ImmLib - A new library for immersive spatial composition

Miguel Cerdeira Negrão
Queen's University Belfast, Sonic Arts Research Centre
Cloreen Park Belfast, BT7 1NN, Northern Ireland
miguel.negrao@friendlyvirus.org

ABSTRACT

ImmLib is a new software library for spatial composition with grid-based loudspeaker systems in the context of computer sound synthesis and audio processing which places emphasis on immersiveness and a global approach to space. It implements techniques for dealing with multiple decorrelated, but perceptually similar, sound streams spatialized at different locations in space with the aim of creating an expanded, broad or diffuse sound source with interesting musical spatial properties. The tool, implemented in SuperCollider, automates the process of creating decorrelated streams from a synthesis definition and provides mechanisms to create and control spatial patterns in a virtual surface by modulating synthesis parameters of the sound processes using different (but coherent) signals for each of the running instances. The main abstraction is the *parameter field* which defines ways to control the spatial patterns across space based on mathematical functions defined on a surface. We present here motivation for the library, the general algorithm and abstractions and a brief overview of the implementation, syntax and empirical evaluation.

1. INTRODUCTION

There is a rich history in computer music of using spatialization technologies as tools serving a compositional idea [1]. Common spatial compositional techniques include using trajectories, serializing the location parameter, diffusion, simulation of acoustics, resonating spaces or using sounds that are evocative of specific physical locations [2]. Different algorithms have been implemented to place sounds in space (VBAP, Ambisonics, WFS, etc), usually under the form of a single point source (or plane wave) and different approaches have been taken regarding how to use those tools in the compositional process. Such spatialization technologies have both been used to contrast a relatively small amount of sources by using clear and well defined trajectories and to create rich spatial textures composed of many micro events or streams with different positions and movements that fuse into a bigger dynamic sound source. There are still few tools to deal with such spatial *textures*, and the ones that exist are usually tied to a single synthesis/processing technique (granular spatialization, spectral spatialization, etc.).

In recent years it has become more commonplace to have multi-loudspeaker setups available when presenting computer music. These setups range from 4 channels at horizontal level, through domes and spheres, to Wave Field Synthesis using tens or hundreds of individual speakers. A number of spatialization technologies for virtual positioning of sound sources (usually point sources) have been implemented in common software environments for real-time synthesis and processing (Max/MSP, SuperCollider, Csound) and digital audio workstations. The most popular are Ambisonics [3, 4, 5], Vector Based Panning (VBAP) [6, 7] and Wave Field Synthesis [8, 9, 10].

Composers who have wished to work with more enveloping sources or with sound objects created out of many individual "particles" or streams have resorted to different strategies to create broader or more diffuse sound sources, usually using one of the techniques previously mentioned for the actual panning. Some of the most relevant such strategies are Spatial Swarm Granulation [11], Spectral Spatialization [12], Decorrelation, [13, 14], Deduplication with band-pass filters, pitch-shifting or delays [15], Spatio-Operational Spectral Synthesis [16] and Image Based Spatialization [17].

There are also several examples of sound works, some computer related and others wholly acoustic, where several similar sounding sound streams were spatialized (or just placed) at different locations and manipulated in order to create coherent spatial patterns. Notable examples are *Terretektorh* and *Nomos Gamma* (Iannis Xenakis, 1966 and 1967-1968), *SoundBits* (Robin Minard, 2002, 576 channels of 1-bit audio, 576 piezo speakers, computer-controlled spatialization [18], *Pneumatic sound field* (Edwin van der Heide, 2006, 42 pneumatic valves, computer-controlled spatialization [19]), *Untitled Sound Objects* (Zimoun and Pe Lang, 2008, 250 prepared electro hub magnets in wooden space), *Signe* (Steve Heimbecker, 2008, Turbulence Sound Matrix system, 64-channels, computer-controlled spatialization [20]), *Coincidence Engine Two* ("The User", Emmanuel Madan and Thomas McIntosh, 2008, 96 modified clocks, custom electronics and software [21]), *Spaced Images with Noise and Lines* (Eric Lyon, 2011, 8-channel composition [17]).

ImmLib is a new software tool being developed with the goal of simplifying the work flow when dealing with spatial textures by allowing many types of spatial patterns to be created through high level control of the parameters of

any sound synthesis or audio processing algorithm (sound process).

