

**The Sonic Generator:
An Algorithmic Tool for Sound Elaboration**

**Adam Scott Neal
Georgia State University
Atlanta, GA
April 2, 2007 (rev. January 2008)**

1. Introduction

The Sonic Generator (named in honor of the new music ensemble at the Georgia Institute of Technology) is a simple batch program which directs several tasks using the executable programs Csound, Level, nGen, and Sox. The Sonic Generator will take a 1-second monaural sound file and generate one minute of music, following a predetermined form. This was originally created as part of my Master's comprehensive exam at Georgia State University in April 2007, but I have since revised it so that the results may be more varied.

This program may be used to create the concert work of the same name. For the concert work, three different sounds should be fed into the program to make a three-movement "suite." A presenter may use the examples I have made, but I believe it will be more interesting for him to use his own samples.

2. The Musical Structure

A user may customize elements of The Sonic Generator, since all of the Csound scores are included in the program folder. For this demonstration, the concert work as mentioned above, I have used a predetermined form. The piece contains five sections, the proportions of which are based upon the Fibonacci series, but with the order shuffled. Sections A and E are 5 seconds in length, section B is 10 seconds in length, section D is 15 seconds in length, and section C is 25 seconds in length.

Section A presents the unaltered sound file, followed by a rising figure constructed of small grains taken from the sound file, gradually pitch-shifted two octaves higher. Section B begins with a sonority constructed of sine waves, reading from bins of the phase vocoder analysis file. Following this are high pitches made with frequency modulation, also reading from bins in the phase vocoder analysis file but playing at randomly-selected times and randomly reading from a set of bins.

Section C is based upon subtractive synthesis. The Sonic Generator first creates a file called "pvocedit1," which edits the sound file, randomly selecting starting points, ending points, and durations. The result of this is fed into a filter bank, which creates pitches out of these small edits. The general effect of this section is a kind of rapid pizzicato, with the sonorities gradually descending in register.

In Section D, the sine wave sonority from Section B returns, this time much lower in pitch than the earlier sonority (and with different pitches), along with unfiltered edits of the sound file, much like those used for the subtractive synthesis section. These edits move lower in pitch and become longer in duration, with more time between attack points. Ring-modulated entries of the original sound enter as well, becoming faster as the section concludes. In Section E, the slowing and lowering edits continue, and the input sound is presented in full, albeit stretched to five seconds, pitch shifted down, and reversed.

3. The Program

The Sonic Generator is a batch file running in DOS, and it is a simple algorithm directing Csound, Level, nGen, and Sox. The Sonic Generator folder contains:

csound.exe	README.txt
level.exe	"data" folder
ngen.exe	"examples" folder
sox.exe	"tools" folder
run.bat	

The "tools" folder includes the scripts that run.bat requires:

additive.orc	mix.orc	pvocedits1.gen
additive1.gen	mix.sco	pvocedits2.gen
additive2.gen	pvoc.orc	pvocedits2.orc
fm.gen	pvoc1.sco	ringmod.gen
fm.orc	pvoc2.sco	ringmod.sco
granulated.gen	pvoc3.sco	subtractive.gen
granulated.orc	pvocedits.orc	subtractive.orc

The "data" folder contains nothing prior to the initial run of the program. It is used to hide the wave files produced prior to the final mix, "output.wav." These files will be overwritten in subsequent passes of the program.

The examples folder contains five example input files and their results. The results were converted to .mp3 format to save space, but the output when the program is run will be .wav format.

```
barkwheeze.wav, sonicgenerator_barkwheeze.mp3
drain.wav, sonicgenerator_drain.mp3
drums.wav, sonicgenerator_drums.mp3
onionchop.wav, sonicgenerator_onion.mp3
yellow.wav, sonicgenerator_yellow.mp3
```

The README.txt file explains that the user must do the following if Sonic Generator is to work correctly: he must place the folder directly in the C:/ directory, and he must place a 1-second monaural WAV file into the folder. Once this is done, he may click on "Run.bat," which automatically follows a sequence of tasks, culminating in the creation of "output.wav."

