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

Mike Wozniewski, Zack Settel[†]*, Alexandre Quessy, Tristan Matthews, Luc Courchesne*

La Société des arts technologiques [SAT]
Montréal, Québec, Canada

[†]Université de Montréal, Faculté de Musique
Montréal, Québec, Canada

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 potential 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 by artists working with interactive new media. Non-standard features need to be supported, including support for multiple listeners, complex sound directivity, integration with networked show control and sound synthesis applications, as well as support for the tuning spatial effects such as Doppler shift, distance attenuation and filtering. Our proposed solution is an open source C++ library called *SpatOSC*,[1] which can be included in existing visualization engines, audio development environments, and plugins for digital audio workstations. Instead of expecting that everyone adopt a new spatial audio format, our library provides an immediate solution, with "translators" that handle conversion between representations. A number of translators are already supported, with an extensible architecture that allows others to be developed as needed.

1. INTRODUCTION

In the domain of 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 adapted 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 pantophonic 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 [4], the general drawback of channel-based representations is that there is no flexibility for experimenting with loudspeaker placement.

The other major drawback is the difficulty of interactive control required by artists, since sounds that are already encoded in raw audio channels cannot be separated. Ambisonic formats [3] are perhaps an exception to this rule, offering a channel-based representation that can be decoded and interactively panned. However, a majority of surround formats are only effective for fixed media production (i.e., non-realtime playback systems) with an authoring stage that is separate from the presentation of the work. For interactive art performances and installations, where sound movement is controlled in real time, an *object-based* representation is more effective [2]. 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, providing artists with new ways to think about, and work with sound.

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 render the virtual audio scene into 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.

There are several computation techniques for such rendering [1]. Binaural systems can render a scene for headphones using filters that are ideally tuned to a listener's head shape. Vector base amplitude panning (VBAP) can produce fairly accurate rendering in the middle ("sweet spot") of equidistantly spaced loudspeakers. Conversely, wave field synthesis (WFS) can provide volumetric simulation of a sound field with no sweet spot but requires a huge number of loudspeakers and thus significant computational resources.

Given this variety and the fact that there is also no standard way of describing a spatial audio scene, a variety of spatializers exist, each with different representations and constraints. Some spatializers only perform *panning*, adjusting loudspeaker amplitudes to give sounds directionality but do not render depth or distance effects. Oth-

[1]`http://code.sat.qc.ca/projects/spatosc`

ers allow full 3-D (x, y, z) placement of a sound source, and even a representation of how sound radiates from that location. Representations of musical instruments for instance, require complex directivity patterns where sound propagates non-uniformly from the sound source. Also, room properties such as size, wall material, and sound-reflecting obstacles may or may not be represented, significantly affecting the rendering.

For artists working with spatial audio in more experimental or demanding contexts, a sound processing environment (SPE) is often used. Examples such as Max/MSP, Pure Data, and Supercollider can handle both spatialization *and* integrated object-based signal processing. Programming interactivity in these environments allows artists to create custom sound generation modules that are dynamically controlled and rendered by an appropriate spatializer in real time.

Works created in such a fashion are however difficult to transfer to new hardware configurations. The chosen spatializer may not support the new loudspeaker arrangement, or a different (possibly proprietary) spatialization system may be required for a particular venue. To facilitate interchange and the ability to render spatial audio in different contexts, we have created and published a low-level open source library that can be included in a variety of computer software. The library, called *SpatOSC*, can automatically convert and send control messages to several common sound spatialization engines. It is extensible, so for each new spatialization system, or sound processing environment, one just needs to add a new translator (a rendering plugin) to the library. Communication with a spatializer (or SPE) is done using OpenSoundControl (OSC), which we have found to be supported in most modern spatializers and SPEs.

2. BACKGROUND & RELATED WORK

At the International Computer Music Conference (ICMC) in 2008, researchers from around the world met to discuss interchange formats for spatial audio scenes [9]. The main development of this panel was the presentation of *SpatDIF*, a spatial sound description interchange format [5]. The format uses 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.

However, SpatDIF is just a format, requiring that control applications and spatializers implement a "SpatDIF Interpreter" to read/write or send/receive appropriate messages. While there are a few implementations (e.g., Jamoma,[2] ICST's Ambisonics Tools[3]), *ALL* other spatializers will need to adopt the format in order to become a successful standard. What makes this difficult is that many high-end spatializers already have an OSC control system that allows external control of 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 substantial inertia to change.

