Ambisonics

We will now cover:

- About ambisonics
- Theory of ambisonics in the ATK
- Encoding
- Decoding (not really though)
- Misc. fun stuff

Advantages of ambisonics

- Artistically flexible
- Sounds are composed irrespective of their playback

Disadvantages of ambisonics

- Setting up speakers for playback is difficult and the format is very dependent on precise placement of speakers
- Needs equidistant setup (like circles or spheres) unless you are an advanced engineer.
- There is a sweet spot (and it is small).
- You need lots of speakers for it to be cool.
- Physically moving about in space makes the illusion collapse to some degree

Alternative to ambisonics: Vector Based Amplitude Panning

- Aka. VBAP
- SuperCollider has support for it.
- See the original research paper: here.
- Some ambisonics decoder methods incorporate nice stuff from VBAP

History of ambisonics

- Developed in Britain in the 1970's by Michael Gerzon et al
- Publically funded research
- Was never a commercial success because it needed precise hardware.
- All of these original trademarks have expired.
- Recent years has seen an explosion in development

The Ambisonic Toolkit

Follow these instructions to do a full install.

A quick tour of ambisonics theory

... and the ATK

Features of the ATK

- Quite probably the most flexible ambisonic toolkit out there
- Maximum order is only limited by number of channels available
- Has helper functions for the difficult theory stuff
- Implements the Near-Field Controlled, aka Near-Field Compensated, form of higher order Ambisonics (NFC-HOA).
- Soundfield decomposition and recomposition -> You can make cool effects
- Plenty of functions for analysis

See this help file for an overview of the

A-format

A-format is a signal in the *angular domain*.

Examples: Mono, stereo, multichannel sound, A-format on microphones

B-format

B-format is a signal in the *ambisonic domain*

Very important point: Ambisonics is an *intermediary sound signal*

From Wikipedia:
"Unlike other multichannel surround formats, its **transmission channels do not carry speaker signals**. Instead, they contain a speaker-independent representation of a sound field called **B-format**, which is then decoded to the listener's speaker setup.

Advantage of high order ambisonics

- "spatial resolution".
- size of sweet spot.
- perception of depth.

Ambisonic orders

- Number of channels needed for order X: **(X + 1)^2**
- Eg for third order 3D: (3 + 1)^2 = 16 channels are needed

Helper functions for orders in the ATK

- Calculate number of channels from order: `HoaOrder(3).size`
- Calculate ambisonics order from number of channels: `AtkHoa.detectOrder(16)`
- Test if an order is as expect: `AtkHoa.confirmOrder(size, order)`

Signal components and ordering

- A first order signal is comprised of the components WXYZ.
- W is the first channel. It is equivalent to an omni directional mono signal. This is useful.
  XYZ are directional details adding to the W component.
- The order of these channels are determined by the **component ordering**. The most commonly used one is ACN, but you may still see FuMa (Furse-Malham) used.
- **ACN is in the ATK format** - verify by running `AtkHoa.ordering` in SC.
- See this wikipedia article for more information on this

Normalization

- The normalization method concerns the scaling/amplitude of the signals.
- SN3D is the standard these days and is part of the AmbiX format, but ***ATK uses N3D*** - verify by running `AtkHoa.normalisation` in SC
- See this wikipedia article for more information on this

Other convenience functions

The `AtkHoa` class has a number of convenient functions.

- Calculate ambisonics order from number of channels:
  `AtkHoa.detectOrder(16)`

The Near Field Effect

- The ATK uses a flavour of ambisonics which includes "near field correction"
- See this help file for more information.
- And the Jérôme Daniel paper "Spatial Sound Encoding Including Near Field Effect: Introducing Distance Coding Filters and a Viable, New Ambisonic Format"

Composing ambisonics signals in SuperCollider

Reference radius

- We will ignore near field correction stuff for now and just use the reference radius for now (1.5 meters py default)
- Available at `AtkHoa.refRadius`

Encoding: Turning our synth into an ambisonics synth

We will use the HoaEncodeDirection UGen to to our ambisonic panning.

