SPList, a Waveset synthesis library and its usage in the composition "draussen"

Olaf Hochherz
Dresdenerstr. 122
Berlin, Germany, 10999
hochherz_olaf@gmx.de
http://www.shwobl.de/splists.html

Abstract

For my compositional work, I developed the SuperCollider library SPList for sample processing. SPList is a class which offers lists of wavesets and methods for their manipulation. The data can be, structured by grouping, restructured by sorting, and modulated. Combining these three types of operation makes it possible to shape sound within different scales of time, from microsound up to entire pieces of music.

Keywords

composition, granular synthesis, waveset synthesis, SuperCollider, non-realtime synthesis

1 Introduction

For my compositional work, I developed a concept for waveset based sample processing. My interest for wavesets began with the search for possibilities of algorithmic sample editing. Out of deliberations about the relationship between music, sound, and time came a SuperCollider library and a small C++ analysis programm. SPList means Structured Period Lists. In this article, I will describe my initial thoughts and how they resulted in the software and a composition.

2 More general thoughts

2.1 The basic ideas

In order to explain my compositional work with the SPList concept, I will have to demonstrate a few basic ideas of my work.

With tape music, the possibilities of controlling sound approach infinity. While looking for a personal conception of composition, I started to think about the possibilities of controling the sampled sound in a direct way. The idea of Xenakis[1] to develop algorithms, which control the movement of air pressure and not frequencies was a starting point. I like to think about samples as a repertoire of these movements. The common granular synthesis concepts did not fit into this idea. In granular synthesis the sound is interpretet as a grain, the dimensions and shape of a grain are mostly determined on a more abstract level by the composer, and not by the sound itself. Because I had no better idea about how the length of a grain could be determind by the sound itself, I chose the positive zerocrossing. I realised that this leads to the same type of sound manipulations as Trevor Wishart's Waveset Distortion[2]. There are a bunch of implementations of this concept, especially in the programs of the Composers Desktop Project and on a basic level in SuperCollider by Alberto de Campo [3]. The idea to use the system to create bigger/longer musical structurs made it necessary to make a new implementation. Another inspiration came from the ideas about concatenative synthesis [4], but I

refer to this idea only on a basic level and do not put any efforts into the implementation for this type of usage.

2.2 Continuity

The starting point of a composition of movements leads to a perspective on music in which sound is understood as a continuous flow. Musical distinctions such as figure, background, event, and rest are not products of direct placement, but of movement in time. A pause or silence does not result from the distance between events, but from the slowness of motion.

Sounds do not appear on a time axis and then vanish again, unlike notes on a score. A music in which things appear out of nothing requires a kind of imagination which I don't have.

2.3 Relations

Conveiving music as movement leads to an understanding in which one does not compose events, but the relationships between them. The point of interest does not lie on a sound by itself, but the way it emerges out of a certain context. Every sound results from movement which makes it possible. If a sound is changed, the relationships within the entire piece change with them.

2.4 Monophony

This one-sided view of music yields an approach of forming sounds in a monophonic way and also using the concept of “one voice” on a compositional level - either in that the whole composition actually has just one voice, or the materials which appear simultaneously form a union.

2.5 Treatment

I like to play with the connections between movements. I want to change the contexts of sounds and create new movements. The sounds perform such detailed movements that direct examination makes the composer lose overview, so I decided to leave the details to the computer.

When I began to divide the sound material into short sections, I found that there is a contradiction between continuity as the basis of time and progression on the one hand, and discrete sections on the other hand. But in perception, this division is nor quite so clear, new sounds and constellations come into existence. My sample processing methods make use of this contradiction.

3 Concept of the SPList

3.1 Waveset

The concept of wavesets was introduced by Trevor Wishart in Audible Design together with a couple of waveset transformations (he calls them waveset distortion). For waveset transformations, one takes a soundfile as a sequence of segments. The soundfile is subdivided each time the wave crosses the zero line from negative to positive. I will refer to these segments as “periods”.

3.1.1 Wave transformations

Transformations which directly affect the waveform of a period.

The algorithms suggested by Trevor Wishart can be summarized as follows:

1. reversal of the samples
2. modulating the play rate (shaking)
3. distortion effects (inversion, distortion, harmonic distortion)
4. modulating the volume (possibly muting)
5. substitution of the wave form with another wave form, sine saw etc.
6. merging of wavesets (averaging, transfer)
7. deletion of wavesets (time-shrinking)

3.1.2 Sequence transformations

Transformations affecting the sequence of the Periods. Trevor Wishart suggested:

1. swapping segments (shuffling)
2. interleaving two sequences
3. repeating segments (time-stretching)

3.1.3 Grouping algorithms

As a means of grouping periods, he proposes the possibility of treating a certain number of successive periods as one period.

The parameters for all these transformations mostly come from literal values or random functions.

4 Progress

I adopted this concept and extended it by three aspects:

1. The transformation parameter values can be specified for each single period. This way they can be derived from the properties of the period and its context.

