SpatOSC: PROVIDING ABSTRACTION FOR THE AUTHORING OF INTERACTIVE SPATIAL AUDIO EXPERIENCES

Mike Wozniewski, Alexandre Quessy, *Zack Settel
La Société des arts technologiques [SAT]
Montréal, Québec, Canada
metalab@sat.qc.ca
*Université de Montréal, Faculté de Musique
Montréal, Québec, Canada
zack.settel@umontreal.ca

ABSTRACT

Creating an interactive project that deals with 3-D sound becomes difficult when it needs to run in multiple venues, due to a diversity of loudspeaker arrangements and spatialization systems. For effective exchange, there needs to be a standardized approach to audio spatialization with a unified, but abstract way of representing 3-D audio scenes. Formats from the gaming and multimedia communities are appealing, but generally lack the features needed for spatial audio performance and interactive new media. Experimental features need to be supported, including things like support for multiple listeners, complex sound directivity, ability to accurately model acoustics, and providing fine controls for tuning spatial effects such as Doppler shift, distance attenuation and filtering. Our solution to the problem was to develop an open source C++ library called SpatOSC, which can be included in existing visualization engines, audio development environments, and plugins for digital audio workstations. Instead of expecting that everyone adopts a new spatial audio format, our library provides an immediate solution, with “translators” that handle conversion between representations. A number of spatializers are already supported, with an extensible architecture that allows others to be developed as needed.

1. INTRODUCTION

In the domain of experimental music and interactive new media, there are no truly standardized ways of representing the spatial characteristics of sound. As a result, there exist a multitude of different audio formats, editing tools, and panning/rendering environments for 3-D audio, and it is quite difficult for a performance or installation to be transported to different venues. The traditional approach has been to use a channel-based representation, where sounds are rendered down to a number of digital audio channels that can be directly reproduced on a known loudspeaker arrangement. The most common format for this approach is 5.1 channel surround sound, which is a pantoephonic arrangement (all loudspeakers are on the same plane) only capable of reproducing horizontal spatial audio effects. Although formats exit for periphonic (3-D) arrangements, like the 22.2 channel system for ultrahigh-definition TV [9], the general drawback of channel-based representations is that there is no flexibility for experimenting with loudspeaker placement.

The other major drawback is the lack of interactive control, since sounds that are already encoded in raw audio channels cannot be separated, and changing the panning is very difficult. This makes such an approach only effective for fixed media production (i.e., non-realtime playback systems) with an authoring stage that is far removed from the presentation of the work. For interactive performances and installations, where sound movement is controlled in real time, an object-based representation is more effective [7]. Instead of raw audio signals, sounds are described as virtual sound sources in 3-D space, and their positions can be manipulated in real time.

This approach also tends to be more efficient in terms of storage and transfer of information, since a sound is represented with only one audio signal and some low-bandwidth control data (instead of one signal per loudspeaker). This is an important consideration for new immersive performance environments which potentially have hundreds of loudspeakers. Examples include the AlloSphere1 in Santa Barbara, the Klangdom2 in Karlsruhe, and the Satosphère3 in Montreal.

While object-based representations offer more flexibility in terms of interactivity, they too come with certain challenges and drawbacks. An application known as a “spatializer” must be able to take the virtual audio scene and render corresponding audio signals for a particular loudspeaker arrangement. This of course must be done in real time, which can take significant computational resources for large systems.

Given that there is also no standard way of describing a spatial audio scene, a variety of spatializers and spatialization techniques exist, each with different representations and constraints. While some spatializers are satisfied with...
with simple 3-D coordinates \((x, y, z)\) for a sound source, others require more complex modelling. Representations of musical instruments for instance, require complex directivity patterns where sound radiates differently in different directions from the sound source. Also, room properties such as size, material of the walls, and sound-reflecting obstacles will significantly affect the resulting sound field. Some spatializers require that all loudspeakers are evenly spaced, and equidistant from the listener, while others offer greater flexibility, to the point where loudspeakers may even be modelled with 3-D positions and updated in real time.