3.1 - Batch file

```
rename *.wav input.wav
csound -U pvanal -n 1024 -h 256 C:/sonicgenerator/c_input.wav
C:/sonicgenerator/input.pv
```

The first step renames the input file as "input.wav," as this is the name to which the following commands will refer. Next, Csound performs a phase vocoder analysis of the

input file, for use in later scores. The phase vocoder analysis captures frequency and amplitude data for certain segments of time. Here a new “snapshot” is taken every 256 samples, and each snapshot is a length of 1024 samples.

```
ngen C:\sonicgenerator\tools\pvocedit1.gen
C:\sonicgenerator\tools\pvocedit1.sco
ngen C:\sonicgenerator\tools\pvocedit2.gen
C:\sonicgenerator\tools\pvocedit2.sco
ngen C:\sonicgenerator\tools\granulated.gen
C:\sonicgenerator\tools\granulated.sco
ngen C:\sonicgenerator\tools\fm.gen C:\sonicgenerator\tools\fm.sco
ngen C:\sonicgenerator\tools\additive1.gen
C:\sonicgenerator\tools\additive1.sco
ngen C:\sonicgenerator\tools\additive2.gen
C:\sonicgenerator\tools\additive2.sco
ngen C:\sonicgenerator\tools\subtractive.gen
C:\sonicgenerator\tools\subtractive.sco
ngen C:\sonicgenerator\tools\ringmod.gen C:\sonicgenerator\tools\ringmod.sco
```

nGen produces eight scores to be used by Csound.

```
csound -W -o C:/sonicgenerator/pvoc1.wav C:\sonicgenerator\tools\pvoc.orc
C:\sonicgenerator\tools\pvoc1.sco
csound -W -o C:/sonicgenerator/pvoc3.wav C:\sonicgenerator\tools\pvoc.orc
C:\sonicgenerator\tools\pvoc3.sco
csound -W -o C:/sonicgenerator/pvocedit1.wav
C:\sonicgenerator\tools\pvocedit.orc C:\sonicgenerator\tools\pvocedit1.sco
sox pvocedit1.wav pvocedit1b.wav
level -l1 pvocedit1b.wav
csound -W -o C:/sonicgenerator/granulated.wav
C:\sonicgenerator\tools\granulated.orc C:\sonicgenerator\tools\granulated.sco
csound -W -o C:/sonicgenerator/additive1.wav
C:\sonicgenerator\tools\additive.orc C:\sonicgenerator\tools\additive1.sco
csound -W -o C:/sonicgenerator/fm.wav C:\sonicgenerator\tools\fm.orc
C:\sonicgenerator\tools\fm.sco
csound -W -o C:/sonicgenerator/subtractive.wav
C:\sonicgenerator\tools\subtractive.orc
C:\sonicgenerator\tools\subtractive.sco
csound -W -o C:/sonicgenerator/pvocedit2.wav
C:\sonicgenerator\tools\pvocedit2.orc C:\sonicgenerator\tools\pvocedit2.sco
csound -W -o C:/sonicgenerator/additive2.wav
C:\sonicgenerator\tools\additive.orc C:\sonicgenerator\tools\additive2.sco
csound -W -o C:/sonicgenerator/ringmod.wav
C:\sonicgenerator\tools\ringmod.orc C:\sonicgenerator\tools\ringmod.sco
csound -W -o C:/sonicgenerator/pvoc2.wav C:\sonicgenerator\tools\pvoc.orc
C:\sonicgenerator\tools\pvoc2.sco
```

Csound compiles the different sections of the final piece separately, in the order they will appear in the final mix (this allows the user to go to the “data” folder to isolate elements). Although pvoc1, pvoc3, and pvocedit1 will not be heard, they are used by subsequent scores.

```
sox granulated.wav granulatedb.wav
sox additive1.wav additive1b.wav
sox fm.wav fmb.wav
sox subtractive.wav subtractiveb.wav
sox pvocedits2.wav pvocedits2b.wav
sox additive2.wav additive2b.wav
sox pvoc2.wav pvoc2b.wav
sox ringmod.wav ringmodb.wav
```