2. There are more detailed possibilities of changing the order of periods and flexible ways to refer to individual areas of the soundfile.

3. The means of grouping periods are sensitive to the properties of the soundfile and can be applied recursively.

To achieve this, I have separated the processing task into an analysis phase and a synthesis / construction phase.

4.1 Analysis

A small C++ programm does the analysis, it reads a monophonic soundfile and writes a binary file with doubles.

For the analysis, a soundfile gets divided into a sequence of periods. Each period has a starting point in the soundfile and a certain number of samples. From the samples of a period, several values can be derived. These values have little musical meaning.

1. amp: maximum absolute sample value
2. duration: number of samples
3. rms: root mean square
4. peaks: number of local minima and maxima
5. delta: sum of the of the absolute differences between samples divided by the number of samples (gives a rough measure of treble)

The data of the resulting binary file can be loaded into the SuperCollider language.

4.2 The classes Period and SPList

The construction of new sounds is done with the SuperCollider language. The source soundfile can now be considerd as a monophonic sequences of periods. Each period is represented as an instance of the Period class. A Period object only contains a reference to the soundfile section and the analysed data and provides methods for sound modulation.

The Periods are contained by an instance of the SPList class. An SPList object can contain Periods as well as other SPLists and can itself be treated like a Period. This way it is possible to build complex structures using 4 different types of methods:

1. grouping a sequence of Periods into sublists (called groups), which are again SPLists.
2. accessing the analysed data of the Periods
3. reordering the periods (or groups) depending on analysis values. Reordering can apply at any level of nesting within an SPList, from the entire list down to any sublist within sublists.
4. applying transformations to periods or groups depending on the position in the structure and the analysis values.

In the following, I will use the term “section”, meaning either a Period or an SPList.

4.2.1 Grouping

Because a period is a very small section, the collection of periods becomes quite large (I often have 1000 periods per second). To keep certain aspects of a sound stable, it is nessesary to keep periods together by grouping them. This is

especially important when using a sorting algorithm. The grouping algorithms I use are:

1. grainThresh...: Looking through the analysed data and grouping while a given parameter value does not pass a threshhold. (upwards and /or downwards)
2. grainMin/grainMax: Looking through the analysed data, and beginning a new group whenever a given parameter value is bigger/smaller than the corresponding value of the surrounding periods.
3. splitN: Looking through the analysed data, and beginning a new group whenever a given parameter value is one of the *n* biggest/smallest values in the neighbourhood.

4.2.2 Accessing analysed data

The Period object contains the analysed parameters position, amplitude, duration, rms, peaks, delta. For the grouping algorithms it is important how the analysed information about a period can be accessed. SPList has methods to access this information and calculate values in different ways:

1. the average: avr(parameter)
2. the sum: sum(parameter)
3. the minimun: small(parameter)
4. the maximum: big(parameter)

4.2.3 Sequence transformations

By using SuperCollider, it was easy to provide standard array transformations like permutation, sorting, repetition, etc.. Sorting has a strong effect on the resulting sound because it creates a completely new context for each period or sublist.

4.2.4 Wave transformation

I provide most of the wave transformations presented by Trevor Wishart, except for distortion. It is easy to add transformations as SuperCollider SynthDefs.

4.3 Final Words on the Techniques Used

The SPList objects offer possibilities to treat sounds in the form of sets. The duration of a period is not a dominant parameter. However, the progression of periods is a crucial aspect for the sound and the composition, of course. The time range in which a transformation applies to the sound, can vary significantly. Depending on how the material is interpreted, transformations can refer to short as well as long segments. Therefore, an effect can have various consequences when it comes to perception. Repetitions can be heard as loops or time stretching. Transpositions may change the pitch or only the timbre. Level changes can produce silence or noise. They can affect continuous areas of the sound or individual periods here and there. As mentioned above, the results of this processing are monophonic audio files, which I have used in my compositions in two distinct ways so far.: a) layering b) concatenating. In an older work I used the second technique of concatenating different results, and in the piece “draussen” I used the technique of layering.

5 General examples

5.1 Sorting

The results of the sorting is predictable but differs by the amount and style of grouping:

load a file:

```
a=SPLSourceFile.newSPL(2,"example.wav");
```

group three times with grainMin with the average of the RMS of the periods:

```
3.do{a.grainMin(\rms,\avr)};
```

sort it from quiet to loud:

```
a.sort{|a,b|
a.avr(\amp)<b.avr(\amp)};
```

write a file:

```
a.write("testout.wav",96000);
```

The result is a crescendo with small grains. Each grain has a different size and there is no layering or envelope. For more interesting results, it is nice

to sort on different layers to get more complex structures

```
5.do{a.grainMin(\rms,\avr)};
a.sortBy(\amp).do{|i|
i.sortBy(\freq).do{|j|
j.sortBy(\delta).do{|k|
k.sortBy(\peaks)}}};
```

