AMBISONICS SYMPOSIUM 2009
June 25-27, Graz

GIRAFE – A VERSATILE AMBISONICS AND BINAURAL SYSTEM

Martin Rumori[1]

[1] Academy of Media Arts Cologne (KHM) / Institute of Electronic Music and Acoustics Graz (IEM)
rumori@iem.at

May 31, 2009

Abstract: Girafe *aims at being a versatile, modular software system for realising projects using Ambisonics or the binaural virtual Ambisonics approach. It will be implemented as a set of unit generators and classes for the Supercollider sound programming environment.* Girafe *eventually will evolve into a flexible toolkit for audio augmented environments. It is targeted at both artistic applications in the field of interactive, virtual acoustics, as well as a tool for scientific research in spatial audio and its artistic applications.*
Currently, the implementation of the basic Ambisonics functionality of Girafe *is being designed. This paper discusses the functionality* Girafe *should provide and the implementation design with respect to its integration with Supercollider.*

Key words: Higher Order Ambisonics, Audio Augmented Environments, Binaural Virtual Ambisonics, Supercollider

1 INTRODUCTION

Audio augmented environments are a major subject in past years' scientific and artistic research. The European project LISTEN [1] for the first time integratively explored both the technological conditions and the aesthetical implications of creating and using binaural audio augmented environments.

Since then, the research efforts being undertaken which are connected to audio augmented environments are getting more and more diverse. The fields of psychoacoustics and multi-sensoric perception revealed various new research results, while, on the other hand, advanced technological approaches for spatial audio rendering became available and feasible.

Amongst these, Higher Order Ambisonics (HOA) provides promising means for greatly enhancing the rendering strategies used for audio augmented environments and virtual acoustics. Current computers are capable of doing massive multichannel realtime audio processing, which is the most important requirement for implementing HOA systems.

Artistic exploration of the capabilities of recent research results depends on the possibility of implementing these results. Most "pure" scientific research areas, however, do not necessarily allow for accessing their reference implementations, as they are most probably also inadequate for being dealt with by non-experts. A common platform which is suitable for both research and application purposes could help with bridging this gap.

Girafe aims at being such a platform. Its development is driven by the aim for a versatile, flexible and modular system, which, at the low level end, is open enough for extensions and enhancements. At higher levels, it should allow for creating an easy to use, yet powerful interface which is targeted at application building, such as artistic projects, but also the technical framework e. g. for listening tests in spatial audio research.

In computer music, the sound programming platform Supercollider provides a framework which is suitable as a basis for the *Girafe* system. It is designed for very efficient multichannel realtime audio processing and provides a sophisticated high level multi-paradigm programming language.

This paper describes the basic building blocks of the *Girafe* kernel for HOA and the design for integrating them with the Supercollider environment.

2 *GIRAFE*'S BASIC FUNCTIONALITY

The kernel of *Girafe* should provide the standard HOA operations, such as encoding, rotating and decoding Ambisonics signals in an open, modular way. It would be desirable to have the following basic functionality in *Girafe*:

2.1. Ambisonics Format

With *Girafe*, it should be possible to use different Ambisonics format variations, depending on the purpose of the application. This includes different Ambisonics orders or degrees, fully periphonic or horizontal-only formats as well as mixed order systems or reduced formats, such as proposed in [2]. This can be achieved by the separation of the actual vector/matrix operations and the calculation of the coefficients, which is described in subsequent sections.