2. IMMLIB

ImmLib implements a set of spatial composition techniques. The library presents an environment with the basic tools for working with multiple copies of the same sound process spatialized at different points in space, allowing users to focus on how the sound processes evolve and the spatial patterns used, relieving them of the tedious task of setting up the multiple streams and connections. The system enables setting up rules for determining the evolution of the spatial relationship of the same parameter in multiple copies of the same sound process spatialized at multiple locations by defining *parameter fields*. As a high level framework for spatial composition the system can use different spatialization algorithms, although currently only VBAP and direct speaker addressing is implemented.

Spatial patterns in this context are taken to mean the suggested movement or spatial impression that is perceived when one hears a complex sound source composed of many individual elements spread in space. The contributions from each source are received by the listener and often (but not always) fuse into one single complex percept. Depending on the spectral and temporal relation between the sonic output of each source they can fuse perceptually into one spatially diffuse sound object, create a texture where individual components can still be somewhat detected but are nevertheless perceived as part of the whole, or be perceived as entirely separate entities. This spatial impression can be indeterminate such as with a sense of ”space”, ”depth” or ”volume”, with it's features only vaguely perceivable and difficult to separate, or be very well defined such as with precise periodic movements (”waves”, ”sweeps”, etc.) with quantifiable features (frequency, width, speed, regularity). In any case, it's not easy to find terms to describe such patterns and, as often happens, one has to borrow terms from the visual domain or mathematics.

2.1 Model and Implementation

ImmLib is implemented as a set of classes for the SuperCollider language [22] and auxiliary programs. It is implemented on top of the UnitLib library [23]. Unit Lib allows synthesis definitions (*Udef* class) to be instantiated into units (*U* class) which can be easily send audio to each other via chains (*UChain* class). A score (*UScore* class) can be built by placing chains on a timeline. Listing 1 shows UnitLib's syntax.

Listing 1. UnitLib usage example.

```
var x = UChain(
        [\whiteNoise, [\amp, 0.5]],
        \stereoOutput
)
.startTime_(10)
.duration_(60);
UScore(x).prepareAndStart
```

ImmLib is designed to work with two-dimensional grids of loudspeakers covering spaces such as domes, spheres or single walls. It spatializes a sound process, defined in a Udef, by creating multiple instances of the sound process and spatializing them at points isotropically distributed on an imaginary surface, conceptualized as hovering over the loudspeaker grid. The speaker system and spatialization method being used with the library must be able to place a virtual source on this virtual surface in order for the library to work correctly (e.g. a full sphere requires speakers above and below the floor level). The spatial patterns are created by associating mathematical functions defined on this conceptual surface, which we call *parameter fields*, with a parameter of the sound process.

The surface can be modeled mathematically by a two-dimensional Riemannian manifold which has associated a set of coordinate maps. A coordinate map is an homeomorphism which sends a portion of a plane into three-dimensional space ($f : A \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$). It's inverse function ($f^{-1} : B \subset \mathbb{R}^3 \rightarrow \mathbb{R}^2$) goes from the surface embedded in three-dimensional space back to the coordinate space. Coordinate map's are useful in this context because to define a function f on the surface we only need to define the function on the flat two-dimensional coordinate space since the coordinate map g will extend the function to the surface in three-dimensional space through composition $f' = g \circ f$. It's usually easier to define functions on the two dimensional coordinate space then on the three-dimensional space where the surface is embedded, and for this reason in ImmLib parameter fields are functions of u, v, the two spatial coordinates and t, time. Having a way to measure distances on the surface is useful for writing interesting functions. Given a connected Riemannian manifold the distance between two points can be determined by the arclength of a minimizing geodesic connecting them. The distance function d defined on the surface can likewise be pulled back onto the coordinate space $d' = g^{-1} \circ d$.