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, so we provide a library that can learn 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.

2.1. 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 sound 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.

2.2. 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 interactive new media experiences. The X3D format [6], which has succeeded VRML as the open standard for describing 3-D content, has received a lot of attention as WebGL and the X3DOM are finding universal support in major web browsers. The format has a default audio format that supports sound source directivity (see Section 3.5) 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, but X3D is an ex-

[2] `jamoma.org`
[3] `icst.net/research/projects/ambisonics-tools`

tensible format and extensions could be written to support these features.

In fact, the AudioBIFS format from MPEG-4 [8] 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 scene graph 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, parameters for modelling acoustics and acoustic materials have been added [10]. BIFS (which stands for Binary Format for Scene Description) are binary, and 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.

In contrast to the previously mentioned formats, ASDF [2] is primarily intended for musical purposes. It is an XML-based extension to SMIL (Synchronized Multimedia Integration Language), which focuses 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.

2.3. Existing Libraries

For software developers working in the fields of 3-D games and virtual reality, there exist several libraries that 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, and it is difficult to combine these libraries with external synthesizers or programming environments like Max/MSP, SuperCollider, or Pure Data. Many APIs also 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 that OSC is used to send spatialization parameters.

3. THE SPATOSC LIBRARY

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

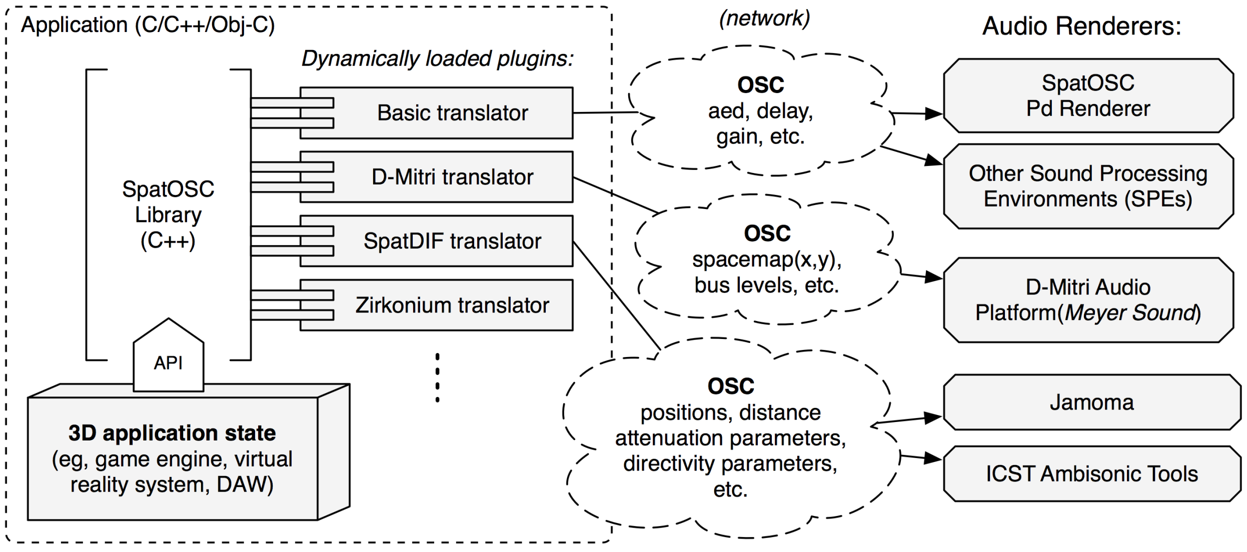
Figure 1 gives an overview of how SpatOSC integrates within an interactive system. This example is typical of most virtual reality systems or 3-D audiovisual installations, where the host might be a plugin for a DAW, an external for Pd or Max/MSP, or any game engine or visualization software built using libraries like Ogre, OpenSceneGraph, or Unity 3D. The host application maintains the state of a virtual scene, including spatial information about audio-related items. 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.

One of the most important aspects of SpatOSC is the *translator* system (see Section 3.6). One can think of a translator as a plugin for the library, which allows for communication with a particular third-party audio spatializer. 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.

3.1. SpatOSC Internal Representation

Internally, SpatOSC maintains its own representation of the scene, with a simplified audio scene description containing elements common to all spatializers: sound sources (which emit sound into the scene) and listeners (which capture sound at a particular location).