Sox renames the files. This was done because the next program, Level, did not recognize the headers on the WAV files produced by Csound.

```
level -l1 granulatedb.wav
level -l1 additive1b.wav
level -l1 fmb.wav
level -l1 subtractiveb.wav
level -l1 pvocedits2b.wav
level -l1 additive2b.wav
level -l1 pvoc2b.wav
level -l1 ringmodb.wav
```

Level compresses the sound files for the different sections of the piece. This ensures that the files are at an optimum level, not too soft, but not clipped.

```
csound -W -o C:/sonicgenerator/mix.wav C:\sonicgenerator\tools\mix.orc
C:\sonicgenerator\tools\mix.sco
```

Next, Csound creates a mix of the different parts, placing them at predetermined points in time and at predetermined dynamic levels.

```
sox mix.wav mixb.wav
level -n -l1 mixb.wav
sox c_mixb.wav output.wav
```

Level performs a final normalization of the mix (Sox renames the mix, as explained above). Since Level appends a prefix “c_” to the beginning of its resultant files, Sox is used to rename the file to the more elegant “output.wav”

```
move C:\sonicgenerator\input.wav C:\sonicgenerator\data\input.wav
move C:\sonicgenerator\c_input.wav C:\sonicgenerator\data\c_input.wav
move C:\sonicgenerator\input2.wav C:\sonicgenerator\data\input2.wav
move C:\sonicgenerator\input.pv C:\sonicgenerator\data\input.pv
move C:\sonicgenerator\fm.sco C:\sonicgenerator\data\fm.sco
move C:\sonicgenerator\granulated.sco C:\sonicgenerator\data\granulated.sco
move C:\sonicgenerator\pvocedits1.sco C:\sonicgenerator\data\pvocedits1.sco
move C:\sonicgenerator\pvocedits2.sco C:\sonicgenerator\data\pvocedits2.sco
move C:\sonicgenerator\pvoc1.wav C:\sonicgenerator\data\pvoc1.wav
move C:\sonicgenerator\pvoc3.wav C:\sonicgenerator\data\pvoc3.wav
move C:\sonicgenerator\pvocedits1.wav C:\sonicgenerator\data\pvocedits1.wav
move C:\sonicgenerator\pvocedits1b.wav
C:\sonicgenerator\data\pvocedits1b.wav
```

```

move C:\sonicgenerator\c_pvocedit1b.wav
C:\sonicgenerator\data\c_pvocedit1b.wav
move C:\sonicgenerator\granulated.wav C:\sonicgenerator\data\granulated.wav
move C:\sonicgenerator\granulatedb.wav
C:\sonicgenerator\data\granulatedb.wav
move C:\sonicgenerator\c_granulatedb.wav
C:\sonicgenerator\data\c_granulatedb.wav
move C:\sonicgenerator\additive1.wav C:\sonicgenerator\data\additive1.wav
move C:\sonicgenerator\additive1b.wav C:\sonicgenerator\data\additive1b.wav
move C:\sonicgenerator\c_additive1b.wav
C:\sonicgenerator\data\c_additive1b.wav
move C:\sonicgenerator\fm.wav C:\sonicgenerator\data\fm.wav
move C:\sonicgenerator\fmb.wav C:\sonicgenerator\data\fmb.wav
move C:\sonicgenerator\c_fmb.wav C:\sonicgenerator\data\c_fmb.wav
move C:\sonicgenerator\subtractive.wav
C:\sonicgenerator\data\subtractive.wav
move C:\sonicgenerator\subtractiveb.wav
C:\sonicgenerator\data\subtractiveb.wav
move C:\sonicgenerator\c_subtractiveb.wav
C:\sonicgenerator\data\c_subtractiveb.wav
move C:\sonicgenerator\pvocedit2.wav C:\sonicgenerator\data\pvocedit2.wav
move C:\sonicgenerator\pvocedit2b.wav
C:\sonicgenerator\data\pvocedit2b.wav
move C:\sonicgenerator\c_pvocedit2b.wav
C:\sonicgenerator\data\c_pvocedit2b.wav
move C:\sonicgenerator\additive2.wav C:\sonicgenerator\data\additive2.wav
move C:\sonicgenerator\additive2b.wav C:\sonicgenerator\data\additive2b.wav
move C:\sonicgenerator\c_additive2b.wav
C:\sonicgenerator\data\c_additive2b.wav
move C:\sonicgenerator\pvoc2.wav C:\sonicgenerator\data\pvoc2.wav
move C:\sonicgenerator\pvoc2b.wav C:\sonicgenerator\data\pvoc2b.wav
move C:\sonicgenerator\c_pvoc2b.wav C:\sonicgenerator\data\c_pvoc2b.wav
move C:\sonicgenerator\ringmod.wav C:\sonicgenerator\data\ringmod.wav
move C:\sonicgenerator\ringmodb.wav C:\sonicgenerator\data\ringmodb.wav
move C:\sonicgenerator\c_ringmodb.wav C:\sonicgenerator\data\c_ringmodb.wav
move C:\sonicgenerator\mix.wav C:\sonicgenerator\data\mix.wav
move C:\sonicgenerator\mixb.wav C:\sonicgenerator\data\mixb.wav
move C:\sonicgenerator\c_mixb.wav C:\sonicgenerator\data\c_mixb.wav
: end