A surface is implemented in ImmLib by the PSurface class which contains an instance of PRiemannianManifold. To create a PRiemannianManifold one must supply a coordinate map and it's inverse, a distance function defined on the coordinate space ($d : \mathbb{R}^2 \rightarrow \mathbb{R}$), the domain of the coordinate map ($A \subset \mathbb{R}^2$), the maximum distance between two points on that domain and whether the surface is closed or not. A PSurface also contains an array with points on the coordinate space which determine the number and location of the point sources to use when spatializing each copy of the sound process. Currently the available surfaces are *PSphere(n)* which generates n points on the unit sphere using a simple algorithm from computational geometry [24], *PGeodesicSphere(n)* which generates a set of points by successive projections of an icosahedron onto the unit sphere and *PPlane(origin, dx, dy, n, m)* which creates a parallelogram rectangle in 3D space.

Given a Udef, a PSurface with n points and and an associative array of m synthesis parameter names (e.g. *'freq'*) to definitions of control signals, the library will create n units and spatialize each one at the nth point of the PSurface. For each parameter in the associative array n control signals are created and connected to the corresponding units.

The control signals are generated either from the SuperCollider language by sending OSC messages to the server at a regular interval (usually 0.1s) or by creating auxiliary audio process in the server and connecting them into the corresponding parameters of each unit. In either case the control signals are generated from parameter fields.

2.2 Parameter Fields

Parameter Fields (pfields) model spatial patterns using real valued functions of time defined on the coordinate space of the surface:

$$f : A \subset \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}.$$

Given a parameter field pf to be assigned to synthesis parameter p, it's value at time t and position $w = (u, v)$,

$$x = pf(u, v, t),$$

represents the numeric value of p at position w at time t. Ideally, the sound coming from that direction should sound as if the parameter of synthesis has the value x. In the simplest case where a parameter field is directly connected to a parameter of synthesis (no further manipulation of the output of the parameter field), given unit u_n spatialized at $p_n = (u_n, v_n)$ the control signal for parameter m is generated by animating the function

$$f_n(t) = \mathrm{pf}(u_n, v_n, t).$$

Since a parameter field is defined in terms of coordinates (u, v) in the coordinate space it's possible to apply the same field to different surfaces by just composing it with different coordinate maps, this allows for the definition of pfields in ImmLib to be independent of the surface being used such that if a score is created with a certain surface in mind, it will be relatively easy to convert it to a different surface.

In general parameter fields can themselves have parameters, in which case the control signal for point n becomes

$$f_n(t) = \mathrm{pf}(u_n, v_n, t, c_1, c_2, ...).$$

The parameters can be modulated by time dependent functions

$$f_n(t) = \mathrm{pf}(u_n, v_n, t, c_1(t), c_2(t), ...)$$

or by other parameter fields

$$f_n(t) = \mathrm{pf}_1(u_n, v_n, t, \mathrm{pf}_2(u_n, v_n, t, d_1(t), ...), c_2(t), ...).$$

If the mathematical function used for a pfield is continuous on the spatial variables [1], which is not uncommon for functions given by simple formulas, then the relationship between the values of the synthesis parameter being modulated at two points close together is not arbitrary: the closer the points the closer the values of the parameter. This gives rise to *spacial coherency* which contributes to the creation of spatial patterns. This is perhaps the most important property of parameter fields.

[1] For all t_0, $(x, y) \mapsto f(u, v, t_0)$ is continuous.

Let us go over a concrete example. Consider the pfield defined by

$$\mathrm{pf}(u, v, t, u_0, v_0, c) = \begin{cases} 0 & \text{if } \mathrm{dist}((u, v), (u_0, v_0)) > c \\ 1 & \text{if } \mathrm{dist}((u, v), (u_0, v_0)) \leq c \end{cases} \quad (1)$$

where $p_0 = (u_0, v_0)$ is a fixed point on the surface. This pfield doesn't depend on time, only on p_0 and c. It acts as a "spotlight": any points closer to p_0 then c are lighted (value 1) and all the other points are in the dark (value 0). Figure 1 shows a plot of this pfield, using the built in plotter (PSmoothSurface and PGridPlot classes) which can animate a real-time visualization of the field and figure 2 shows the positions of the virtual sources. Listing 2 shows the ImmLib code for creating the field defined in equation 1. First a Udef is created with one white noise generator passing through a low pass filter, with controls for amplitude and filter frequency, then the "spotlight" pfield is defined using a function and connected to the `freq` parameter using an ImmDef. The ImmDef is given specs used to create a GUI with sliders for real-time control of u_0, v_0, and c. Finally a chain is created with one unit and associated ImmDef and added to a score.

