The BBCut Library

Nick Collins

Proceedings of the International Computer Music Conference 2002

Current contact: http://www.cus.cam.ac.uk/~nc272/

Abstract

To facilitate work on automated breakbeat cutting it was expedient to establish a general framework promoting better code reusability. This framework is a publicly released collection of SuperCollider classes and help files called the BBCut Library. Whilst notionally for the cutting of breakbeat samples, its remit is much wider, into the use of algorithmic composition techniques to cut up any source audio.

The library is based upon a specific hierarchy of phrase/block/cut sufficient to implement a wide variety of cut procedures. Hierarchical information allows cut aware effects which can update parameters in coordination with rhythmic events.

The benefits of the library include the interchangeable use of any type of synthesis and source with any cut procedure. This makes it much simpler to write a new cut procedure which is immediately able to cut any target signal.

1. BACKGROUND

The BBCut Library began out of work on an algorithm to simulate the automatic cutting of breakbeats in the style of early jungle or drum and bass [Collins 2001a]. As the present author began to write further types of cut sequence generator they realised that they were continually repeating the basic nested spawn structure and cut synthesis code. The separation of cut synthesis from cut choice gave a much better software reuse, and a better paradigm for thinking about cut decisions.

The library is publicly available under the GNU General Public License and is a collection of SuperCollider 2 [McCartney 1998] classes with help files available from the author's web site quoted above. This paper cannot possibly go into great detail on every assumption and method of the library, but should provide a technical introduction in combination with the many help files and the commented code itself.

To set this work in the context of algorithmic composition research, let us borrow terms from [Pierce 2001], a paper which attempts to use neural nets to fashion some basic semiquaver resolution drum and bass drum loops. Existing cut procedures implemented in the library [Collins 2001b] are those of an 'active style synthesiser' rather than an 'empirical style modeller'. However, the library itself is neutral as regards what algorithmic composition methodology is utilised in cut procedures. It is a tool to assist research and composition, and one could imagine an implementation of a neural net trained on cut patterns from drum and bass classics as a NeuralNetCutProc.

There is little *academic* research on dance music or electronica, which are fast evolving current styles rather than 'dead' musics for dissection. Hence the practical experimental approach of this work. It is not enough to model early drum and bass without allowing extrapolations of new techniques.

The sort of audio cutting assumed here is usually at haptic or human rhythmic rates, with the obvious capability to reach inhuman speeds. In the main then, there is a macro level structural view rather than the microrhythms of granularisation [see Roads 1996 pp 180-184 especially]. That there is some overlap with granular techniques is evident though a very familiar effect of current electronica (AphexTwin, SquarePusher et al), that of using extremely fast iterated repeats of a small chunk of source (implemented in the BBCL's WarpCutProc). These repetitions played at audio rate speeds translate to specific pitches from a noisy wavetable. Further, set at faster inhuman tempi any cut procedure begins to lose its rhythmic sense and become more like textural granularisation. Human perceptual limits for this are around semiquavers at 250 bpm, see for instance the table of discrimination of the ear in [Pierce 2001].

2. A SUMMARY OF CAPABILITIES

This section lists some features and philosophies of the BBCut Library.

(i) Support for composers who wish to experiment with their own automatic cutting algorithms.

(ii) Separation of effects and synthesis from the algorithmic composition routine so as to allow any audio source to be cut by any cut procedure.

(iii) Effects processing on cuts is responsive to the hierarchical levels.

(iv) Everything works in realtime (this is SuperCollider after all!). The code has been tested over many months in live situations.

(v) SuperCollider is a very beautiful language; functions are easily passed as arguments making

much more general algorithm control possible. Scheduling in beats is masterminded by SuperCollider and smoothly copes with tempo changes.

(vi) Multiple synced BBCutters can complement each other very successfully in polyrhythmic patterns that are still aware of bars and beats.

(vii) MultiProc and MultiBBCutSynth classes allow the swapping of cut procedures and cut synthesis while running.

(viii) Extendable, open ended: everything is a SuperCollider class, derive your own classes to cover novel requirements.

Currently supported cut synthesis classes include the cutting of source soundfiles and other signal buffers (allowing offset information throughout the source), the cutting of a live stream like the current audio in (where offset information is only applicable to past events) and cut aware reverbs, envelopers, filters and panners. Cut procedures exist based on syncopated cuts, choosing a block or a cut at a time, using changing weights and statistical feedback [Ames 1990] and algorithms investigating recursion and even campanology [Collins 2001b]. There is also a simple automatic offset detection algorithm included to aid in dealing with human timing in beats and less regular rhythms. The BBCutSynthSFAO class supports user defined offset points (AO= Allowable Offset).

The distribution necessarily includes a number of auxiliary classes, including stream classes for Charles Ames' method of statistical feedback and campanology permutation chains.