In recent years there have been efforts to develop an interchange format for these object-based audio representations. The SpatDIF format \(^{13}\) is a notable example, first presented at ICMC in 2008, and provides an extensible format in order to support the more experimentation features required by some systems. However, SpatDIF is just a format, requiring that control applications and spatializers implement the proper mechanisms to read/write or send/receive the appropriate messages. While adopted by some researchers of computer music, it will take time and effort to overcome the inertia that is typical of the professional audio production industry.

We have realized that a worthwhile goal in the interim is to develop a number of translation algorithms, capable of converting one sound scene representation to another. As a result, we have created and published a low-level open source library that can be included in a variety of computer software, and can automatically convert and send spatialization control messages to several common sound spatialization engines. The library, called SpatOSC, is extensible, so for each new spatialization system, one just needs to add a new translator (a rendering plugin) to the library. Communication with a spatializer is done using network protocols such as Pure Data’s FUDI or OpenSoundControl (OSC), which we have found to be supported in most modern spatializers.

2. BACKGROUND & RELATED WORK

The goal of sound spatialization is to render sound for a listener such that the source is perceived to originate in the 3-D space surrounding the listener. This problem can be approached in many ways depending on the audio display being used (i.e., the number of loudspeakers available and their positions relative to the listener).

2.1. Spatialization Techniques

Binaural methods aim to reproduce the interaural timing and intensity differences heard by human ears. By filtering a signal with an HRTF (head-related transfer function), signals for each ear can be obtained that simulate sound sources originating from arbitrary directions \(^6\). This approach is typically used with headphones, but binaural techniques do exist for loudspeakers too. Amplitude and timing differences between stereo loudspeakers can be used to affect the apparent location of a sound source, and crosstalk cancellation can ensure that each ear receives a unique sound signal \(^5\).

When using more than two speakers, a number of amplitude differencing methods may be used. Vector base amplitude panning (VBAP) \(^17\) is a perhaps the most common, in which the speakers are grouped into triplets and each sound source is rendered using only three loudspeakers rather than the whole array.

Ambisonics on the other hand is a technique that provides amplitude panning for all loudspeakers in the array. The ambisonics B-format \(^8\) encodes audio signals with directional components \((w,x,y,z)\), where \(w\) is the non-directional gain component, and \((x,y,z)\) are the gains in each of the three directions. Decoders exist that reproduce the appropriate loudspeaker signals using a linear combination of these four channels, for any position in the real space. Higher-order ambisonics use more channels, and can thus be used to more accurately represent the spatial characteristics of a sound in 3-D space.

Both ambisonics and VBAP are meant to be deployed on loudspeakers that are uniformly spaced around the user, providing accurate spatialization at the convergence point, or “sweet spot”. Wave Field Synthesis (WFS) \(^1\) on the other hand attempts to create a sound field throughout an entire volumetric space. This requires a large array of small speakers and is computationally expensive compared with the aforementioned techniques. The concept of WFS is based on Huygen’s principle, which states that a wavefront can be seen as the composition of elementary waves, each of which propagate with the same velocity and wavelength as the original wave. Each small loudspeaker is thus responsible for generating one of these elementary waves, which combines with others to re-create the true wavefront of a particular sound experience.

There are other implementations that render sound at arbitrary spatial positions using a virtual microphone approach \(^2\) \(^12\) \(^14\). Mainly based in audio engineering,
these implementations tend to focus on acoustic space simulation and typically use pre-recorded material with an acoustical model of the recording space to resynthesize a signal at any given position. There are also a number of spatializers that are heuristic-based, simulating the most important psychoacoustic effects related to the perceptual experience of 3-D sound. Examples include Spat~ developed at IRCAM [11, 10] and the Space Unit Generator originally found in the cmusic sound synthesis program in 1982 [25].

2.2. SpatDIF and the ICMC 2008 panel

At the International Computer Music Conference (ICMC) in 2008, researchers from around the world met to discuss interchange formats for spatial audio scenes [22]. The main development of this panel was the presentation of SpatDIF, a spatial sound description interchange format [13]. The format uses OpenSoundControl (OSC) messages to communicate both the general properties of the scene (coordinate system, units, etc.) and control messages for dynamically changing sound source positions. The assumption is that there is a lowest common denominator for describing an audio scene that is common to all spatializers. Any SpatDIF compliant renderer should be able to understand these “Core Descriptors” and produce spatial sound to match the desired effect.