```

The batch file now moves all of the intermediary WAV to the “data” folder. This way, the user will not have to hunt for “output.wav,” and they can more easily delete these files without fear of deleting one of the essential programs or scripts. I did not delete the files for troubleshooting purposes, and also because a user may want to use a certain portion of the result instead of the entire result.

3.2 - Csound files

```

;Additive.orc
;The Sonic Generator - Additive Synthesis Components
sr      = 44100
kr      = 4410
ksmps  = 10

```

```

nchnls = 1

instr 1 ; Pvread instrument for Additive Synthesis
ibeg = p4 ;beginning timepoint in phase vocoder analysis file
iend = p5 ;ending timepoint in phase vocoder analysis file
ibin = p6 ;analysis bin to be played
iamp = p7 ;amplitude adjustment
kclick linseg 0, p3*.01, iamp, p3*.98, iamp, p3*.01, 0
ktime line ibeg, p3, iend ;reads timepoint from phase vocoder analysis file
kfreq, kamp pvread ktime, "C:/sonicgenerator/input.pv", ibin
asine oscili kamp, kfreq*.5, 1
out asine*kclick
endin

```

The additive synthesis instrument takes the phase vocoder analysis file, then plays back only one component sine wave from the sound for each note event. nGen files generate two scores for this instrument, used in sections B and D. For each pass of the program, the start times remain the same, but the bins from which the instrument reads will be randomly selected.

```

;FM.orc
;The Sonic Generator - Frequency Modulation Component
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1 ; pvread-controlled FM Synthesizer, modified from Karpen
ibeg = p4 ;beginning timepoint in phase vocoder analysis file
iend = p5 ;ending timepoint in phase vocoder analysis file
ibin = p6 ;analysis bin to be played
iamp = p7 ;amplitude adjustment
ipan = p8 ;stereo pan position
ktime line ibeg, p3, iend ;reads timepoint from phase vocoder analysis file
kindex expon 3, p3, .001 ;controls FM index
kfreq, kamp pvread ktime, "C:/sonicgenerator/input.pv", ibin
amod oscili kfreq*kindex, kfreq, 1
acar oscili kamp, kfreq+amod, 1
acar = acar*iamp
outs acar*ipan, acar*(1-ipan)
endin

instr 2 ; pvread-controlled FM Synthesizer, modified from Karpen
ibeg = p4 ;beginning timepoint in phase vocoder analysis file
iend = p5 ;ending timepoint in phase vocoder analysis file
ibin = p6 ;analysis bin to be played
iamp = p7 ;amplitude adjustment
ipan = p8 ;stereo pan position
ktime line ibeg, p3, iend ;reads timepoint from phase vocoder analysis file
kindex expon 3, p3, .001 ;controls FM index
kfreq, kamp pvread ktime, "C:/sonicgenerator/input.pv", ibin
amod oscili kfreq*kindex, kfreq, 1
acar oscili kamp, kfreq+amod, 1
acar = acar*iamp
outs acar*ipan, acar*(1-ipan)
endin