Listing 2. PField example

```
(
Udef(\filteredNoise, {
  arg freq = 440, amp = 0;
  var noise = WhiteNoise.ar;
  var in = UIn.ar(0,1);
  var out = BLowPass.ar( in, freq.clip
      (20,20000).lag(1), 1 );
  UOut.ar(0, out * amp )
})
.setSpec( \freq, \freq.asSpec )
)
(
var m = 40;

//surface definition
var surface = PSphere(m);
var distFunc = surface.distFunc;
var maxDist = surface.maxDist;

//pfield definition
var pf = PField({
  arg u, v, t, u0=0.0, v0=0.0, c=0.0;
  var x = distFunc.(u, v, u0, v0)/maxDist;
  if( x < c ){ 1 }{ 0 }
});

//connect pfield to freq param.
var def = ImmDef({
  //timer and sliders event streams
  arg t, u0, v0, c;

  var out = pf.(t, u0, v0, c);

  //scale between 100Hz and 3000Hz
  var out2 = out.linlin(0.0, 1.0,100,3000);

  ( freq: UArg( out2 ) )

}, surface, 0.1,[
        //specs for sliders
        \u0, \azimuth,
```

```
        \v0, \elevation,
        \c, [0,1.0]
]);

//setup chain and score
var mod = ImmMod(def, [\c, 0.5]);
var chain = ImmUChain(
  surface, 0, 1, inf, true,
  [\filteredNoise, [\amp,0.1], mod];
).fadeIn_(1).fadeOut_(1);
var score = ImmUScore(surface, chain);
score.gui
)
```

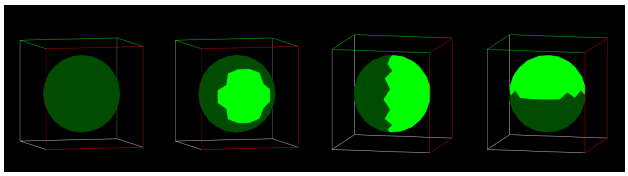

Figure 1. Parameter field "spotlight" plotted with $(u_0, v_0) = (0, 0)$, $c = 0$, $c = \frac{1}{4}$, $c = \frac{1}{2}$ and $(u_0, v_0) = (0, \frac{\pi}{2})$, $c = \frac{1}{2}$

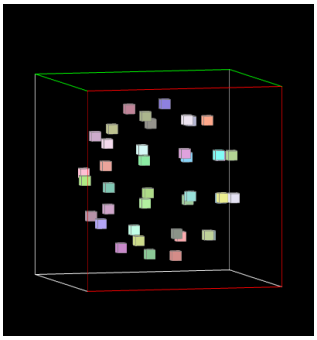

Figure 2. Positions of the virtual sources for *PSphere(40)*

Listening to this example on a spherical setup with speakers above and below the floor level, with $c = 0$ one would hear an enveloping cloud of low passed noise coming from all directions. Each of the 40 virtual sources is playing a decorrelated noise signal which contributes to the feeling of being enveloped. Increasing the c parameter from 0 to 1, one would hear the sound in the frontal direction start to become brighter, this brighter region would start to expand symmetrically until finally occupying the whole sphere. The lag on the *freq* control ensures a smoother transition between the two frequency values.

2.3 Functional Reactive Programming

The evaluation and animation of pfields is done in *sclang* [2] using an implementation for SuperCollider of Functional Reactive Programming (FRP) [25, 26] part oftechniques FPLib [27]. This allows for the animated pfields to easily interact with other event streams [3] such as from graphical user interfaces, MIDI or Open Sound Control (OSC) messages. Time is represented by an event stream of floats carrying the elapsed time in seconds and updated periodically by a timer. A PField is evaluated using the mandatory time event stream, together with optional event streams for additional parameters and returns another event stream carrying an array with the result of the evaluation of the function at each of the points of the PSurface. The returned event stream can be further manipulated before being associated with the synthesis parameters using any of the combinators of FRP such as `fmap` (processing values), `accum` (storing state), `apply` (applicative functor interface, for combining streams) or `switch` (dynamic event switching) [4]. For instance two evaluated PFields can easily be multiplied:

```
//evaluate field 1
var a = pf1.(t)
//evaluate field 2
var b = pf2.(t)
//multiply streams
//applicative syntax
{ arg x,y; x*y} <%> a <*> b
//lift syntax
{ arg x,y; x*y}.lift.(a,b)
//using polymorphism
a * b
```

Mathematical operators can also be used directly on event streams through polymorphism, but this is expensive as it creates one thunk per operation.