A controller or editor application thus only ever needs to know one format or language, and sends messages without needing to know anything about the output system in advance. For more complex behaviours, SpatDIF has a number of “extensions”, to specify things like sound source directivity, acoustic spaces, distance attenuation, trajectories, doppler, etc. Renderers that support these extensions will understand how to deal with these messages, while simpler systems will simply ignore them.

From the perspective of the spatializer, one needs to implement a “SpatDIF Interpreter”, which receives those OSC messages and adjusts the rendering process accordingly. There are already implementations of such interpreters in Jamoma [14], ICST’s Ambisonics Tools [20], and GMEM HoloEdit [16]. A standard format however puts the burden on all other spatializers to adopt the format and implement an interpreter. This is a daunting task, seeing as many high-end spatializers already have an OSC control system that allows external control of spatialization parameters. Examples like D-Mitri from Meyer Sound, Spatialisateur from IRCAM, and Zirkonium from ZKM already have many users and existing tools that work with these systems, creating lots of inertia to change. It would obviously be possible to create an external interpreter, which simply receives OSC in SpatDIF format, converts it to a given spatializer’s OSC format, and sends it onward. However, such an approach introduces extra delay, and an extra process that must be managed.

We believe that a better solution is to introduce the translation on the sender side. We assume that spatializers have their own custom formats that are resistant to change, and provide a library which knows all those formats. The controller or editor application can link with this library and use a single API for communicating with any spatializer that is known by the library. This does imply that the type of spatializer is known, which is not the case in SpatDIF, where a control application may be completely unaware of the output technology.

2.3. A solution for digital audio workstations (DAWs)

Another important development at the ICMC panel session in 2008 was the realization that better spatialization tools were needed for DAWs. Many of the audience members, and indeed many people working with spatialization, come from a sound engineering background and have limited programming knowledge. Most spatializers on the other hand require the use of audio programming environments like Pure Data, SuperCollider, or Max/MSP. The SpatOSC library offers a solution in that the majority of DAW plugins (VSTs, Audio Units, and LADSPA plugins) are written in C++ or Objective-C, and can be easily linked with the SpatOSC library. An audio unit is already available for DAWs running on OSX; see Section 3.7 for more information.

2.4. Other Formats

In addition to SpatDIF, there are a few other formats dedicated to the description of spatial audio content, yet most tend to be insufficient for experimental audio/music experiences. The X3D format [15], which has succeeded VRML as the open standard for describing 3-D content, has seen a lot of attention as WebGL and the X3DOM are finding universal support in major browsers. Its extensible architecture makes it a strong contender for support in spatializers, where even SpatDIF could be added as an extension and parsed out by any applications who choose to support it. However, X3D does have its own audio format, which supports source sound directivity (see Section ??) and scheduling of soundfile playback. Generally, only one listener is supported and the format tends to describe static scenes where all sounds need to be defined in advance. It would be difficult to use this format for live interactive performances, as no direct specification for live inputs is provided.

Another noteworthy format is AudioBIFS from MPEG-4 [21]. BIFS, which stands for Binary Format for Scene Description, is in an extension of X3D with a focus on describing audiovisual scenes in a compact and object-oriented fashion that ultimately leads to streamable interactive content. A scenegraph hierarchy is used to organize nodes, and specialized nodes for audio processing are defined, such as: ‘Sound’, ‘AudioSource’, ‘AudioFX’, and ‘ListeningPoint’. Recently, ‘DirectiveSound’ has been added, allowing for directional sound sources, as well as ‘WideSound’ which provides a description for a sound source that occupies a larger volume. Likewise, additional parameters for modelling acoustics and acoustic
materials have been added [23]. BIFS are always binary in format, hence they must be authored separately and then compiled for distribution. Interactivity is accomplished by allowing certain fields to be exposed and updated by the content server, but this control is not implicit. Rather, the developer must pre-define all possible interactions during the authoring phase, which limits the possibility for live experimentation.