Exercise: Add ambisonic encoding to your synth using this building block:

```
HoaEncodeDirection.ar(
        in: sig,
        theta: angle * pi,
        phi: elevation * pi,
        radius: AtkHoa.refRadius,
        order: 1
);
```

Hints: Encode just before the output stage. Also: `angle` and `elevation` need to be added as arguments.

```
SynthDef(\sawtooth, {
	// Arguments
	arg freq=442, cutoff=500, amp=0.5, out=0, attack=0.001, decay=0.99, dur=1, gate=1, angle=0, elevation=0;

	// Kill synth when done
	var done = 2;
	// Percussive envelope
	var env = Env.perc(attack, decay).kr(done, gate, dur);

	// The sound generator
	var sig = Saw.ar(freq);

	// Scale amplitude and apply envelope
	sig = sig * amp * env;

	// Filter high frequencies
	sig = LPF.ar(sig, cutoff);

	// Encode as ambisonics
	sig = HoaEncodeDirection.ar(
		in: sig,
		theta: angle * pi,
		phi: elevation * pi,
		radius: AtkHoa.refRadius,
		order: 1
	);

	Out.ar(out, sig)

}).play;
```

Why does it sound so weird?

- Because we are listening to the raw ambisonic signal
- Let's decode it!

Decoding

We are gonna keep it simple with this step, but here is more information for later:

- Stereo: stereo monitor help file
- Binaural: binaural decoding help file

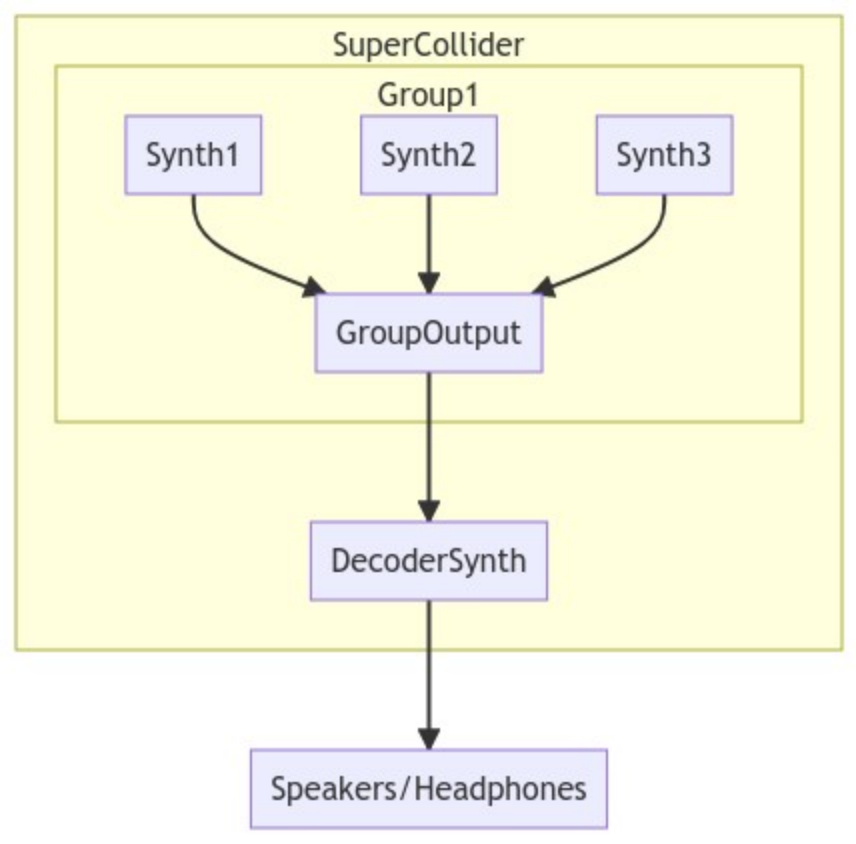
Making a decoder synth

There are many decoding methods out there.

But from a SuperCollider perspective, the signal path is simple.

All spawned synths must be passed through a "main output" type effect.