In principle, this separation also allows for using different channel sequences and normalisation schemes. The standard implementation will use the *Ambisonic Channel Num-*

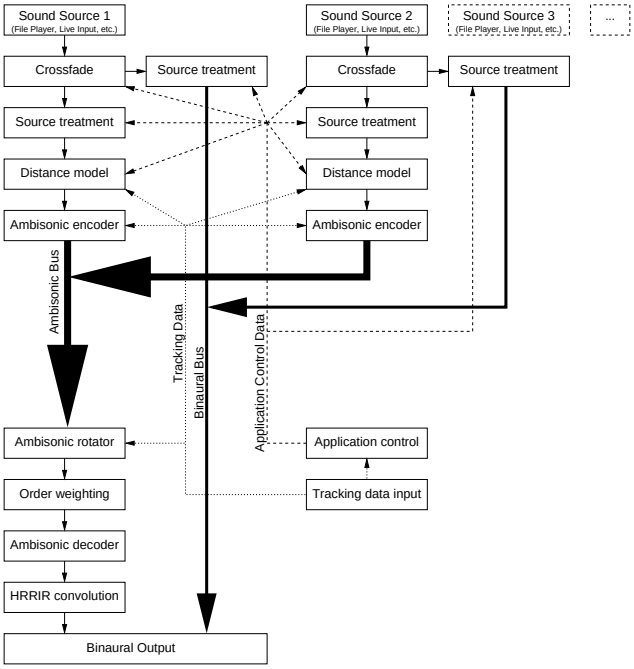

Figure 1: Example block diagram of a more complex virtual audio augmented environment application.

ber (ACN) convention [3] though, such that:

$$ACN = l * (l + 1) + m \tag{1}$$

The favoured normalisation scheme would be the semi normalised 3 D form (SN3D), as discussed in [4, pp. 155 ff.].

2.2. Ambisonics Encoding

Encoding a signal to the Ambisonics domain means the multiplication of the signal with the encoding coefficients vector. In *Girafe* the calculation of this vector and the generic multiplication operation are separated, instead of having monolithic Ambisonics panning signal processing units. This way, the encoding stage becomes very flexible and determines the layout of the Ambisonics domain signal. Of course, all subsequent operations in the Ambisonics domain have to take into account the actual layout of the signal, which is achieved by the same flexibility.

2.3. Ambisonics Rotation

Ambisonics rotation is done by multiplying the Ambisonics domain signal vector with the rotation matrix. This matrix is calculated separately from the matrix multiplication operation in *Girafe*, which allows for rotating signals according to their layout created at the encoding stage.

An interesting alternative for efficient rotation in 3 D is presented in [5]. Here, the rotation is expressed as three rotations around the z-axis plus two fixed rotations of 90 degrees each around the y-axis. This approach seems to be more generic to future extensions towards higher orders, as the rotation around the z-axis is fairly simple and can be implemented efficiently.

2.4. Ambisonics Decoding

Decoding an Ambisonics signal to a specific real or virtual loudspeaker layout is done by multiplying the signal vector with the decoding matrix. Finding an optimal decoding matrix especially for asymmetrical loudspeaker layouts is not a straightforward task. *Girafe* at least should provide the standard calculation method, which is the pseudo-inverse matrix of the encoded loudspeaker positions.

2.4.1 Dual-Band Decoding

One possibility for improving the localisation of sound sources is to use two decoding bands and to optimise them independently according to psychoacoustic laws. The low frequency band would be optimised to maximise the velocity vector ($max\ r_V$), while the high frequency band should have a maximum energy vector ($max\ r_E$).

Dual-band decoding is done by splitting the Ambisonics signal using phase-aligned filters, followed by decoding the bands separately with different decoding matrices.

Such band splitting filters can be found in *Ambdec* [6], which is a standalone application providing dual-band decoding up to second order. For the use in *Girafe*, these filters will be ported to Supercollider, which also would allow for using them for higher order Ambisonics domains.

2.5. Distance Coding/Near Field Correction

In [7], distance coding for HOA has been introduced. At the Ambisonics encoding stage, the distance of virtual sources is encoded with respect to a reference radius of the playback system. Before decoding, the signal still can be corrected in order to match the actual speaker readius.

[9] introduces an implemention of near field coding filters up to the fourth Ambisonics order. They have been ported (though yet unpublished) to Supercollider unit generators. Using these filters, *Girafe* supports Ambisonics distance coding.