5.2 Transposition

Every transformation can be easily controlled by analysed data:

get smallest amplitude:

```
~as=a.small(\amp);
```

get biggest amplitude:

```
~ab=a.big(\amp);
```

perform on each period:

```
a.do{|i|
```

The amplitude of the Period i is used for the transposition factor. The value is mapped from a scale between the global minimum and maximum amplitude to a scale from 0.5 to 5.

```
i.transpose(i.amp.linlin(~as,~ab,
0.5,5));
};
```

5.3 Looping

This example shows a simple movement through a SPList with sublists and the different effects of looping sections of different sizes:

Group period and sublists as long as the list has more than five elements:

```
while({a.size.postln>5},
{a.grainMin(\freq,\avr)});
```

A recursive function which loops segments of different sizes depending on the RMS and the frequency:

```
f={|a|
a.do{|i|
if(i.avr(\rms)<a.avr(\rms),
{i.loop(i.avr(\freq).linlin(a.smal
l(\freq),a.big(\freq),10,0))},
{f.(i)});}};
```

6 "draussen" (German for outside)

In the piece "draussen", I used the the method of layering results of transformations. The source material was a recording of the sounds which emerge when an open mouth moves. I had different transformation scripts which are applied in varying succession. Each of the different processes had a defined role. There are three processes for sorting and two processes for repetition (looping). All these processes use the possibility of grouping periods to apply transformations to bigger sections. The three loop processes are specialised for sections with different volume (quiet, medium, loud). The number of repetitions is dynamic for each section. The sorting processes differ in the sorting style, from quiet to loud or from high to low frequencies. Also, the sorting processes the lists in different states of grouping. For example: a sorting is processed on a complete soundfile or only on small subsections.

The order in which the processes are used is important. Adjacent periods could end up in different parts of the piece after sorting. The details of the repetition can be destroyed by sorting ([1,2,3,1,2,3,1,2,3].sort -> [1,1,1,2,2,2,3,3,3]). These are effects which I have used in order to create certain sonorities or to break through beyond simple repetitive structures.

6.1 Sorting example

The following function sorts inside subsections. The idea was to sort only sections which are more quiet than others. These periods get sorted from low to high frequencies, so they produce short glissandi on different times in the soundfile. The length of the glissandi depends on the context of the segments. The whole soundfile gets split into 8 sections. A section only gets sorted if it is more quiet than the average amplitude of the complete soundfile but louder than the average amplitude of the 8 sections. In the other case the process is

repeated on each section, so the sections get smaller.

This function gets the complete SPList without subsections.

```
f={|spl|
var minamp=spl.small(\amp);
var maxamp=spl.big(\amp);
var pavr=spl.avr(\amp);
spl.splitN(3,\splitNextMax,
    70,\amp,\avr);
spl.do{|c|
    if((c.avr(\amp)<pavr)
    &&
    (c.avr(\amp)>(~avramp)),
    {c.sort{|a,b|
    a.avr(\freq)<b.avr(\freq)}},
    {f.(c)}
)}};
```

6.2 Timestretch example

Another example is the looping of quiet groups of periods, to stretch the sound with small repeating patterns. To achieve this I looped just quiet and low frequency period groups.

```
while({a.size>50},
{a.grainMin(\rms,\avr)});
 f={|a|
    var aamp=a.avr(\amp);
    var afreq=a.avr(\freq);
    var srms=a.small(\rms);
    var brms=a.big(\rms);
    a.do{|i|
 if((i.avr(\amp)<(aamp/2))
 &&(i.avr(\freq)<afreq
 ,{i.loop(i.avr(\rms)
    .linlin(srms,brms,6,1))},
 {f.(i)})}
 };
 f.(a);
```

6.3 Development in the piece

With the help of sorting it is truly simple to create developmental movements in sound. The effects of such a process become significant in “draussen”. The sorting of larger sections according to their average loudness creates a movement in the piece from quieter to somewhat louder, and from calm to somewhat more agitated.

Because the tension needed to be resolved after the crescendo, I decided to create a sort of coda. For this I used a section of the soundfiles from the piece, filtered and reversed.

7 Conclusion

The SPList is a highly specialised library, a small tool for rapid sound transformation with an interface which allows to describe strong transformations in a complex way. It is more an example for an approach about the style of control in music composition than a software library. SPList is an approach of analysis driven granular synthesis on a basic level. It shows in which way sorting algorithms can be used in music and what the results could be.

8 Acknowledgements

Thanks to Sefan Kreitmayer and Casey Mongoven.

References

[1] Xenakis Iannis. Formalized Music. Thought and Mathematics in Music. New York: Pendragon Press (Revised Edition), 1992.

[2] Trevor Wishart. Audible Design. York: Orpheus the Pantomime, 1994.

[3] Alberto de Campos implementation of Waveset Synthesis can be found in the SuperCollider swiki: http://swiki.hfbk-hamburg.de:8888/MusicTechnology/183

[4] Bob L. Sturm. Adaptive Concatenative Sound Synthesis and Its Application to Micromontage Composition, in Computer Musik Journal, Winter 2006, Vol. 30, No. 4, Pages 46-66.