The switchInto combinator allows changing the stream currently associated with a synthesis parameter dynamically, when some other stream produces an event. This allows extending the system to use not just pure mathematical functions but also algorithmic definitions which react to events in real-time. It also allows for efficient implementation of functions which are piece-wise on the time parameter. Listing 3 shows an example where every time a slider moves from it's two values, 0 or 1, the result stream switches between using pf1 or pf2.

Listing 3. Dynamic event switching

```
var def = ImmDef({
  arg t, slider1;

  var out = slider1.switchInto{ |v|
    if(v==0){pf1.(t)}{pf2.(t)}
  };

  ( amp: USpecArg( out ) )

}, surface, 0.1,[
  //specs for sliders
  \slider1, ControlSpec(0,1,step:1)
]);
```

Event streams can also be used to entirely bypass the pfield mechanism, creating the final values directly. This was used to prototype and experiment with *continuous cellular automata* and *reaction-diffusion systems* which instead of using pure mathematical functions are implemented

[2] The SuperCollider language interpreter.

[3] Classic FRP has two main abstractions Event Streams (discrete) and Signals (continuous). In ImmLib we use mainly Signals, nevertheless we will continue to refer to event streams which should be clearer for a wider audience.

[4] In FPLib these are called `collect`, `inject`, `<%>` and `switchInto`.

through algorithms that update values of a cell of a grid based on the values of neighboring cells.

2.4 Predefined Parameter Fields

The library comes supplied with a number of predefined pfields, below we list some of them. Gradient, wave2D, spotlight, expandContract and sphericalHarmonic are defined directly using mathematical functions, continuousRandomSpotlight, randomHills and moveHills are defined using FRP and the spotlight function.

- *gradient(t, u0, v0, a, b)* : A pfield defined by

  $$pf(u, v, t, u_0, v_0, a, b) = a(1 - x) + bx$$

  where

  $$x = \frac{d((u, v), (u_0, v_0))}{\text{maxDist}} \text{ and } a, b \in [0, 1].$$

  It goes linearly from value a at point (u_0, v_0) to value b at points at the maximum distance from (u_0, v_0). See Figure 3.

- *wave2D(t, u0, v0, l, freq, g)*: A pfield defined by

  $$\text{pf}(u, v, t, u_0, v_0, f, l) = g(lx + ft)$$

  where

  $$x = \frac{d((u, v), (u_0, v_0))}{\text{maxDist}}$$

  It implements a wave traveling through a surface caused by a point source at (u_0, v_0) emitting a signal given by function g where the speed of propagation is proportional to l. The speed of propagation controls the spatial wavelength, i.e. the distance of two successive peaks on the surface for fixed t. See Figure 3.

- *spotlight(t, u0, v0, c, d)* : A pfield similar to equation 1. It grows symmetrically from a start point until occupying all the surface. It returns 1 if the distance between (u, v) to (u_0, v_0) is smaller then $c \in [0.1]$ and 0 otherwise. $d \in [0, 1]$ controls the wideness of the smoothing area when transitioning from 0 to 1.

- *expandContract(t, u0, v0, c)* : A pfield defined by

  $$\text{pf}(u, v, t, u_0, v_0, c) =$$
  $$\begin{cases} \text{spotlight}(u, v, t, u_0, v_0, 2c) & : c \in [0, 0.5[\\ \text{spotlight}(u, v, t, u'_0, v'_0, 2(1 - c)) & : c \in [0.5, 1]. \end{cases}$$

  It expands symmetrically from a start point until occupying the entire surface and then shrinks into (u'_0, v'_0), the antipodal point.

- *sphericalHarmonic(m, l)(t, f)* : A pfield defined by the spherical harmonic function of degree l and order $-l \leq m \leq l$. The spherical harmonic function is multiplied with a sinusoidal function with frequency f. See Figure 3.