These filters also can serve for compensating for the near field effect of the playback speakers in case of plane wave encoding. In non-ideal playback systems, the speakers may have different distances to the sweet spot, which might require separate near field correction for every speaker, along with a delay correction. Applying a correction filter after the decoding stage needs a decoder which outputs each Ambisonics order's portions of each speaker separately, as the filter curves differ for each ambisonic order.

3 SUPERCOLLIDER

Supercollider [10] is a generic purpose sound programming environment originally developed as a proprietary, commercial application for the Macintosh computer. In 2002, Supercollider in its version 3 became Free Software and shortly after was ported to the Linux platform.

Supercollider 3 introduced a client/server architecture. The

Supercollider server, *scsynth*, incorporates a powerful realtime processing model. Small signal processing entities, so called *unit generators*, form the building blocks of larger scale signal processing networks, so called *synths*. They are organised in an ordered tree on the server, which also controls their order of execution. The signal routing between Supercollider *synths* is done by *busses*, which allow for very complex, multichannel routing schemes. This system architecture makes Supercollider an ideal platform for HOA applications. The Supercollider server is entirely controlled by *OSC*, the *Open Sound Control* protocol [11], which makes it potentially agnostic to different control clients, and which also facilitates multi-node parallel signal processing.

For adding new *unit generators*, Supercollider provides a simple plugin interface in the C++ language. So called *pseudo unit generators* may also be constructed dynamically out of networks of simple mathematical operations, most probably with some performance loss compared to native implementations of these networks.

The standard Supercollider client, *sclang*, is a fully fledged programming language which implements the *scsynth* control protocol. It is a dynamically typed, object oriented language with many characteristics inherited from functional programming languages, has a C-style syntax, and its interpreter incorporates realtime garbage collection.

The Supercollider language contains *JITLib*, the Just In Time programming library, which was originally targeted at live coding. Through its highly dynamic approach, it is also very well suited for rapid prototyping for artistic mockups, but also for validation of scientific research results or for testing scenarios.

As *sclang* and *scsynth* usually only communicate via *OSC*, they may run on different machines, or one language client might control several servers.

Realtime multi-user audio augmented environments may be larger scale installations, which require both parallel processing, but also have different constraints on the latency of their signal processing. They may incorporate tracking systems which control Ambisonics operations. For dealing with control data on different levels of required realtime latency, Supercollider offers great flexibility. Processing may take place on client and/or the server side, which makes it possible to balance performance with convenience and system load. *Girafe* should enable the application developer to use either way of operation.

4 IMPLEMENTATION DESIGN

In this section, the implementation design in Supercollider for the basic Ambisonics operations of *Girafe* is presented. As mentioned in section 2, one fundamental principle is the separation of the actual vector or matrix operations from the calculation of the coefficients. For Ambisonics decoding, this is almost always the case, as the decoding matrix usually is static. This approach could be extended also to the encoding and rotation stage, although the coefficients need to be recalculated at a certain rate, usually control rate or a rate determined e. g. by the input of a tracking system.

The separation of the matrix operations and the calculation of the coefficients has several avantages over a monolithic approach with respect to the integration with Supercollider:

- Ambisonics encoding, rotating and decoding are generic vector or matrix multiplications, independent of the Ambisonics orders, format conventions, channel sequence etc., and therefore can be easily and efficiently implemented as server-side *unit generators*.
- Calculation of the coefficients can be done server-side or client-side, depending on the calculation rate, timing and performance demands.
- Client-side calculations can benefit from powerful language features, such as higher order functions or function composition.
- Server-side computing may take place in the same synth as the vector/matrix operation or in a different node, potentially combined with calculations for other synths at the same time.
- Server-side calculations can be dynamically composed as pseudo unit generators. Performance-relevant standard calculations can be factored out to native unit generators if necessary.
- Unified design principle for all Ambisonics operations, which are therefore easier to encapsulate for user access at a higher level of abstraction.
- As the separate coefficient calculation allows for a dynamic implementation, different Ambisonics signal layouts, normalisation schemes, reduced layouts etc. can be implemented much easier than with monolithic encoding/rotating/decoding plugins.