There is also ASDF [7], which contrary to the previous formats, is primarily intended for musical purposes. It is an XML-based extension to SMIL (Synchronized Multimedia Integration Language), which focuses only on temporal control and synchronization of audiovisual content. ASDF adds spatial positioning to that format and although it still lacks a lot of features, the attention to timing and synchronization is important for musical composition. In contrast, formats like X3D provide little or no means to reorganize sequences of events, since timing is abstracted through routings between scripts and processes.

2.5. Existing Libraries

For software developers working in the fields of 3-D games and virtual reality, there do exist a few of libraries to manage virtual sound source simulation. Examples like FMOD, Wwise, irrKlang, DirectX and OpenAL provide realistic real-time rendering of sound sources, and are easily integrated into existing development environments. Unfortunately, these libraries are not really geared towards musical or highly interactive control of sound synthesis, hence they often have an impoverished audio representation. For instance, most APIs have no method to specify directivity of sounds and consider all sources as omni-directional. In the cases where directional sounds are supported, these are typically implemented with linear attenuation applied as a function of angle. There is usually no support for the complex radiation patterns of traditional musical instruments, or the cardioids that are commonly found in audio equipment. Furthermore, there is often only support for a single listener and standard audio formats (e.g. stereo or 5.1 channel surround). Arbitrary speaker configurations are rarely supported, and the listener is usually assumed to be wearing headphones or sitting in a centralized sweet spot that is equidistant from all speakers.

In most cases, these APIs also provide hardware abstraction and directly send computed sound signals to the sound card, which may be desirable for a game programmer, but limiting to a sound artist. Our approach instead assumes that the audio hardware is controlled by a separate process, often on a separate machine, and uses OSC to send spatialization parameters.

3. THE SPATOSC LIBRARY

A system that uses SpatOSC is composed of a host application (typically written in C, C++ or Objective-C), and an audio renderer (which provides physical output to loudspeakers). These are independent components, which may or may not be located on separate machines, and which communicate via OpenSoundControl (OSC) in order to update spatialization parameters.

Figure 2 gives an overview of how SpatOSC integrates with the application and audio spatializer. This example is typical of most virtual reality systems or 3-D audiovisual installations, where the host might be a plugin for a DAW, an external for Pd or Max/MSP, or any game engine or visualization software built using libraries like Ogre, OpenSceneGraph, or Unity 3D. The host application maintains the state of a virtual scene, including spatial information about audio-related items. Specifically, it maintains a list of sound sources (which emit sound into the scene) and listeners (which capture sound at a particular location). The SpatOSC library provides an API to the application that allows for the creation and updating of these nodes. Whenever a 3-D sound source is moved, a function call is made to SpatOSC and the state within the library is updated.

3.1. SpatOSC Internal Representation

Internally, SpatOSC maintains its own representation of the scene, with a structure that allows for easy conversion between the formats needed by the spatialization techniques described in Section 2.1. At the base level, there are the common elements to all spatializers: sound sources and listeners. But contrary to most approaches, SpatOSC offers greater flexibility by supporting multiple listeners, so that a single scene can be shared among multiple participants.

A listener may be thought of as the virtual representation of a loudspeaker or group of loudspeakers. In the case of something like a 5.1 channel surround system, all loudspeakers combine to simulate the sound field at one location in the scene, so they are grouped together and thought of as one listener. However, there may be cases where loudspeakers actually move during a performance, like when multiple sets of headphones are tracked with a motion tracking system. In such a case, multiple listeners need to be defined. Another example is an installation without a centralized sweet spot, such as a long sound wall, where each loudspeaker provides a localized rendering of a certain portion of the wall. In such a case, a listener would need to be defined for each loudspeaker.

Sound sources on the other hand are generative, and emit sound energy into the virtual scene at particular locations. Their signals may derive from pre-recorded sound files, remote streams from other locations, realtime sound synthesis generation, or live input signals from audio hardware or software busses.