- *continousRandomSpotlight(t, numSecs, curve)* : A pfield that grows symmetrically from a start point until occupying all the surface and then shrinks back to the same point, it then randomly chooses another point to grow from and repeats the procedure. It takes *numSecs* to grow from and shrink back to the chosen point.

- *randomHills(t, numSecs, numHills, sizeA, sizeB, bumpSize, heightA heightB)* : Every *numSecs* seconds switches into a new function. Each function is composed of the sum of *numHills* "spotlight" functions with centers at random points (the hills) each hill having a random wideness in *[sizeA, sizeB]* and height in *[heightA, heightB]*. The animation progressively cross-fades from one set of hills to another in *numSecs* seconds.

- *moveHills (t, numSecs, numHills, size, step)* : *numHills* "spotlight" functions are created with random center locations and wideness given by *size*. On each iteration of the timer (the time between iteration is given by the *delta* parameter of ImmDef) the center of each hill is moved in a random direction by a distance given by *step*. This creates Brownian movement of the center of the spotlight functions.

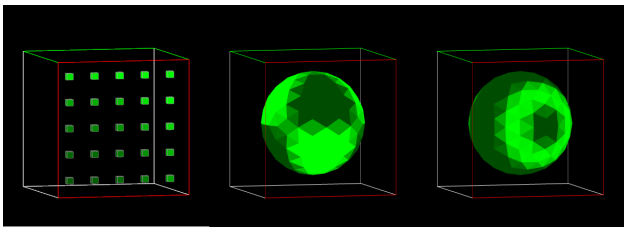

Figure 3. Plots of parameter fields *gradient* (wall configuration), *spherical harmonic* and *wave2DSin* (spherical configuration).

3. EMPIRICAL EVALUATION

The pfields created so far have been evaluated by the author in the Sonic Lab of the Sonic Arts Research Center of Belfast Queens University. The lab has 4 rings of 8 speakers at heights of 5.5m, 2.83m, 0m, and -4.6m with a total of 32 speakers. This system allows placement of sounds roughly in any direction, which suggests the use of a sphere as surface for ImmLib. Each pfields was evaluated taking note of how different types of sound processes (noisy, pitched, synthetic, file based, etc.) and different values of the pfield parameters create different types of behaviors.

While this evaluation is ongoing we can already formulate some provisional observations.

Given that the sound process is entirely determined by the user the final sonic result will in practice be the outcome of the interaction between pfields and the parameters of synthesis they control. When using the library one needs to experiment with assignment to different synthesis parameters as the results can be quite different depending on the parameter chosen.

Each instance of the mono sound process which is created should be decorrelated [14] from the other instances. If this is not the case, then the multiple streams will sound identical and might be perceived as a single point source due to the precedence effect. In practice this means that either the sound process should include non-deterministic unit generators (which SuperCollider automatically instantiates with different random seeds) or in case all unit generators are deterministic the range used for the values should be big enough for the resulting streams not to be correlated. When using a (mono) sound file as source material, it should either be sent through an included decorrelation unit [5] , the reading start positions in the sound file should be different for each point of the surface or further processing should be applied taking into account what was just mentioned before. Using noise and impulse generators as main sound source works quite well with the system since the streams are completely uncorrelated at each point of the surface.

Depending on the values of the parameter field, the type of parameter it is attached to and the type of sound process used there can be strong or weak agreement between what the pfield looks like on the animated plot and what is actually heard. When the pfield is moving the agreement tends to be stronger with slow movements and break down completely when speeds cause each stream to amplitude modulate at frequencies close to 20Hz.

Amplitude is a special parameter, slightly different from other parameters: assigning a pfield to the amplitude parameter causes the sound to be played at different times in different areas of the surface, therefore we can make the sound "travel" through the surface. Traveling includes not only going from point a to point b but also going from nowhere to everywhere, bifurcating and other more complex behaviors.

As an example of the observations made during this evaluation, we present part of the analysis of the *wave2DSin* pfield assigned the to amplitude control of a white noise generator.

With $f = 0.2\text{Hz}$ and $u_0 = 0, v_0 = 0$ a slow rhythm is created, as it takes 5 seconds for the wave to travel from one side to the other of the room. Depending on the value of l different effects are created:

1. $l = 0$: There is no spatial movement, sound comes from every direction.