3. PHRASES, BLOCKS AND CUTS

The BBCut Library helps to facilitate the process of writing cut procedures separately from synthesis code, but only within a restricted paradigm of phrase, block, cut described in figure 1. Further levels of hierarchy must be added by the user. The current levels have been shown sufficient for the variety of cut procedure experiments undertaken so far.

The phrase is the top level of the hierarchy, and corresponds in usual practice to a cut sequence lasting a number of measures. A block is a collection of repetitions of an atomic cut, at a common offset.

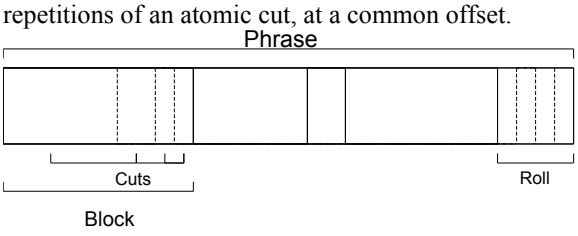

Figure 1 The Cut Hierachy

For example, for cuts being synthesised from the current audio in stream, a block might involve recording whilst playing a small chunk of audio and then repeating the stored buffer for as many instances as make up the block.

Cut procedures calculate on a per block basis, with a nested spawn. The outer spawn calls to the cut procedure to choose a new block. This choice then informs a finite number of repetitions of a spawn synthesising each cut. This master spawn structure is programmed in the BBCut class as the *ar class method. This pseudo code lists the main calls required:

```
(spawn each block)

BBCutProc object - choosenewblock

        This object may update the phrase if the last
        phrase has terminated. If necessary, updatephrase
        call to the BBCutSynth.

calls to inform and prepare BBCutSynth for a new block

(spawn carrying out the block)

BBCutSynth synthesises each cut in turn
```

The above intimately ties the BBCut, BBCutProc and BBCutSynth classes. To provide new cut procedures one derives from the BBCutProc class. To derive new effects which are aware of the hierarchy, and can update at phrase or block boundaries, or new methods of input audio, derive from BBCutSynth. Some intermediate classes are provided in the library to ease this process, but the basic understanding of the system can rest on these three classes.

4. WRITING A NEW BBCUTPROC CLASS

To demonstrate the Library in practical terms we shall walk through creating a new cut procedure, the ChooseBlockProc. This is a simplification of the specialised WarpCutProc from the library. It is so called because we shall literally choose the size of each block as we go, and worry about how many cuts for each block as a second stage. Appendix A contains the SuperCollider code for the class as reference.

The following is the bare code from the base class BBCutProc for the essential chooseblock method:

```
this.newPhraseAccounting;

//each phrase has one block
blocklength=currphraselength;          (*)
cuts=[blocklength];                    (**)

bbcutsynth.chooseoffset(phrasepos, beatspersubdiv,
currphraselength);

this.updateblock;
this.endBlockAccounting;
```

Helper methods newPhraseAccounting and endBlockAccounting are provided in the base class to take work away. The call to updateblock is another helper method wrapping the update call to the attached BBCutSynth. The offsets are chosen by the BBCutSynth using whatever method is appropriate- there is no need for the cut procedure to know how the BBCutSynth achieves this (there is facility for the cut procedure to select this, but we will not need that). The important central lines (*) and (**) work out the cuts Array, whose individual durations sum to the blocklength.

All that one needs to do for an arbitrary new chooseblock method is to generate the cuts Array in a chosen manner.

For the ChooseBlockProc there are two arbitrary auxiliary functions, (passed in as user parameters) the first for selecting a blocklength, the second for selecting the number of subdivisions (cuts) of that block. So we must replace (*) above with

```
blocklength= blocksizefunc.value
```

and (**) by

```
numcuts= numcutsfunc.value
cuts= Array of numcuts elements, each of size
blocklength/numcuts
```

The Appendix code implements exactly this. Figure 2 shows what the calls to the new procedure look like in SuperCollider. BBCutSynthParam's easysf method gives quick access to a typical BBCutsSynth chain for audio sample cutting. Note that any other BBCutSynth chain could be substituted without ChooseBlockProc needing modification.

```
(
{
var sf;
sf= SoundFiles3.new.init([":Sounds:floating_1",":Sounds:break"],[2,8]);

thisSynth.tempo_(2.4);     //144 bpm

BBCut.ar(BBCutSynthParam.easysf(sf),ChooseBlockProc.new)
+
BBCut.ar(BBCutSynthParam.easysf(sf,1),ChooseBlockProc.new({1.0},{4}))
}.play;
)
```

Figure 2 calls to the ChooseBlockProc class

5. THE BBCUTSYNTH

The BBCutSynth object used to synthesise cuts is in actual fact a nested set of BBCutSynth derived objects, with a method of obtaining audio at the innermost layer. Calls to update on phrase or block boundaries, and the synthesisecut method itself pass through the linked list establishing and applying the effects chain. It was considered whether to differentiate an audio collecting class (for example for the innermost layer, like BBCutSynthSF or BBCutSynthAudioIn from the library) from an effects class (like BBCutSynthParam, which does standard enveloping) but this was avoided as adding extra unnecessary complexity. There is a certain onus on the creator of a synthesis chain to get a few components in the right order, as regards source and enveloping, but this is seen as a necessary evil when it is considered that arbitrary effects can be established by mix and matching BBCutSynth derived classes, and all should work with any given BBCutProc.