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



Application (C/C++/Obj-C)
Dynamically loaded plugins:
SpatOSC Library (C++)
Basic translator
D-Mitri translator
SpatDIF translator
Zirkonium translator
API
3D application state
(eg, game engine, virtual reality system, DAW)
(network)
OSC
aed, delay, gain, etc.
OSC
spacemap(x,y), bus levels, etc.
OSC
positions, distance attenuation parameters, directivity parameters, etc.
Audio Renderers:
SpatOSC Pd Renderer
Other Sound Processing Environments (SPEs)
D-Mitri Audio Platform(Meyer Sound)
Jamoma
ICST Ambisonic Tools

Figure 1. Overview of SpatOSC and its integration into 3rd party packages

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, realtime sound synthesis, or live input signals from audio hardware or software busses.

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

SPATIAL POSE: Both sources and listeners have 6 degrees of freedom: they are positionable in cartesian (x, y, z) space, and they have an orientation which can be described either by a *direction vector*, Euler angles (pitch, roll, yaw), or quaternions. Positions are always defined relative to a global coordinate system.

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 class that is used to describe both the logical transfer of sound from one location to another, and also the physical modelling effects involved in the transfer. The CONNECTION contains information about distance, attenuation of gain resulting from sound field propagation, the time delay incurred for sound to travel that distance, and other physically-modelled properties.

A benefit of maintaining explicit connections includes the ability to temporarily disable sections of the scene, or to provide 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.

3.3. Manipulation of connection effects

In order to provide artists with flexibility for experimentation and non-standard audio scene interaction, the CONNECTION object provides the ability to tune (enable, disable, scale) several spatial effects, and thus provides fine grained controls for spatialization that is not typically possible with other libraries. The DISTANCEFACTOR specifies the extent to which distance attenuation and absorption effects should be applied. The DIRECTIVITYFACTOR provides a parameter to scale the effect of sound directivity. And the DOPPLERFACTOR allows for Doppler shift to be minimized or emphasized, which is particularly useful in order to preserve timing and intonation in musical contexts.

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.) may only do panning and do not support simulation of distance effects. For convenience, SpatOSC computes a gain coefficient for each connection using the following formula (where **A** and **B** are the positions of the source and listener):

$$g = \frac{1}{(1+|\mathbf{B}-\mathbf{A}|)^{\frac{\beta}{0.5}}} \qquad (1)$$

The result, in the range of [0,1], represents the amount by which the amplitude of the signal should be scaled to approximate the decay of sound due to distance. The DISTANCEFACTOR, β, helps to control the steepness of the exponential decay. With this parameter, one may gradually transition from zero distance attenuation ($\beta = 0$) to the *inverse square law* that sound intensity observes in nature ($\beta = 1$), and even beyond in order to create hyper-localized sounds ($\beta > 1$).

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 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.

For SpatOSC, we use axisymmetric attenuation tables that specify sound intensity at different angles from the sound source's direction vector. These tables are used 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).

Figure 2 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, α, away from the direction vector. In this case, the directivity has a cardioid shape, which is easily

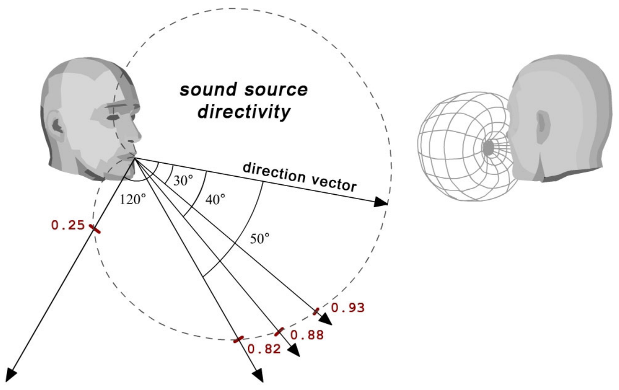


Figure 2. The directivity of a sound source is defined by a direction vector and sampled attenuation values at various angles (stored in a table).

defined with a mathematical function:[4]

$$f(\alpha) = \left[\frac{(1+\cos(\alpha))}{2}\right]^{\gamma} \qquad (2)$$

We can sample this function (with $\gamma = 1$) every few degrees and store the values in the 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 DIRECTIVITYFACTOR parameter γ can be used to apply 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 γ 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