```

Much like the additive synthesis instrument, these frequency modulation instruments are affected mostly by the amplitude information in the analysis file. In addition, another sine wave modulates the main sine wave, creating more sidebands (more frequencies) and therefore a more complex timbre. An extra envelope, “kindex,” controls the FM index, so that the sound is brighter (more sidebands/frequencies) at the attack and becomes darker (less sidebands/frequencies) over the duration of the note.

```
;pvoc.orc
;The Sonic Generator - Phase Vocoder Components
  sr      = 44100
  kr      = 4410
  ksmps   = 10
  nchnls  = 1

instr 1
idur = p3 ;stretch duration
kenv  linseg 0, p3*.01, 1, p3*.98, 1, p3*.01, 0
ktime line p6, idur, p7
kpit  line p4, idur, p5
asig  pvoc ktime, kpit, "C:/sonicgenerator/input.pv"
      out asig*kenv
      endin

;pvoc1.sco
;ins  strt  dur  pit1  pit2  tpst  tpend
i1   0     10   1     1     0     1
e

;pvoc2.sco
;inst strt  dur  pit1  pit2  tpst  tpend
i1   0     5    .5    .4    1     0
e

;pvoc3.sco
;inst strt  dur  pit1  pit2  tpst  tpend
i1   0     10   .5    .3    0     1
e
```

The three pvoc files can stretch the original 1-second file, change its pitch, and play it forward or backward. “Dur” (duration) shows how much the file has been stretched, “Pit1” and “Pit2” show the pitch for the beginning and end (1 is normal, .5 is one octave down). “Tpst” is the time pointer starting point, and “Tpend” is the time pointer ending point. These can be set anywhere within the .pv file, so starting at 1 and going to 0 means playing the file backwards, while starting at .5 and going to 1 would play the second half, forward.

```
;pvocedit.orc
;The Sonic Generator - Sound File Player for algorithmic edits, no pitch shifting
  sr      = 44100
  kr      = 4410
  ksmps   = 10
  nchnls  = 2

instr 1
ipan   = p5
```

```

kenv  linseg 0, p3/3, 1, p3/3, 1, p3/3, 0
asig  soundin "C:/sonicgenerator/pvoc1.wav", p4
asig  = asig*kenv
asig  dam asig, 7500, 1.5, 1.5, .1, .1
      outs asig*ipan, asig*(1-ipan)
      endin

instr 2
ipan  = p5
kenv  linseg 0, p3/3, 1, p3/3, 1, p3/3, 0
asig  soundin "C:/sonicgenerator/pvoc1.wav", p4
asig  = asig*kenv
asig  dam asig, 7500, 1.5, 1.5, .1, .1
      outs asig*ipan, asig*(1-ipan)
      endin

```

The pvocedit's orchestra is a sound file player. Its dynamic range is expanded using the “dam” opcode, and each note is given a trapezoidal envelope to prevent clicks at the beginning and end of each note.

```

;subtractive.orc
;The Sonic Generator - Subtractive Synthesis components
  sr      = 44100
  kr      = 4410
  ksmpls  = 10
  nchnls  = 2

gasendl  init 0
gasendr  init 0

instr 1                                     ;Soundfile Player
asigl, asigr soundin "C:/sonicgenerator/c_pvocedit's1b.wav", 0, 4
gasendl = asigl
gasendr = asigr
      outs asigl*0, asigr*0
      endin

      instr 2                               ;Filter Bank
kclick linseg 0, p3*.1, 1, p3*.8, 1, p3*.1, 0
kband  linseg 5, p3*.5, 10, p3*.5, 5
asig1l butbp gasendl, cspch(p5), kband
asig2l butbp gasendl, cspch(p6), kband
asig3l butbp gasendl, cspch(p7), kband
asig4l butbp gasendl, cspch(p8), kband
asig5l butbp gasendl, cspch(p9), kband
asig6l butbp gasendl, cspch(p10), kband

asig1r butbp gasendr, cspch(p5), kband
asig2r butbp gasendr, cspch(p6), kband
asig3r butbp gasendr, cspch(p7), kband
asig4r butbp gasendr, cspch(p8), kband
asig5r butbp gasendr, cspch(p9), kband
asig6r butbp gasendr, cspch(p10), kband

amixl  = (asig1l+asig2l+asig3l+asig4l+asig5l+asig6l)*kclick
amixr  = (asig1r+asig2r+asig3r+asig4r+asig5r+asig6r)*kclick

      outs amixl*p4, amixr*p4
      endin