In order to allow for this flexibility, the coefficients have to be forwarded to the vector/matrix operations using Supercollider *buffers*. Their content can be set via *OSC* commands from the client side, or for server-side calculation using the `SetBuf` unit generator. If the coefficients are calculated in the same *synth* as the vector/matrix operations take place, the `LocalBuf` unit generator can be used, which operates on a buffer local to this *synth*. The downside of this approach is an inherent overhead for the buffer operations and the additional network traffic.

4.1. Ambisonics Encoding

Ambisonics encoding in *Girafe* is provided by a unit generator which multiplies a scalar (the input signal) with a vector of variable length (the encoding coefficients). The length of the vector is a creation-time argument. The unit generator is a so called `MultiOutUGen`, as it has a multichannel output, the Ambisonics signal. It can be processed separately, or can be sent to an Ambisonics bus using the `Out` unit generator. The encoding coefficients are read from a buffer.

```
GfAmbiEnc.ar(in, numOuts,
  bufnum = -1, bufIndex = 0);
```

A first order encoding *synth* with server-side coefficient computing might be implemented like this:

```
SynthDef.new('ambienc_1O', {

  arg out, in, azi, elev;

  var coeffs = LocalBuf.new(4);
  var cosElev = cos(elev);

  coeffs.set([1, cosElev * sin(azi),
    sin(elev), cosElev * cos(azi)]);

  Out.ar(out, GfAmbiEnc.ar(In.ar(in, 1),
    BufFrames.ir(coeffs), coeffs));
});
```

The same encoding with client-side coefficient computing might look like this:

```
b = Buffer.alloc(s, 4);

x = { arg out, in, coeffs;

  Out.ar(out, GfAmbiEnc.ar(In.ar(in, 1),
    BufFrames.ir(coeffs), coeffs));
}.play;

// calculate coeffs when needed
b.setn(0, [1, cos(elev) * sin(azi),
    sin(elev), cos(elev) * cos(azi)]);
```

4.1.1 Distance Coding

Distance coding could be implemented with *Girafe* using the filters described in section 2.5. For each Ambisonics order, a different distance coding filter has to be applied. In an efficient implementation, this would happen for each virtual sound source already before the Ambisonics encoding stage, as the number of channels and thus the number of filters is much lower. For encoding a source signal which is already split up into the contributions of different Ambisonics order, the encoding *unit generator* has to support multiple input signals, such that the final `GfAmbiEnc` signature design will look like this:

```
GfAmbiEnc.ar(in, numOuts,
  bufnum = -1, bufIndex = 0,
  routeBufnum = -1, routeBufIndex = 0,
  sparseBufnum = -1, sparseBufIndex = 0,
  sparseTrig = 1);
```

A separate vector of length `numOuts` is read from a buffer, which specifies the routing information of the input signal vector to the Ambisonics output signal vector. The default routing is to use only channel 0 of the input signal.

`GfAmbiEnc` also provides sparse optimisation as described in section 4.2, although the benefit usually will not as high as e. g. for Ambisonics rotation.

4.2. Ambisonics Rotation

Ambisonics rotation is provided by a unit generator which multiplies the Ambisonics signal vector with a quadratic rotation matrix. Likewise as with Ambisonics encoding, the rotation matrix is read from a buffer. This allows for calculating the matrix on the language or server side.

```
GfAmbiRot.ar(in, numIns,
  bufnum = -1, bufIndex = 0,
  sparseBufnum = -1, sparseBufIndex = 0,
  sparseTrig = 1);
```

Especially in the case of a rotation around the z-axis only, the rotation matrix is sparse. Therefore, `GfAmbiRot` provides a way of optimising the matrix multiplication for a static sparse matrix layout, which seems to be the most efficient way in this special context. A configuration matrix can be presented to the unit generator which will adjust its iteration processes to only take into account the non-zero values of the configuration matrix. The `sparseTrig` input will reread the configuration data. In case the actual rotation matrix contains no zeros at "non-sparse" indices, which is true e. g. for a z-axis rotation with a non-zero angle, the same buffer number for `bufnum` and `sparseBufnum` may be used for initial configuration. The default is to use no sparse matrix optimisation and to evaluate all indices of the rotation matrix.

