Spatializer: A Web-based Positional Audio Toolkit

Beomjoo Seo, Min Min Htoon, Roger Zimmermann
School of Computing
Natl. University of Singapore, Singapore 117417
[seobj, htoon, rogerz]@comp.nus.edu.sg

Chung-Dau Wang
Audiary Pte Ltd.
I-Cube Building, 21 Heng Mui Keng Terrace
Singapore 119613
chungdaw@audiary.com

ABSTRACT
The web is becoming increasingly multimedia enabled and visuals have paved the way to the point where high quality online videos and detailed three-dimensional virtual worlds are now a reality. Innovative audio applications have been lagging somewhat, but positional audio system are gradually gaining attention as they help people to attain a better spatial perception and immersion into three-dimensional virtual spaces by presenting important spatial cues. However, existing positional audio systems provide either interactive, live voice communications for networked virtual environments or static sound sources, but not both. Here we describe our web-based prototype system called Spatializer that encapsulates positional audio functionalities for both dynamic voice communications and static sound sources, all in a peer-to-peer toolkit with a simple yet powerful unified web interface.

Categories and Subject Descriptors
H.4.3 [Information Systems Applications]: Communications Applications; C.2.4 [Computer-Communication Networks]: Distributed Systems - Distributed applications

General Terms
Design, Human Factors, Verification

Keywords
Peer-to-peer, overlay topology, positional audio, virtual environments, VoIP, XMPP

1. INTRODUCTION
Positional audio, i.e., the rendering of audio sources across a plane surrounding the user, is a popular feature that is successfully used to make many game applications more appealing. Developers utilize 3D audio rendering engines such as OpenAL and the FMOD library which provide simple interfaces that hide complicated implementation details. However, existing solutions for 3D sound have a number of limitations that we aim to address with our web-based positional audio toolkit.

First, existing engines are generally limited to the rendering of static audio sources (e.g., environment sounds such as engine noise and explosions) which are accessed from files either locally or remotely. With the recent success of networked multi-player on-line games such as World of Warcraft (WoW) and the development paradigm of Rich Internet Applications (RIAs), the game vendors have migrated their applications to newer, networked platforms (e.g., Adobe Flash). With multi-player games, communication between participants becomes important. However, currently no positional audio engine exists that integrates interactive voice communication among players and the playback of static sounds in 3D space. As a result of this situation separate solutions have emerged. For example, the WoW client uses the FMOD library to render static sounds and a proprietary engine for voice, which does not preserve the spatial cues of voiced communications.

A second challenge is that most peer clients tend to have asymmetric network connectivity, where the download bandwidth is typically an order of magnitude larger than the upload capacity. Thus, the careful use of the upload data rate is a crucial aspect in enabling scalability and widespread deployment of distributed audio applications. Since the upload bandwidth is generally the bottleneck resource, we considered that limitation in our design.

Third, if interactive voice communication is handled with a client-server paradigm, there exists asynchrony between the rendered audio streams and the avatars’ positional movements because the audio spatialization task is executed on a remote server. Hence the avatar movement may occur slightly ahead or behind the sound “movement” (i.e., render location re-positioning). Such movement asynchrony can disrupt the user’s perceptual experience. In a peer-to-peer (P2P) model this can be avoided since the audio spatialization and rendering are performed on the local machine and thus are tied with no lag to the avatar movement. Although low latency among players is undeniably a key factor for good interactivity, we believe that it is equally or more critical for spontaneously voiced speech to closely match players’ movements. If this is achieved it will positively impact a user’s perception of the system responsiveness.

Based on these observations, we have set out to develop a peer-to-peer-based positional audio system with the following specific aims: (1) enabling high-quality scalable spatialized voice conferencing, (2) providing low processing latency for better interactivity, (3) maintaining the synchronization
of users’ movements and their sounds, and (4) providing a simple yet powerful unified interface that supports the rendering of static sounds and interactive voices.

To demonstrate the feasibility of our approach, we have developed a prototype system called Spatializer that encapsulates the positional audio engine as a web browser plug-in. Our implementation demonstrates the feasibility of the plug-in to provide a spatialized “audiosphere” for sounds being rendered from multiple browser windows simultaneously.

