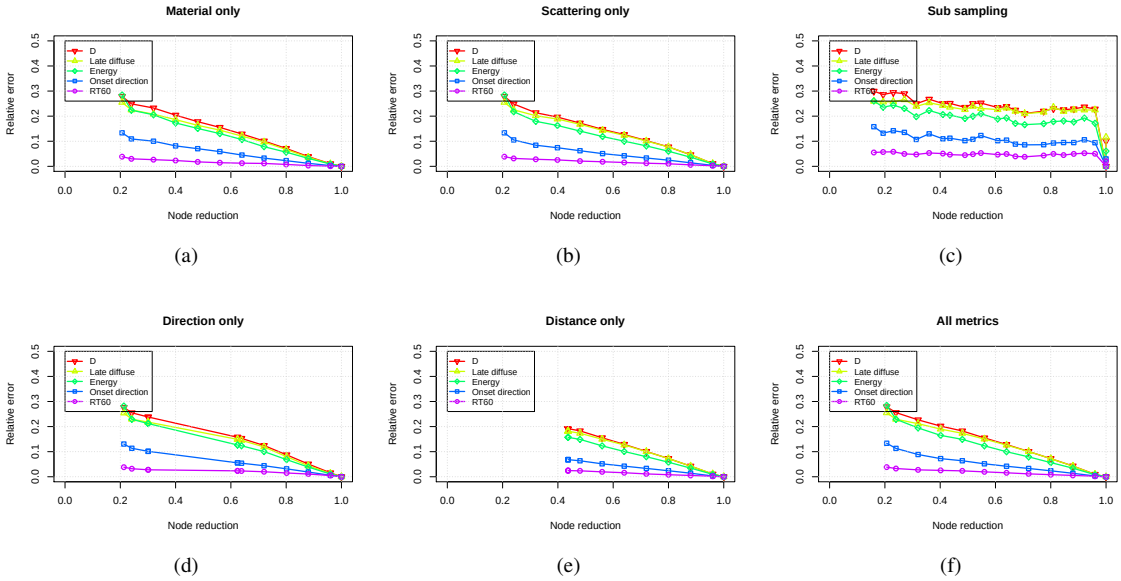


Figure 5: Individual metric results (Small City): *We show the reduction results when only a single metric is enabled on the Small City scene. Single metrics cannot robustly predict the acoustic field: distance (e) alone cannot reduce node count below 40% (because of our criteria) and direction (d) fails to robustly segment the scene for some values of S. For general scenes, all metrics are required for robust segmentation, as they allow significant node reduction (i.e. reduced storage overhead) and lower error in our sampled representation.*

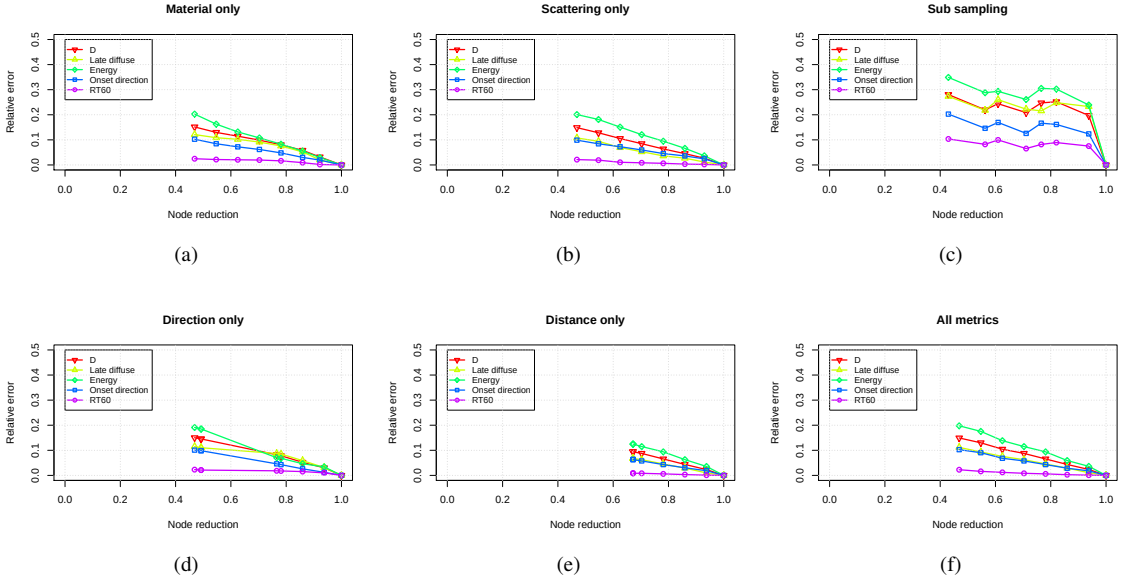
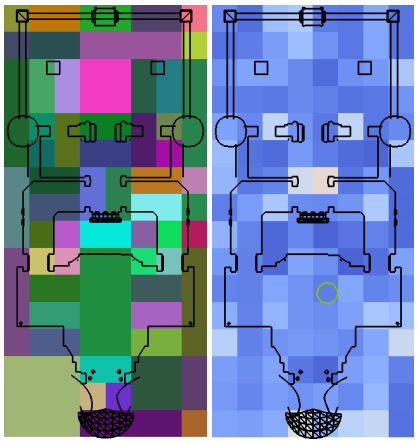


Figure 6: Individual metric results (FPS): *Similar to the Individual metric results shown in Figure 6, we show the reduction results when only a single metric is enabled on the FPS scene. This plot demonstrates the benefit of using all four metrics, as opposed to a single metric. For general scenes, all metrics are required for robust segmentation, as they allow significant node reduction (i.e. reduced storage overhead) and lower error in our sampled representation.*



(a) (b)

Figure 3: Segmentation region error: *The segmentation map for the FPS scene is shown in (a), where each unique color is an acoustic region. The total energy relative error resulting from this segmentation is shown in (b) We show the source position sampled in Figure 2 as a green circle. The legend for (b) is the same as the legend in Figure 4.*

2.4 Algorithm details

Prior to simulation, acoustic regions are identified. The details of the adaptively merging algorithm is given in Algorithm 1.