```

The subtractive synthesis instrument plays a soundfile (pvocedit1, from above), sends the output through a filter, but not to the final mix. The filter bank removes frequency information from all but six frequencies, creating a series of six-voiced chords. For interest, the bandwidth of the filters expands and contracts to allow more frequencies to come through, bringing out more of the original character of the input sound. The “cspch” opcode is used to aid in inputting pitches, so that one can read in pitch class numbers.

```
;mix.orc
sr      = 44100
kr      = 4410
ksmps  = 10
nchnls = 2

strset 1, "C:/sonicgenerator/input.wav"
strset 2, "C:/sonicgenerator/c_granulatedb.wav"
strset 3, "C:/sonicgenerator/c_additive1b.wav"
strset 4, "C:/sonicgenerator/c_fmb.wav"
strset 5, "C:/sonicgenerator/c_subtractiveb.wav"
strset 6, "C:/sonicgenerator/c_pvocedit2b.wav"
strset 7, "C:/sonicgenerator/c_additive2b.wav"
strset 8, "C:/sonicgenerator/c_ringmodb.wav"
strset 9, "C:/sonicgenerator/c_pvoc2b.wav"

instr 1
kamp linseg 0, p5, p6, p7, p6, p8, 0
asig diskin p4, 1
outs asig*kamp, asig*kamp
endin

instr 2
kamp linseg 0, p5, p6, p7, p6, p8, 0
asigl, asigr diskin p4, 1
outs asigl*kamp, asigr*kamp
endin

;mix.sco
;P1 P2 P3 P4 P5 P6 P7 P8
;INSTR STRT DUR FILE ENVDUR1 AMP ENVDUR2 ENVDUR3
i1 0 1 1 1 .01 .75 .98 .01 ;original
i2 1 5 2 .01 .5 3.99 1 ;granulated rising
i1 5 15 3 .01 .25 9.99 5 ;additive 1
i2 5 15 4 1 .25 7 7 ;pvread fm
i2 15 25 5 1 .5 21 3 ;subtractive
i2 35 25 6 20 .5 4.99 .01 ;pvoc edits
i1 30 30 7 .01 .5 29.98 .01 ;additive 2
i2 32 25 8 .01 .75 24.98 .01 ;ring modulation
i1 55 5 9 .01 .75 4.98 .01 ;original, backwards
e
```

The mix file reads in the compressed versions of the WAV files created previously by Csound. The score shows the start times for each component, as well as the amplitude for the files. To prevent clicking at the beginning and end of the note statements, an envelope has been added, so that each file begins and ends at a level of 0. Over the

duration determined by P5, the sound will fade in to the level determined by P6, stay at that level for the duration of P7, then fade out for the duration of P8. This provided flexibility for some files to seemingly play at full volume for their duration (a .01 second fade out is nearly imperceptible but necessary to prevent clicks), and for some to fade in and out more gradually.

3.3 - nGen files

The remaining score files are generated by nGen. A few of them create many note events, so I will only include examples of “granulated.sco” and “subtractive.sco” as a demonstration of nGen's results.

3.3.1 nGen primer

nGen uses its own script to generate score files. As shown below, a line of code is written for each p-field. Items following a greater-than sign (such as the function table: >f1 0 16384 10 1) are placed within the score file verbatim. “rs” stands for “random seed,” and this can (and for that matter, must) be changed to yield different results. The instrument field (such as “i1 = 8 0 15”) determines several factors. The first number shows the number of p-fields to be used. The second number shows the start time, and the final number shows the duration in seconds.

The “ra” command (shown in p2: ra(15, .75 [.05 .2] .25 [.3 .7])) is the random command. This line instructs nGen to create note events with start times x seconds apart. The example is weighted so that the quicker onset of note events occur 75% of the time, and the slower onset occurs 25% of the time. The “se” command (se(15, 1. [65 67 69 71 73 75])) is the set command. For this line, nGen randomly chooses (at an even distribution) the six numbers provided. The “mo” command moves over the course of the file from one number or range to another.