For efficient calculation of a 3 D rotation matrix, especially using the approach mentioned in section 2.3, optimised matrix multiplications are also desirable. Both the R_z and the fixed R_{y90} rotation matrices are highly sparse. A special rotation matrix calculating unit generator could benefit from these optimisations.

4.3. Ambisonics Decoding

As with Ambisonics encoding and rotating, Ambisonics decoding is provided by a unit generator which performs the matrix multiplication. The decoding matrix is communicated via a buffer containing `numOuts` rows of `numIns` elements each, the latter depending on the Ambisonics order and signal layout being used.

```
GfAmbiDec.ar(in, numIns, numOuts,
  bufnum = -1, bufIndex = 0,
  numRoutes = 1,
  routeBufnum = -1, routeBufIndex = 0,
  sparseBufnum = -1, sparseBufIndex = 0,
  sparseTrig = 1);
```

`GfAmbiDec` also provides a sparse matrix configuration input, which will optimise the unit generator to not take into account zero fields in the decoding matrix. As the decoding matrix most probably will be static for a given setup, `sparseBufnum` can simply be the same as `bufnum`.

4.3.1 Near Field Correction

As described in section 2.5, separate near field correction for each speaker will need separate outputs for each Ambisonics order's contribution to each speaker signal. When

the argument `numRoutes` to `GfAmbiDec` is greater than one, the actual number of output channels will be $numOuts * numRoutes$. In this case `GfAmbiDec` will expect a routing vector configuration of length `numIns` in `routeBufnum` which specifies the grouping of the Ambisonics channels to the output routing groups. For a fully periphonic second order system, `numRoutes` will equal 3 and the contents of `routeBufnum` will be:

```
[ 0, 1, 1, 1, 2, 2, 2, 2, 2 ]
```

This means, that the zero order component (a. k. a. "W"-channel) will be output at the current speaker output index, the three first oder components will be summed up and output to the following output index, and the five second order components to the next following output channel. The near field correction filters can then indepently work on each Ambisonics order per speaker, before the corrected signals are summed up for output.

5 OTHER IMPLEMENTATIONS

5.1. Ambisonics Unit Generators

There are some Ambisonics related unit generators already contained in the Supercollider distribution or in an additional plugin package: the *AmbisonicUGens* and the *JoshAmbiUGens* packages. They are very well suited for audio spatialisation at lower Ambisonic orders up to second order. Their monolithic design is no affected by the complexity and performance issues of higher orders as discussed in this paper.

5.2. AmbIEM

AmbIEM [12] is a Supercollider port of the Ambisonics system developed at IEM Graz for Pd. Originally it is part of the SonEnvir project on applications of sonification.

AmbIEM incorporates classes for higher order Ambisonics encoding, rotating and decoding. All operations are implemented as pseudo unit generators with networks of basic mathematical operations. Therefore, the resulting *synth* networks get fairly complex, which eventually should affect the overall performance. The matrix operations are not especially optimised.

AmbIEM does not provide the same flexibility as *Girafe* eventually should do, as the coefficient calculations are coded into the classes for some Ambisonic orders and the fully periphonic approach only.

For the development of *Girafe*, *AmbIEM* serves as a reference implementation. Prior experiments and artistic explorations which led to the design proposal of *Girafe* were carried out using *AmbIEM* [13] [14].

6 CONCLUSION

The functionality aims of *Girafe* and the design proposal for its basic Ambisonic operations have been presented. As *Girafe* should be open and extendable to further scientific and artistic development, emphasis is put on these lower level design issues, as they may turn out as drawbacks and limitations later.

This paper should also serve as a discussion basis for the upcoming implementation process of *Girafe* and the design of its higher level parts, which will eventually more visible to the application developer.