6. CONCLUSIONS AND FUTURE WORK

In the establishing of a paradigm, restrictions are born. There is no claim here to represent the sole route to cutting up audio, just a methodology that has proven successful for breakbeat cutting. The composer who wishes to demand a close connection between synthesis parameters and cutting parameters may find it easier to program a very specific class rather than work within this library's assumptions. It is hoped that the situations that the library does cover are broad enough to give it some appeal as a shortcut.

The feedback of users will inform further work on the library. The invention of new cut procedures will naturally suggest revisions and better organisation.

As an immediate example, the library could be reworked to always generate [inter-onset duration, event duration, cut offset] lists on calls to chooseblock, converting blocks into a more general intermediary in the hierarchy. Blocks could then contain many different offset rolls for instance. This author errs on the side of avoiding such complexity in the current instance, assuming that overlaps are typically constant or proportional to inter onset duration. The block retains its same offset roll as the basic factor.

In the context of future cut procedures the previous paper gave a discussion of motifs as an additional hierarchical layer. Some revision of the hierarchy may inevitably lead to a BBCut Library 2.

Whilst the basic breakbeat cutter was adapted to Csound in an opcode version, the work described here is very much tied to the power of SuperCollider as an audio programming language.

Perhaps in the future the author will turn their hands to a Granular Cuts Library or similar, with much more emphasis on grain overlap. For the time being, there are a lot more algorithmic composition experiments within the BBCut Library to attempt.

REFERENCES

Ames, Charles. (1990). Statistics and Compositional Balance. Perspectives of New Music. 28:1.

Collins, Nick. (2001a). Algorithmic Composition Methods for Breakbeat Science. Proceedings of Music Without Walls, De Montfort University, June 21-23, 2001.

Collins, Nick. (2001b). Further Automatic Breakbeat Cutting Methods, Proceedings of Generative Art, Milan Politecnico, Dec 12-14, 2001.

McCartney, James (1998). Continued Evolution of the SuperCollider Real Time Synthesis Environment. Proceedings of the International Computer Music Conference, Ann Arbor, Michigan, 1998.

Pearce, M. T. and Wiggins, G. A. (2001). Towards a Framework for the Evaluation of Machine Compositions. In Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, (pp.22-32). Brighton, UK: SSAISB.

Pierce, J. (2001). Hearing in Time and Space. in Cook, P.R. (ed) (2001). Music, Cognition and Computerized Sound. MIT Press.

Roads, C. (1996) The Computer Music Tutorial. MIT Press.

APPENDIX A- CHOOSEBLOCKPROC.SC

```
ChooseBlockProc : BBCutProc
{
var blocksizefunc,numcutfunc,probs,accel;

//variables persisting between spawns
var beatsleft;

*new
{
arg blocksizefunc, numcutfunc, phraselength=12.0,
bpsd=0.5;

^super.new(bpsd,phraselength).
initChooseBlockProc(blocksizefunc,numcutfunc);
}

initChooseBlockProc
{
arg bsf,ncf;

blocksizefunc=bsf ?
{
//the calling procedure will automatically correct the retruned
//value if the choice is longer then beatsleft
arg left,length; [0.5,1,2].wchoose([0.5,0.4,0.1]);
};

numcutfunc=ncf ? {arg size;
if(size<1.0,{[4,8,16].choose},
{[8,16,32].choose})};
}




chooseblock
{
var repeats,temp;

//new phrase to calculate?
if(phrasepos>=currphraselength,
{this.newPhraseAccounting;});

beatsleft= currphraselength - phrasepos;

//always new slice/roll to calculate
blocklength=blocksizefunc.value
(beatsleft, currphraselength);

//safety to force conformity to phrases
if(blocklength>beatsleft,{blocklength= beatsleft;});

repeats=numcutfunc.value(blocklength);
temp=blocklength/repeats;
cuts=Array.fill(repeats,{temp});


//correction for arithmetic errors
cuts.put(repeats-1,temp+(blocklength-(temp*repeats)));

//offsets are now decided by the cut renderer
bbcutsynth.chooseoffset(phrasepos,
beatspersubdiv,currphraselength);

this.updateblock;

this.endBlockAccounting;
}

}
```