The following properties may be associated with sound sources and listeners:

Spatial Pose: All nodes (sources and listeners) have 6 degrees of freedom: they are positionable in cartesian (x, y, z) space, and they have a 3-axis orientation which can be described either by a direction vector (a unit vector in the local coordinate space of the node), Euler angles (pitch, roll, yaw), or quaternions. Positions are always defined relative
Figure 2. Overview of SpatOSC and its integration into 3rd party packages

3.1. Representations

RADIUS: A radius parameter allows a sound to occupy a volumetric region instead of acting as an infinitesimally small point source. If a listener node enters within this radius, all spatial effects are disabled and the sound plays naturally, with unity gain and no filtering. A similar parameter is often found in sound panning systems under names like spread, diffusion, blur, etc. By default, the radius for all nodes is set to zero.

URI: For sound sources, the URI describes the media type and location emitted at that location. This could be a simple sound file reference (file://loop.wav), a live input channel on the soundcard (adc://1), a stream (http://stream.mp3), or a plugin (plugin://looper~.pd).

EXTENSIONS: Each node can also be extended with arbitrary key-value parameter pairs that describe additional features that may be required by specialized systems.

3.2. Connections

One of the biggest differences between SpatOSC and other spatial audio representations is that sound sources are not necessarily connected to every listener, and there is an explicit connection structure that must be made. The CONNECTION represents both the logical transfer of sound from one location to another, but also the physical modelling effects involved in the transfer. For example, the CONNECTION contains information about distance, attenuation of gain resulting from attenuation effects, the time delay incurred for sound to travel that distance, and other physically-modelled properties. Some VBAP-style renderers may only do panning, and not have any built-in computation for distance effects. In such a case, it would still be possible to use this information (connection gain, variable delay) from SpatOSC to pre-process the signals before they are sent to the panner.

Other benefits of maintaining explicit spatial audio connection include the ability to temporarily disable sections of the scene, or to provide specific / unique sounds to some listeners without others hearing them (even if they are close by). However, the main benefit is the ability to customize spatial effects for some connections differently than others, as described in the next section.

3.3. Manipulation of connection effects

As sound travels between different nodes in the scene, SpatOSC computes a few simple parameters related to audio propagation. This includes the modelling of sound decay as a function of distance, delay of sound resulting from travel time between source to listener, and also some filter coefficients that may be used to simulate spatial effect.

The CONNECTION object provides the ability to tune (enable, disable, scale) several spatial effects, and thus provides fine grained control over spatialization that is not typically possible with other libraries. The possible tuning factors are:

DISTANCE FACTOR: Specifies the extent to which distance attenuation and absorption effects should be applied.

DIRECTIVITY FACTOR: Provides a parameter to scale the effect of sound directivity.

The filter coefficients require the spatializer to pre-process a signal before spatialization. For example, a lowpass filter coefficient is provided to approximate the absorption of high frequencies as the sound travels longer distances.
**DOPPLER_FACTOR:** Allows for Doppler shift to be minimized or emphasized.

Consider the following concrete examples: some sound sources in a scene may provide more of a narrative / global voice, and should not be perceived as originating from a particular direction. In such a case, localization or panning effects could be diminished, but the gain could still attenuate as the listener leaves a specific area. Another example may be the requirement for Doppler shift to be diminished for a sound source containing musical material, in order to preserve timing and intonation in musical contexts. Conversely, Doppler shift is often emphasized in sound tracks for cinema because it adds dramatic effect. The **DOPPLER_FACTOR** parameter of the connection allows for a scaling of the effect both for emphasis and minimization.

### 3.4. Internal Conversions & Computations

Given that SpatOSC needs to be able to supply a number of different parameters to different types of spatializers, we often need to convert, scale, or apply transformations to the internal data. Consider for example, the gain attenuation of a signal as a result of distance. Amplitude panning systems (VBAP, etc.) do not support simulation of distance effects, so for convenience, SpatOSC computes a gain coefficient for each connection, using the following formula:

\[
g = \frac{1}{(1 + |B - A|)^\beta}
\]

This gain value is always in the range of [0,1], and represents the amount by which the amplitude of the signal should be scaled to approximate the decay of sound due to distance. The additional control parameter, \(\beta\), which we call the **DOPPLER_FACTOR**, helps to control the steepness of the exponential decay. When \(\beta = 1\), it results in an exponent of 2.0, which is identical to the inverse square law that sound observes in nature. Meanwhile, a value of \(\beta = 0\) effectively cancels the affect of distance decay since no matter what the distance may be, the gain will always be unity (1.0). Lastly, \(\beta > 1\) can be used for even sharper decay than what is common in nature, allowing for very localized sounds.