2. RELATED WORK

Among the available 3D voice engines, client/server-based solutions such as Vivox (vivox.com), Dolby Azon (dolby.com) and Mumble (mumble.sourceforge.net) have been successfully deployed not only in many games but also in networked virtual environments. With this approach, a voice server collects the position information of the participating players, then constructs spatialized audio bitstreams and delivers them to the participants. While requiring low network bandwidth for each voice client, such centralized solutions are exposed to server-side performance issues. For example, we measured the end-to-end delay of the Vivox voice solution in the Second Life virtual world, using the measurement method described by Hao Huang [1]. Even excluding the transmission delay, the measured processing delay was lengthy at a consistent 350 ms; the processing delay of the official WoW solution (without positional audio) was even longer at 600 ms or more.

Unlike the above mentioned client/server-based positional audio systems, Boustead et al. [2] proposed a differentiated treatment of a player’s auditory scene. In their model, voiced speeches coming from nearby neighbors are directly delivered to a listening player, while the voices from faraway players are delivered via a voice server. This method achieves synchronization of audio and movement events, which the traditional model cannot maintain, but it still suffers from a server bottleneck. Hence our solution focuses on a peer-to-peer design.

3. DESIGN

Our toolkit, called Spatializer, consists of three layers (illustrated in Figure 1): a Javascript API, a browser abstraction layer, and our positional audio library. The Javascript API layer allows web developers to easily access our audio library via simple Javascript calls. It includes remote voice-session and static-sound-session management, location handlers (i.e., bi-directional position information updates), and Extensible Messaging and Presence Protocol (XMPP) handlers (to reflect status changes of registered XMPP users). The browser abstraction layer maps the Javascript API to browser-specific internal calls. For example, the Firefox browser allows users to access its extensions via Mozilla’s Cross Platform COM (XPCom) interfaces, while similar functionality is provided in Internet Explorer via ActiveX controls. The positional audio library layer encapsulates a set of software components namely the audio rendering engine, the P2P traversal framework, and overlay topology management. The following sections describe the design issues and choices of each component.

3.1 Audio Rendering Engine

The audio rendering engine includes the following noteworthy features: (1) the use of existing audio library for normal audio-related operations, (2) efficient binaural localization, and (3) multi-source multi-sink support.

3.1.1 Platform-independent Audio Library

Based on our previous experience [reference remove for review process], we found two issues especially challenging: providing platform independence and avoiding in-buffer data accumulation. One of our early prototypes was written on top of the legacy Windows audio services (MME, MultiMedia Extensions), we had difficulty to fully utilize the features of the latest Windows audio stack. Moreover, we were unable to port our system to different operating systems. As more and more Internet applications are running in web-based environments, platform independence has become an important feature. The buffer accumulation problem originates in the low-level mismatch between the sound sampling rate and the playback rate due to clock skew on (inexpensive) personal computer soundcards. To hide its effects, we originally employed intentional packet drops of silence data, which still lead to some aperiodic, noticeable clicks. As a result we moved to the use of existing audio libraries that abstract the audio related API and remove the above issue in their own implementation. Among the available audio engines we selected OpenAL since it provides 3D spatialized rendering functions (later, this part was replaced with our own solution), platform independence, acceptable capturing and rendering performance and – most importantly – an easy-to-understand API.

3.1.2 Multi-source Multi-sink Engine

Our system is designed with the assumption that audio streams are binaurally localized with respect to an individual listener’s location. The binaural localization process that converts incoming mono signals into stereo signals uses a Head-Related Impulse Responses (HRIR) dataset that is specifically designed for headset users. To lower the processing burden, we apply two optimization methods: downsampling and discarding less-contributing responses. The downsampling uses a 16-kHz audio sampling rate, which shortens the size of the impulse responses from 512 (for 44.1 kHz) to 186 samples. Additionally, we remove less-contributing responses, since the impulses are only effectively available within a millisecond – i.e., with an effective length of 46.
A challenge occurs when we need to support a multi-listener model on a single peer node. This means that the peer node supports the binaural localization not only for itself but also for other resource-constrained nodes. While the FMOD library supports up to four guest listeners, it does not preserve spatial cues of audio sources. Since no other audio libraries support this feature, we implemented it by ourselves.