2. $l = 0.05$: The sound, appearing from the back, quickly fills the room, stays in all the space for a while, then exits quickly from the front going into full silence. The filling of the space happens symmetrically along the back-front axis. One needs to be attentive to detect that the sound is starting from the back.

3. $l = 0.2$ It's very clear that sound expands from the back to the front. At no point does it fill the whole space.

4. $l = 0.3$: There is even less filling of the space, it's more clear where the sound is at each moment as if it was a band or stripe. The loudness when the peak crosses the center of the room is higher the previously. Occasionally it appears one can hear two distinct peaks of amplitude halfway through the movement. Still goes to silence after each passage.

5. $l = 1.2$: The peaks pass through the room without it ever going to full silence. When one peak of amplitude is arriving on the front one hears another peak appearing again at the back.

Front to back and back to front movements (i.e. the same movement after the listener has rotated 180 degrees) provoke different reactions. Back to front seems to be more intense, perhaps because the sensation that something is behind oneself increases alertness.

Comparing the difference between white noise, white noise modulated with a low frequency square wave and a random impulse generator (where in all three the pfield was assigned to the amplitude control) and fm synthesis (where the pfield was assigned to the fm modulation index, *i*) it was observed that simple white noise seems to make the movement more clear when the movement is sideways (relative to the head) while modulated white noise seems to be more clear when the movement is along the front-back axis. The impulse generator creates a quite clear movement as well. Fm creates a much more ambiguous movement, although setting the range for *i* high enough makes the movement definitely perceivable. The spatial sensation created by modulating *i* is quite interesting, it's subtle yet paying close attention one can indeed track the movement of the increase in energy of higher frequencies across the room.

The library has some practical issues. As it is based on the duplication of the same sound process n times, the cpu load also increases n times when compared to a non-duplicated process. Even on modern multi-processor and multi-core computers [6] it's fairly easy to consume all the available resources which places a limit on the complexity of a composition. When testing a single sound process it was possible to use a surface with up to 60 points on an 8-core computer, but with a complex composition the maximum number of points can only be a fraction of that. The default evaluation strategy for pfields runs on *sclang* which is notably slow and not multi-core aware, the number of pfields/points that can be evaluated simultaneously is therefore not very high. To address this issue, an alternative evaluation strategy that runs on *scsynth* has been added although it is limited to pfields that don't use dynamic event switching and the syntax used is different. The learning curve of the library is somewhat steep as users must first become familiar with the syntax of both UnitLib and FPLib's FRP system before they can fully use ImmLib.

The library has been used by two composers at SARC as part of a collaboration started in order to obtain feedback and useful insight with one work making use of the library already presented to the public.

[5] The use of the decorrelation unit does not significantly increase cpu usage.

[6] UnitLib can take advantage of all the cores available by using n scsynth servers for n cores, although some restrictions apply.

ImmLib will be made freely available under the GNU General Public License.

4. CONCLUSIONS AND FUTURE WORK

In this paper we presented a new library for spatial composition with grid-based loudspeaker systems. The framework built around surfaces and functions on surfaces (pfields), the emphasis on spatial coherency across multiple parallel control signals and the generality in regard both to the synthesis/processing algorithms and the spatial functions used distinguish this system from other spatialization tools.

Future work will focus on adding more pfields to the library and adapting it to make use of other panning systems (Ambisonics, binaural). Pfields could also be used for the creation of clouds of events if interpreted as probability distributions: the value of a pfield at a point would represent the probability that an event is created at that location. The system could be extended to use 3D grids of speakers (e.g. 5 x 5 x 5 with 2m spacing) by using volumes instead of surfaces and using parameter fields defined in 3D space which would be functions of x,y,z,t instead of u,v,t. The output for each point could be sent directly to each speaker or Distance-Based Amplitude Panning could be used[28]. Finally there is potential in exploring systems such as reaction-diffusion due to the richness of behavior that can be extracted from a comparatively small amount of parameters.

Acknowledgments

This work was funded by the Portuguese Foundation for Science and Technology (*Fundação para a Ciência e a Tecnologia*).