SuperCollider decoder signal path



```
(
fork{
    // Load resources
    Server.local.sync;
    ~binauralDecoder = FoaDecoderKernel.newCIPIC(subjectID: 21);

    // Make decoder synthdef
    Server.local.sync;
    SynthDef(\binauralDecoder, {|out=0, bypass=0|
        var order = 1;
        var clean = In.ar(out, order.asHoaOrder.size);
        var hoa = In.ar(out, order.asHoaOrder.size);

        // format exchange: HOA >> FOA
        var lowCutFreq = 30.0;  // highpass freq

        // design encoder to exchange (ordering, normalisation)
        var encoder = FoaEncoderMatrix.newHoa1;
        var foa, stereo, sig;

        // exchange (ordering, normalisation)
        // truncate to HOA1
        foa = FoaEncode.ar(hoa.keep(AtkFoa.defaultOrder.asHoaOrder.size), encoder);

        // pre-condition FOA to make it work with FoaProximity
        foa = HPF.ar(foa, lowCutFreq);

        // Exchange reference radius
        foa = FoaProximity.ar(foa, AtkHoa.refRadius);

        // Decode to binaural
        stereo = FoaDecode.ar(in: foa,  decoder: ~binauralDecoder);

        // Pad output with silence after the stereo channels
        stereo = stereo ++ Silent.ar().dup(order.asHoaOrder.size-2);

        // Allow to bypass
        sig = Select.ar(which: bypass,  array: [stereo, clean]);

        ReplaceOut.ar(bus: out,  channelsArray: sig);
    }).add;
}
)
```

Spawn the decoder synth

Spawn a decoder synth AFTER all the other synths (that are in group 1)

```
Synth.after(1, \binauralDecoder);
```

Note: *You need to do this every time you press cmd-period*

Playing your new ambisonic synth using a pattern!

- Specifying an instrument to play in a Pbind
  - Use the `\instrument` key: `\instrument, \sawtooth`
- Before proceding: Don't forget to spawn a decoder synth
  - **Make sure it's only one -** run `s.plotTree` and inspect the running synths to verify.

```
Pbind(
        \instrument, \sawtooth,
        \degree, Pseq((1..10), inf),
        \angle, 0,
        \elevation, 0,
).play;
```

Try modulating the spatial parameters

Random azimuth for example:

```
Pbind(
        \instrument, \sawtooth,
        \degree, Pseq((1..10), inf),
        \angle, Pwhite(-1.0,1.0),
        \elevation, 0,
).play;
```

Now have fun with this for a bit

Bonus: Interfacing with Ambix

To output ATK format ambisonics to be used in the IEM, Aalto or similar plugin suites:

- Filter the near field radius to an appropriate radius
- The channel ordering of Ambix and ATK format is the same, leave unchanged
- Normalization needs to be changed from N3D to SN3D

Don't worry: There's a full synthdef on the next slide

Convert from ATK to Ambix format:

```
(
SynthDef(\atk2ambix, {|out=0, bypass=0|
    var order = 1;
    var hoa = In.ar(out, order.asHoaOrder.size);
    var clean = hoa;

    // format exchange: HOA >> FOA
    var lowCutFreq = 30.0;  // highpass freq

    hoa = HoaNFCtrl.ar(
        in: hoa,
        encRadius: AtkHoa.refRadius,
        decRadius: 10.0,
        order: order
    );

    // exchange normalisation: N3D to SN3D
    hoa = HoaDecodeMatrix.ar(
        in: hoa,
        hoaMatrix: HoaMatrixDecoder.newFormat(format: \ambix, order: order)
    );

    // Allow to bypass
    hoa = Select.ar(which: bypass,  array: [hoa, clean]);

    ReplaceOut.ar(bus: out,  channelsArray: hoa);
}).add;
)
```

… And then:

```
Synth.after(1, \atk2ambix);
```

Bonus: Advanced decoding

- You can use the VST Plugin package as well (see this example)
- Custom decoders using The Ambisonic Decoder Toolkit. This package integrates it in SC
- Use a head tracker. We've made this open source one. which is a remix of an IEM head tracker.

Bonus: Recommended further reading: Ambisonic enlightenment

A help file that comes with ATK in SC:
Ambisonic enlightenment

Bonus: Making natively ambisonic effects

- The ATK allows you to make ambisonic native sound effects
- The technique is called "Spherical decomposition" (and recomposition)
- See this help file for a tutorial: Spherical decomposition guide

Thanks for sticking around

If you see this, you probably endured the whole workshop. Well done :)