### 3.1.3 Audio Codec

Among the available audio codecs, we preferred a lightweight, royalty-free, and high-quality voice codec. As of today, we have experimented with two voice codecs that meet our criteria: Speex from Xiph.org and SILK from Skype. Speex, popularly used in many VoIP applications such as Google Talk and Adobe Flash-based services, features variable bit rate, voice activity detection (i.e., silence suppression), discontinuous transmission, echo cancellation, noise suppression, automatic gain control, and adaptive jitter management [3]. SILK, recently released by Skype, is reported to have a higher Mean Opinion Score (MOS) than Speex and possess a better packet loss recovery scheme via forward error correction [4]. Through a series of exploratory experiments, we enabled all the main features of these codecs and confirmed their operability in our environment.

One significant disadvantage of both codecs is the lack of independent stereo coding. Although Speex supports intensity stereo coding, it cannot preserve all spatial cues when multiple speakers are involved. In response, we use a dual coding scheme that encodes and decodes the left and right channels separately, doubling the bandwidth over mono-channel coding. We plan to explore a better encoding scheme that is targeted for stereo coding of spatialized sound as part of our future work.

### 3.2 Local Peer Considerations

Several issues had to be addressed in the implementation of our audio toolkit to achieve high performance on each peer. In this section, we describe our techniques for NAT traversal, modular filter chaining, and latency optimization.

#### 3.2.1 NAT Traversal

To connect peers directly and reliably, we leveraged the open-source P2P framework libjingle. Libjingle is an implementation of Jingle, an extension of XMPP that embeds the delivery of audiovisual media data. Enabling direct connectivity between peers that use private, non-routable IP addresses is a crucial component for P2P-based applications. One of the well-known solutions is hole punching. This technique attempts to establish a communication path through the NAT device(s) (e.g., a home network router) to disguise incoming traffic as valid outgoing traffic. The mechanism is usually referred to as STUN (Simple Traversal of UDP through NAT). If a direct connection cannot be established after hole punching, an indirect connection via a third host with a public, routable IP address may provide a solution. While UDP hole punching has already been proven in many applications, a similar method is difficult to implement based on TCP. Since reliable signaling among multiple peers is crucial for the robustness of the overlay topology manager, our objective for the P2P engine is to provide both reliable TCP-based signaling and NAT-aware data channels.

These requirements can be satisfied with the open-source library libjingle. The control message exchange among peers is handled via a Jingle-compliant XMPP server reliably, while a P2P transport channel can also be established between peers. This allows several transport mechanisms for example UDP data or media stream delivery. Hence in our design, the overlay topology signaling uses an XMPP channel via a Jingle server.

#### 3.2.2 Modular Filter Chaining

Our voice application consists of a sequence of audio signal processing modules that share audio samples. To simplify this real-time task in an efficient manner, we chose to implement a filter chaining mechanism. Such a pipeline-based framework simplifies many of the intricate details and has been widely adopted for many media-processing intensive applications. The basic idea resembles the Windows' DirectShow architecture, but it is more lightweight.

A filter represents one stage in the processing of media data and can be connected to other filters. A filter chain, consisting of a sequence of filters, forms a single unit of execution. The filter uses a blocking function-call mechanism that inline executes the function in the next filter(s) in the chain, delivering media data implicitly and waiting for its return. To avoid a long blocking delay of the first filter, a subsequent filter may use a dedicated worker thread, which allows the previous processing module to return immediately.

#### 3.2.3 Low Latency Optimization

A P2P architecture has the potential to result in a lower latency than the client/server model, because each peer can deliver data directly to other peers. Even if connected via a third host (when NAT hole punching fails), it only requires two hops, which is comparable to the client-server architecture. The demerit of the P2P model is that a listener peer that is indirectly connected to a speaker via a speaker-specific multicast route—due to lack of outgoing bandwidth of the speaker—may experience a longer delay. If the auditory scene is more crowded, more users will receive audio packets late. However, the impact will be minimized as long as the packets do not contain time-related contextual cues, because the received samples will be rendered according to the receiving user's time.

Another crucial design decision is how to send packetized audio in an efficient manner. To optimize network bandwidth usage, we have decided to employ variable bit-rate audio coding and frame aggregation. When an audio frame (the minimal handling unit of an audio codec) is generated every 20 milliseconds, the length of the encoded bitstream may range from 2 to 80 bytes per frame in the case of Speex. We denote the frame byte length by \( m \), i.e., \( 2 \leq m \leq 80 \). When using RTP (the Real-Time Protocol) as the transmission format, the overhead for an audio packet consists of 40 bytes: 20 bytes for the IP header, 8 bytes for the UDP header, and 12 bytes for the RTP header.