3.3.2 examples

```
;additive1.gen
>f1 0 16384 10 1 ;creates sine wave
rs(3197)
i1 = 7 0 -9
{
;start
  p2    .1, .1, .1, .1, .1, .1, .1, .1, .1
;duration
  p3    406, 407, 408, 409, 410, 411, 412, 413, 415
;timepoint start
  p4    0
;timepoint end
  p5    1
;bin#
  p6    se(-1 .7 [20 21 22 23 24] .3 [25 26 27 28 29])
        se(-1 .7 [25 26 27 28 29] .3 [30 31 32 33 34])
        se(-1 .7 [30 31 32 33 34] .3 [35 36 37 38 39])
        se(-1 .7 [35 36 37 38 39] .3 [40 41 42 43 44])
        se(-1 .7 [40 41 42 43 44] .3 [45 46 47 48 49])
```

```

se(-1 .7 [45 46 47 48 49] .3 [50 51 52 53 54])
se(-1 .7 [50 51 52 53 54] .3 [55 56 57 58 59])
se(-1 .7 [55 56 57 58 59] .3 [60 61 62 63 64])
se(-1 .7 [60 61 62 63 64] .3 [65 66 67 68 69])
;amp
p7 200
}

```

For both `additive1.gen` and `additive2.gen`, the start times and durations are predetermined, but the instrument will select from the weighted sets of bins. By changing the random seed (“rs(xxxx)”), the user can create different sonorities each time.

```

;fm.gen
>f1 0 16384 10 1
rs(2525)
i1 = 8 0 15
{
;start
p2 ra(15, .75 [.05 .2] .25 [.3 .7])
;duration
p3 ra(15, 1. [5 9])
;timepoint start
p4 0
;timepoint end
p5 1
;bin #
p6 se(5, .3 [65 66 67 68 69 70] .7 [70 71 72 73 74 75])
se(5, 1. [70 71 72 73 74 75])
se(5, .7 [65 66 67 68 69 70] .3 [70 71 72 73 74 75])
;amp
p7 500
;pan
p8 ra(15, 1. [0 .5])
}

rs(2626)
i2 = 8 0 15
{
;start
p2 ra(15, .75 [.05 .2] .25 [.3 .7])
;duration
p3 ra(15, 1. [5 9])
;timepoint start
p4 0
;timepoint end
p5 1
;bin #
p6 se(5, .3 [65 66 67 68 69 70] .7 [70 71 72 73 74 75])
se(5, 1. [70 71 72 73 74 75])
se(5, .7 [65 66 67 68 69 70] .3 [70 71 72 73 74 75])
;amp
p7 500
;pan
p8 ra(15, 1. [.5 1])
}

```

Over the course of 15 seconds, the start times are randomly selected, 75% of the time occurring between .05 and .2 seconds apart, and 25% of the time occurring between .3

and .7 seconds apart. The note lengths are also randomly selected, lasting from 5 to 9 seconds each. Each note will read from the very beginning to the very end of the .pv file, regardless of length, and it will play back on of the weighted sets of bins determined by p6. The amplitude is constant at 500, though this can be altered. To create more density, there are 2 instruments, one controlling events panned generally to the left, and one controlling events panned generally to the right (the exact pan placements are randomly selected).

```
;granulated.gen
rs(1535)
i1 = 6 0 5
{
;start
  p2   ra(5, 1. [.01 .25])
;duration
  p3   mo(5, 1. 400.3 400.05)
;skip time
  p4   mo(5, 1. 0 .9)
;pitch
  p5   mo(5, 1. 2 8)
;pan
  p6   ra(5, 1. [.5 1])
}

rs(1536)
i2 = 6 0 5
{
;start
  p2   ra(5, 1. [.01 .25])
;duration
  p3   mo(5, 1. 400.3 400.05)
;skip time
  p4   mo(5, 1. 0 .9)
;pitch
  p5   mo(5, 1. 2 8)
;pan
  p6   ra(5, 1. [0 .5])
}
```