There are also helper functions available to convert between units. For instance, one might want to know the gain value in decibels rather than amplitude gain, or the position of a sound source in radial units (AED: azimuth, elevation, distance) instead of cartesian units (x, y, z). One might also require sound source positions to be relative to a listener instead of the global coordinate system. In such a case, the **CONNECTION** object can be queried and the relative radial or cartesian coordinates can be retrieved.

Conversions between coordinate systems are also supported. SpatOSC provides transformations for the entire scene that allows conversion between right- and left-handed coordinate systems, flipping axes, and rotating the global coordinate system.

### 3.5. Directivity

The representation of sound source directivity is potentially complex, and not surprisingly, one of the least standardized aspects of spatial audio scene description. In fact, most spatializers only support omnidirectional sound sources, while others represent directivity patterns with parametric functions or simplified models such as sound cones or ellipsoids. The goal of SpatOSC is to be able to store directivity in such a form that it is convertible into any of these other formats.

Examining formats like X3D [15], we find directivity described with two concentric ellipsoids. When a sound is heard within the inner ellipsoid, the signal is unmodified, then decays linearly from the boundary of the inner to the outer ellipsoid. Two parameters control each ellipsoid, defining how far it stretches to the front and back. DirectX and OpenAL have similar representations, except that they use concentric cones. The signal is unmodified within an inner cone, and decays linearly to an outer cone.

Other methods describe directivity with just one parameter. For example, Space Unit Generator [25] has a parameter called back, which is omnidirectional (provides no backward attenuation) with a value of 0, and transforms into a cardioid and eventually a hyper-cardioid as the value reaches 1.0 (where there is full backward attenuation. IRCAM’s Spatialisateur has a similar parameter, called aperture.

In SpatOSC, we use attenuation tables that specify sound intensity at different angles from the sound source’s direction vector. Such a representation can be effectively used to describe all of the above cases. Similar tables are used in implementations like CATT [6] and ViMiC [2]. The premise is that a complex directivity pattern can be sampled, attenuation values can be specified for particular angles, and linear interpolation can be used in between. Some table-based implementations assume radial symmetry, meaning that the radiation is the same in all directions (up to 180 degrees), while others like OpenDAFF [7] describe attenuation points on an equiangular spaced sphere grid.

For SpatOSC, we use axisymmetric attenuation tables for both the horizontal and vertical directions, meaning there are two attenuation tables representing orthogonal angles from 0 to 180 degrees. We use variable sampling instead of fixed sampling, allowing complex patterns to be described with more detail while an omnidirectional source can be defined with a single table entry of 1.0 (instead of repeating the value many times).

![Source Directivity](image)

Figure 3. The directivity of a sound source is defined by a direction vector and sampled attenuation values at various angles (stored in a table).

Figure 3 shows an illustration of a source signal that radiates with unity gain along its direction vector and exhibits varying levels of gain attenuation at different angles of incidence, \(\alpha\), away from the direction vector. In this case, the directivity has a cardioid shape, which is easily defined with a mathematical function [8].

---

[17] Note that \(\gamma = 1.0\) produces a normal cardioid, \(\gamma = 0\) flattens the shape resulting in omni-directional radiation, and \(\gamma > 1.0\) results in a...
\[ f(\alpha) = \left( \frac{1 + \cos(\alpha)}{2} \right)^\gamma \] (2)

We can sample this function (with \( \gamma = 1 \)) every few degrees and store the values in an attenuation table for simple lookup during runtime. However, more abstract directivity patterns that are not easily represented by mathematical functions can also be stored.