SpatOSC's internal state can be converted for use in SPEs such as Pure Data, and various third-party audio spatializers via the use of spatializer-specific "translators." These are implemented as plugins with "lazy loading," so that they are only loaded at runtime when the user requests a particular translator. Developers can thus 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 of a translator needs to define how a translator responds to notifications from the SpatOSC scheduler. The translator must be coded so that it extracts what it needs from the internal scene representation, using any of the conversion functions provided, and sends OSC messages to its remote

[4]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 hyper-cardioid. This results in a single parameter in order to change from an omni-directional to highly directional sound source.

spatializer.[5] 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 SpatOSC's node extensions (see Section 3.1) to set these extra key-value parameters for each node. To provide a concrete example, let us consider Zirkonium [7], which is a VBAP-syle spatializer that only performs panning with no distance effects or source directivity. Two parameters: "azimuth span" and "zenith span," 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 SpatOSC, so it is useful to use extension properties to store the azimuth and zenith span values for every node. The Zirkonium translator thus knows how to deal with those specifically-named properties, while other translators simply ignore these parameters.

It should also be noted that multiple translators can be assigned at once, making it possible to simultaneously render the scene on different spatializers. This is useful in situations where multiple listeners are exploring the environment.

3.7. The SpatOSC Audio Unit plugin

Given that the SpatOSC library is written in C++, it is generally quite easy to include the library within a plugin for a DAW. As of writing, only an Audio Unit (for OSX) has been developed, but the intention is to also develop VST (Windows) and LADSPA (Linux) plugins.

The Audio Unit can be added to a track in a composition, and provides parameters and a graphical user interface (GUI) for moving sounds. In the most common setup, that track's signal output is sent directly to an input channel on the spatializer while OSC messages send corresponding control signals as to how that input should be processed. SpatOSC has no methods for storing trajectories or sequencing spatial parameters, so one of the best benefits of using SpatOSC within a DAW is the built-in automation system used to record and playback trajectories for moving sounds.

4. CONCLUSION & DISCUSSION

We have discussed the need for an abstract audio scene description mechanism, geared towards 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.

SpatOSC currently has no way to model the environmental effects of a 3-D scene; further work in this area is needed in order to achieve realistic spatial audio effects. Further attention also needs to be paid to sequencing or timed playback of material.

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. As more translators are written, interchange between systems will increase, hopefully leading to more experimentation with 3-D sound throughout the artistic community.

[5]SpatOSC provides all handling of networking, creation of sockets, including support for multicast, and the ability to send to several translators simultaneously.

5. REFERENCES

[1] D. R. Begault, *3-D sound for virtual reality and multimedia.* Academic Press Professional Inc., 1994.

[2] M. Geier, J. Ahrens, and S. Spors, "Object-based audio reproduction and the audio scene description format," *Organised Sound*, vol. 15, no. 03, pp. 219–227, 2010.

[3] M. Gerzon, "Periphony: With-height sound reproduction," *J. Audio Eng. Soc.*, vol. 21, no. 1, pp. 2–10, 1973.

[4] K. Hamasaki, K. Hiyama, and R. Okumura, "The 22.2 multichannel sound system and its application," in *AES Convention*, 2005.

[5] N. Peters, "Proposing SpatDIF - the spatial sound description interchange format," in *Panel session at ICMC*, 2008.

[6] M. Pohja, "X3D and VRML sound components," HUT, Telecommunications Software & Multimedia Laboratory.

[7] C. Ramakrishnan, "Zirkonium: Non-invasive software for sound spatialisation," *Organized Sound*, vol. 14, pp. 268–276, 2009.

[8] E. Scheirer, R. Väänänen, and J. Huopaniemi, "AudioBIFS: Describing audio scenes with the MPEG-4 multimedia standard," in *IEEE Trans. Multimedia*, vol. 1, no. 3, 1999, pp. 237–250.

[9] Transcripts and notes, "Towards an interchange format for spatial audio scenes," http://redmine.spatdif.org/projects/3/wiki/Belfast_2008, 2008.

[10] R. Vaananen and J. Huopaniemi, "Advanced AudioBIFS: virtual acoustics modeling in mpeg-4 scene description," *IEEE Multimedia*, vol. 6, no. 5, pp. 661 – 675, 2004.