Algorithm 2 describes how responses are inserted into our storage structure during precompute. Our storage algorithm is extremely efficient on large scenes by not storing empty responses and coalescing duplicate data.

At runtime, the storage structure is queried for acoustic response data. Algorithm 3 shows the details of this process.

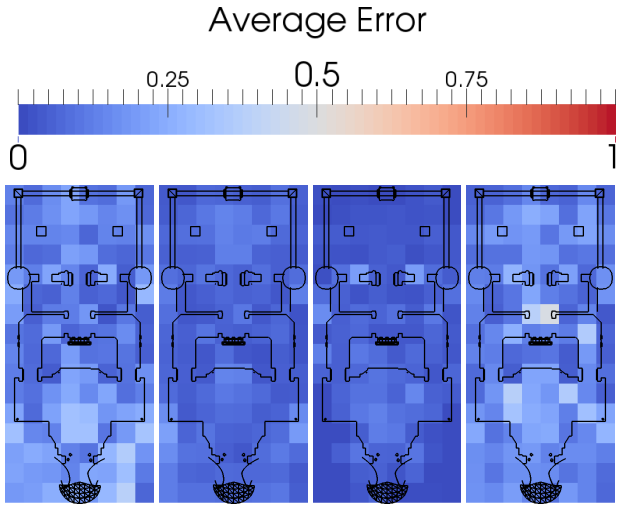
Algorithm 1 Adaptively merge regions with similar signatures. Each region can be subdivided into four subregions. Neighboring subregions could form larger rectangular regions. In this algorithm, the ordering of merges ensures that the largest possible regions are merged. That is, if there is a possible merge of a rectangular region, it is used. The $similar()$ function tests if the signatures in the region satisfy the LOS and $s_p < s_{the}$ constraints.

function MERGE(corner, size)
  $c \leftarrow corner$
  $s \leftarrow size$
  $h \leftarrow size/2$
  $entireRegion \leftarrow (c, c), (s, s)$
  $topRect \leftarrow (c, c), (s, h)$
  $botRect \leftarrow (c, c + h), (s, h)$
  $lefRect \leftarrow (c, c), (h, s)$
  $rigRect \leftarrow (c + h, c), (h, s)$
  $topLef \leftarrow (c, c), (h, h)$
  $topRig \leftarrow (c + h, c), (h, h)$
  $botLef \leftarrow (c, c + h), (h, h)$
  $botRig \leftarrow (c + h, c + h), (h, h)$

  if similar($entireRegion$) **then**
    merge($entireRegion$)
  else if similar($topRect$) and similar($botRect$) **then**
    merge($topRect$); merge($botRect$)
  else if similar($lefRect$) and similar($rigRect$) **then**
    merge($lefRect$); merge($rigRect$)
  else if similar($topRect$) **then**
    merge($topRect$); merge($botLef$); merge($botRig$)
  else if similar($botRect$) **then**
    merge($botRect$); merge($topLef$); merge($topRig$)
  else if similar($lefRect$) **then**
    merge($lefRect$); merge($topRig$); merge($botRig$)
  else if similar($rigRect$) **then**
    merge($rigRect$); merge($topLef$); merge($botLef$)
  else
    merge($topLef$); merge($topRig$)
    merge($botLef$); merge($botRig$)
  end if
end function

Algorithm 2 Insert response data. All the insertions are performed during simulation phase. We give details of our insertion algorithm based on the hash table representation.

function INSERTRESPONSEDATA(data, location)
  if $data = emptyData$ **then**
    return
  end if
  $ptrD \leftarrow pointer(data)$
  $id \leftarrow map.Query(ptrD)$
  if $id = emptyID$ **then**
    $list.Insert(data)$
    $ptrD \leftarrow pointer(list.End)$
    $map.Insert(ptrD)$
    $pair(location, list.Size)$
    $hashmap.Insert(pair)$
  else
    $pair(location, id)$
    $index.Insert(pair)$
  end if
end function



(a) (b) (c) (d)

Figure 4: Relative error maps: *We compute the relative error in FPS scene with respect to different evaluation metric: (a) onset delay; (b) onset wave direction; (c) RT60; (d) and definition D. A wireframe of the scene is overlaid on the error maps. Red areas indicate high error. In most regions the errors in terms of onset delay, RT60 and definition are low. A few locations result in high values of the onset direction relative error.*

Algorithm 3 Get response data. This query is performed at runtime to lookup the acoustic data vector locations.

function GETRESPONSEDATA(location)
  $id \leftarrow index.Query(location)$
  if $id = emptyID$ **then**
    return $emptyData$
  else
    return $data[id]$
  end if
end function