During runtime, the \( \gamma \) parameter, which we call the Directivity Factor, can be simulated by a scaled reading from the attenuation table. When \( \gamma = 0 \), we read only the first element from the table, reducing the effect of directivity. Then we read exponentially faster through the table as \( \gamma \) increases, such that when \( \gamma = 1 \) the reading becomes perfectly linear, and tends towards hyper-directivity with higher values. Of course, this only works when directivity functions are monotonically decreasing from the direction vector.

### 3.6. Translators

One of the most important aspects of SpatOSC is the translator system. One can think of a translator as plugins for the library, which allow for communication with various third-party audio spatializers. A user chooses which translator to use depending on their system configuration, and typically only needs to specify the remote hostname and port to which OSC messages are sent.

Translators are used with “lazy loading”, meaning that they are only loaded at runtime when the user requests a particular translator. This allows developers to easily create new translators for new pieces of hardware and software, or even for a particularly esoteric performance or installation.

On the implementation level, the developer needs to define how a translator responds to notifications from the SpatOSC scheduler. A list is provided of all nodes and connections that have changed, and the developer must send updates in the appropriate format to the remote spatializer. This means that the translator must extract what it needs from the internal scene representation, use any of the conversion functions provided, and send OSC messages. In some cases, only relative positions or angles will be sent, while in other cases, delay times and filter coefficients might be included as well. It all depends on the requirements and abilities of the remote technology.

In the event that the internal representation does not provide adequate information, the translator may wish to use the SpatOSC’s node extensions (see Section 3.1), which implies that the host application needs to set these extra key-value parameters for each node, as needed by the translator. To provide a concrete example, let us consider Zirkonium [13] (used in the Klangdom; see Figure 1), which is a VBAP-style spatializer that just does panning with no distance effects or source directivity. However, they provide two parameters: “azimuth span” and “zenith span”, which allow the panning of a sound source to be stretched in either the horizontal or vertical direction. While this resembles our directivity attenuation tables, it is not analogous. For instance, the effect of fully open azimuth span and no zenith span creates a “ring” of sound on a horizontal plane, meaning that sound is emitted uniformly from all speakers at a certain height. This situation has no meaningful 3-D representation in hyper-cardioid. This results in a single parameter in order to change from an omni-directional to highly directional sound source.

SpatOSC provides all handling of networking, creation of sockets, including support for multicast, and the ability to send to several translators simultaneously.
The scheduler allows for the management of network load, but can also be used to balance the computational load on the spatialization system. Each update will potentially cause a recomputation of several filters or DSP processes, which consume valuable CPU resources on the rendering machine. However, care must be taken, because a delay of spatialization events may lead to a degradation of perceived spatial cues. Studies have shown that latencies above 70ms start to decrease an individual’s ability to localize sounds.

4. CONCLUSION & DISCUSSION

We have discussed the need for an abstract audio scene description mechanism, catered to musicians and artists. By exploring the diversity and complexity of various spatialization techniques and implementations, we note that it is a difficult task to create a standard spatial audio format. Instead, we have developed an open source software library with an extensible architecture based on translator plugins, which can accommodate a range of technologies. Networking via OpenSoundControl allows for a separation of tasks, letting spatializers do what they do best, while providing flexibility for the integration of spatial audio control into a number of different host applications. We have written unit tests, provided code examples, and deployed artworks with interoperability between multiple spatializers and loudspeaker layouts.

Further work is needed to effectively deal with acoustic simulation, since SpatOSC currently has no way to model the environmental effects of a 3-D scene. This is a major obstacle to achieving realistic spatial audio effects. Further attention also needs to be paid to sequencing or timed playback of material. The Audio Unit described in Section 4.2 is currently the only way to sequence or compose with SpatOSC.

However, the utility of SpatOSC for real-time interactive work is clear. We have already created audio installations which easily play on multiple types of spatialization hardware, and have bypassed the limitations of simple panning systems using the extra computations provided by SpatOSC’s connection models. As more translators get written, this will allow for greater interchange between systems, and hopefully lead to more experimentation with 3-D sound throughout the artistic community.

5. ACKNOWLEDGMENTS

Omitted for blind review

6. REFERENCES