If a packet is delivered every 20 ms, the transmission rate will be \((0.4 \times m + 16)\) kbps, where 16 kbps accounts for the header overhead. Therefore, the overhead accounts for at least one-third of the required bandwidth. If we aggregate \( x \) frames and deliver packets every \( x \times 20 \) ms, the expected bandwidth will be \( (\sum_{i=1}^{x} m_i + 40) \frac{40}{x} \) bps. Using \( m \) as the expected number of bytes per frame, the calculation can be
simplified to $0.4 \times m + \frac{11}{x}$ kbps. To compromise between the latency and the bitrate, we empirically chose $x$ to be 4.

As an alternative we also designed a more lightweight protocol since the RTP overhead still accounts for 30% of the total overhead in the example above. We eliminated unused fields in the RTP header, reducing it to just two bytes: one for time-stamping and to detect packet losses within a 20 seconds time span and the other for future extensions. As a result, the final expected transmission rate for a mono stream is $0.4 \times m + 0.8 + \frac{11}{x}$ kbps.

### 3.3 Overlay Topology Management

Some existing audio conferencing applications construct a shared application-level multicast tree and then reorganize the tree whenever a better connection is found [reference removed for review process]. This approach is successful in its goal of supporting massive numbers of concurrent users with an optimal use of network resources. However, leaf nodes inevitably experience a longer end-to-end delay. The core technology is based on employing an audio mixing technique during the delivery of the data to optimally use the bandwidth. With positional audio, however, we generally can no longer utilize audio mixing, since it does not preserve the spatial cues of the original sources. Therefore, the shared-multicast-tree based approach is less appealing when audio mixing cannot be used.

In our overlay design, every peer is assumed to maintain information about a set of visible peers and to have data-channels established with them. Then, the overlay manager constructs a depth-2 tree topology rooted at each active source node. To gain system scalability, every source selects a number of forwarding nodes randomly from a Distributed Hash Table (DHT) space and attempts to establish indirect paths to individual receivers within its hearing range to as many as possible. While the client/server-based model has a very limited selection of alternate server hosts, our randomized relay node selection mechanism has the potential to find better relay nodes that reduce transmission delays.

An overlay manager is responsible for creating connections between peers. Once outgoing paths are established, the manager continues to reorganize the overlay topology by periodically probing forwarding nodes that have unused outgoing degrees and then newly assigning them to nodes to make them reachable. The final step is to attempt to minimize the processing latency by detecting whether incoming data is silence data and quickly informing its neighbors before processing. Such early silence detection can help to reduce the processing time before delivery, hence achieving a lower latency.

### 4. PROTOTYPE IMPLEMENTATION

We implemented our scalable P2P-based audio toolkit as a Firefox browser extension, termed Spatializer (see Figure 2). In the figure, two circles in the middle window represent two remote users (i.e., interactive, live Google Talk voice sources), while two separate browser windows are rendering on-demand YouTube videos with their associated audio. When the circles are positioned within the browser (through user mouse interactions), their respective position information is passed to the Spatializer extension via Javascript calls, resulting in accordingly rendered 3D audio. The position of each audio source in every other browser window, in relation to the middle window, is computed and delivered to the extension whenever the side windows are moved. We have measured the processing delay between two directly connected peers at around 200 ms, which is very encouraging. We continue to optimize our design to further reduce the latency.

Our design is very scalable and a performance evaluation has shown that a system with an Intel Quad Core processor (Xeon E5240 at 2.50 GHz) and 8 GB of memory can handle 140 incoming audio sources per processor. This value implies that such a machine can support 10 input and 14 output streams simultaneously.

### 5. CONCLUSIONS

We present the design and implementation of a system that not only integrates interactive 3D voice communication but also unifies the spatialized 3D rendering of local and remote audio sources, using a web-based positional audio engine toolkit. Through a prototype we demonstrate the practicality and effectiveness of our approach. In addition to further encourage improvement of our toolkit, we plan to make it available to the public.

### Acknowledgement

This research has been funded in part by A*Star grant 082 101 0028. We also acknowledge the support of the NUS Interactive and Digital Media Institute (IDMI).

### REFERENCES