7 ACKNOWLEDGMENTS

The author would like to thank many people for their input, discussions and direct or indirect contributions to the initial development attempts of *Girafe*, namely Fons Adriaensen, Gerhard Eckel, Stefan Kersten, Thomas Musil, Markus Noisternig, Alois Sontacchi, Johannes Zmölnig, Franz Zotter and many others.

REFERENCES

[1] Gerhard Eckel: *The Vision of the LISTEN Project*. Seventh International Conference on Virtual Systems and Multimedia (VSMM 01), p. 393, 2001.

[2] *A First Proposal to Specify, Define and Determine the Parameters for an Ambisonics Exchange Format*. Ambisonics Xchange Internet Platform, IEM Graz, `http://ambisonics.iem.at/xchange/format/a-first-proposal-for-the-format`, last retrieved: 2009–05–31.

[3] *Ambisonic Channels*. The Ambisonics Association, `http://ambisonics.ch/standards/channels/`, last retrieved 2009–05–31.

[4] Jérôme Daniel: *Représentation de Champs Acoustiques, Application à la Transmission et à la Reproduction de Scènes Sonores Complexes dans un Contexte Multimédia*. Ph. D. Thesis, University of Paris 6, Paris, France, 2000.

[5] Franz Zotter: *Spherical Harmonics Rotation Matrices. Rotation of Spherical Harmonics in R^3*. Ambisonics Xchange Internet Platform, IEM Graz, `http://ambisonics.iem.at/xchange/format/docs/spherical-harmonics-rotation`, last retrieved: 2009–05–31.

[6] Fons Adriaensen: *AmbDec. An Ambisonic Decoder for First and Second Order*. Linux Audio Page of Fons Adriaensen, `http://kokkinizita.net/linuxaudio/`, last retrieved: 2009–05–31.

[7] Jérôme Daniel: *Spatial Sound Encoding Including Near Field Effect: Introducing Distance Coding Filter and a Viable, New Ambisonic Format*. Proceedings of the AES 23rd International Conference, Copenhagen, Denmark, 2003.

[8] Markus Noisternig, Thomas Musil, Alois Sontacchi, Robert Höldrich: *3D Binaural Sound Reproduction using a Virtual Ambisonic Approach*. IEEE International Symposium on Virtual Environments (VECIMS), Lugano, Switzerland, 2003.

[9] Fons Adriaensen: *Near Field Filters for Higher Order Ambisonics*.

http://kokkinizita.net/linuxaudio/papers/hoafilt.pdf, last retrieved: 2008–09–12.

[10] James Mc Cartney et al.: *SuperCollider. Real-time Audio Synthesis and Algorithmic Composition*. http://supercollider.sourceforge.net, last retrieved: 2009–05–31.

[11] Matt Wright: *The Open Sound Control 1.0 Specification*. Version 1.0, March 26, 2002, http://opensoundcontrol.org/spec-1_0, last retrieved: 2009–05–31.

[12] Christopher Frauenberger et al.: *AmbIEM. An Implementation of an Ambisonics Rendering System for SuperCollider 3*. http://sonenvir.at/downloads/sc3/ambiem, last retrieved: 2009–05–31.

[13] Martin Rumori: *Spatial Interaction with Sonic Objects in Audio Augmented Environments*. Final Report on a Short Term Scientic Mission (STSM) in COST–IC 0601, Sonic Interaction Design, Institute of Electronic Music and Acoustics, Graz, Austria, 2008, http://trac.sme-ccppd.org/SID/browser/action/stsm/reports/Rumori/STSM_rumori_report_call1.pdf, last retrieved: 2009–05–31.

[14] Martin Rumori: *Advanced Binaural Synthesis for Interactive Audio Augmented Environments*. Final Report on a Short Term Scientic Mission (STSM) in COST–IC 0601, Sonic Interaction Design, Institute of Electronic Music and Acoustics, Graz, Austria, 2008, http://trac.sme-ccppd.org/SID/browser/action/stsm/reports/Rumori/STSM_rumori_report_call3.pdf, last retrieved: 2009–05–31.