5. REFERENCES

[1] J. Chowning, "Turenas: the realization of a dream," *Proc. of the 17es Journées d'Informatique Musicale*, 2011.

[2] M. A. Baalman, "Spatial composition techniques and sound spatialisation technologies," *Organised Sound*, vol. 15, no. 03, pp. 209–218, 2010.

[3] M. A. Gerzon, "Periphony: With-height sound reproduction," *Journal of the Audio Engineering Society*, vol. 21, no. 1, pp. 2–10, Feb. 1973.

[4] D. Jérôme, "Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia," Ph.D. dissertation, Ph. D. Thesis, University of Paris VI, France, 2000.

[5] J. C. Schacher and P. Kocher, "Ambisonics spatialization tools for max/msp," *Omni*, vol. 500, p. 1, 2006.

[6] V. Pulkki, "Virtual sound source positioning using vector base amplitude panning," *Journal of the Audio Engineering Society*, vol. 45, no. 6, 1997.

[7] S. Wilson and J. Harrison, "Rethinking the BEAST: Recent developments in multichannel composition at birmingham ElectroAcoustic sound theatre," *Organised Sound*, vol. 15, no. 03, p. 239–250, 2010.

[8] A. J. Berkhout, D. de Vries, and P. Vogel, "Acoustic control by wave field synthesis," *The Journal of the Acoustical Society of America*, vol. 93, p. 2764, 1993.

[9] S. Spors, R. Rabenstein, and J. Ahrens, "The theory of wave field synthesis revisited," in *124th AES Convention*, 2008, p. 17–20.

[10] "GameOfLife - WFSCollider." [Online]. Available: https://github.com/GameOfLife/WFSCollider

[11] S. Wilson, "Spatial swarm granulation," in *Proceedings of the International Computer Music Conference*, Belfast, N. Ireland, 2008.

[12] D. Kim-Boyle, "Spectral spatialization - an overview," in *Proc. Int. Computer Music Conf*, Belfast, 2008.

[13] H. Vaggione, "Composing musical spaces by means of decorrelation of audio signals," in *Proceedings of the DAFx Conference on Digital Audio Effects*, 2001.

[14] G. S. Kendall, "The decorrelation of audio signals and its impact on spatial imagery," *Computer Music Journal*, vol. 19, no. 4, pp. 71–87, Dec. 1995.

[15] R. McGee, "Sound element spatializer," MS Thesis, University of California, 2010.

[16] D. Topper, M. Burtner, and S. Serafin, "Spatio-operational spectral (sos) synthesis," in *Proceedings of the International Conference on Digital Audio Effects*, Hamburg, Germany, 2002.

[17] "Image-based spatialization," in *Proceedings of the International Computer Music Conference*, Ljubljana, Slovenia, 2012, pp. 200–203.

[18] "Robin minard." [Online]. Available: http://robinminard.com/minard.content.php?id=17&sh=0

[19] "Edwin van der heide - pneumatic sound field." [Online]. Available: http://www.evdh.net/pneumatic_sound_field/

[20] "Signe." [Online]. Available: http://www3.sympatico.ca/qubeassm/Signe

[21] "Coincidence engine two." [Online]. Available: http://www.undefine.ca/en/projects/coincidence-engines/coincidence-engine-two-approximate-demarcator-of-constellations-in-other-cosmos/?artist=24

[22] J. McCartney, "Rethinking the computer music language: Super collider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, Dec. 2002.

[23] "Unit-lib." [Online]. Available: https://github.com/GameOfLife/Unit-Lib

[24] R. Bauer, “Distribution of points on a sphere with application to star catalogs,” *Journal of Guidance and Control*, vol. 23, no. 1, 2000.

[25] C. Elliott and P. Hudak, “Functional reactive animation,” in *ACM SIGPLAN Notices*, vol. 32, 1997, p. 263–273.

[26] C. M. Elliott, “Push-pull functional reactive programming,” in *Proceedings of the 2nd ACM SIGPLAN symposium on Haskell*, 2009, p. 25–36.

[27] “FPLib.” [Online]. Available: https://github.com/miguel-negrao/FPLib

[28] T. Lossius, P. Baltazar, and T. de La Hogue, “DBAP-distance-based amplitude panning,” in *Proc. Int. Computer Music Conf*, 2009, pp. 489–492.