Over the course of 5 seconds, start times occur between .01 and .25 seconds apart, and the note durations gradually become shorter, from .3 seconds to .05 seconds, making the texture more sparse at the end. The notes gradually read through the file, and the pitch rises 2 octaves. To create more density, there are 2 instruments, one controlling events panned generally to the left, and one controlling events panned generally to the right (the exact pan placements are randomly selected). Changing the random seeds will yields slightly different results, but this nGen file produced the following score file:

```
;granulated.sco
;I-block #1 (i1):
i1  0.000 0.300 0.000 2.000 0.633
i1  0.228 0.293 0.024 2.162 0.866
i1  0.474 0.286 0.049 2.324 0.856
i1  0.677 0.280 0.073 2.486 0.668
i1  0.847 0.273 0.097 2.649 0.557
```

i1	1.001	0.266	0.122	2.811	0.989
i1	1.213	0.259	0.146	2.973	0.984
i1	1.398	0.253	0.170	3.135	0.592
i1	1.628	0.246	0.195	3.297	0.969
i1	1.731	0.239	0.219	3.459	0.962
i1	1.847	0.232	0.243	3.622	0.836
i1	1.998	0.226	0.268	3.784	0.780
i1	2.109	0.219	0.292	3.946	0.515
i1	2.133	0.212	0.316	4.108	0.691
i1	2.329	0.205	0.341	4.270	0.889
i1	2.398	0.199	0.365	4.432	0.884
i1	2.530	0.192	0.389	4.595	0.998
i1	2.556	0.185	0.414	4.757	0.816
i1	2.724	0.178	0.438	4.919	0.629
i1	2.746	0.172	0.462	5.081	0.977
i1	2.963	0.165	0.486	5.243	0.573
i1	3.037	0.158	0.511	5.405	0.732
i1	3.060	0.151	0.535	5.568	0.982
i1	3.147	0.145	0.559	5.730	0.794
i1	3.291	0.138	0.584	5.892	0.921
i1	3.418	0.131	0.608	6.054	0.602
i1	3.665	0.124	0.632	6.216	0.878
i1	3.698	0.118	0.657	6.378	0.593
i1	3.880	0.111	0.681	6.541	0.774
i1	4.093	0.104	0.705	6.703	0.921
i1	4.284	0.097	0.730	6.865	0.971
i1	4.364	0.091	0.754	7.027	0.822
i1	4.491	0.084	0.778	7.189	0.948
i1	4.603	0.077	0.803	7.351	0.975
i1	4.621	0.070	0.827	7.514	0.766
i1	4.693	0.064	0.851	7.676	0.570
i1	4.792	0.057	0.876	7.838	0.768
i1	4.877	0.050	0.900	8.000	0.759

;I-block #2 (i2):

i2	0.000	0.300	0.000	2.000	0.488
i2	0.226	0.293	0.025	2.167	0.291
i2	0.310	0.286	0.050	2.333	0.259
i2	0.431	0.279	0.075	2.500	0.352
i2	0.679	0.272	0.100	2.667	0.281
i2	0.734	0.265	0.125	2.833	0.478
i2	0.976	0.258	0.150	3.000	0.163
i2	1.180	0.251	0.175	3.167	0.072
i2	1.232	0.244	0.200	3.333	0.471
i2	1.308	0.237	0.225	3.500	0.197
i2	1.338	0.231	0.250	3.667	0.182
i2	1.474	0.224	0.275	3.833	0.096
i2	1.680	0.217	0.300	4.000	0.023
i2	1.836	0.210	0.325	4.167	0.268
i2	2.049	0.203	0.350	4.333	0.371
i2	2.213	0.196	0.375	4.500	0.400
i2	2.410	0.189	0.400	4.667	0.164
i2	2.462	0.182	0.425	4.833	0.280
i2	2.578	0.175	0.450	5.000	0.211
i2	2.822	0.168	0.475	5.167	0.009
i2	3.045	0.161	0.500	5.333	0.352
i2	3.271	0.154	0.525	5.500	0.108
i2	3.311	0.147	0.550	5.667	0.200
i2	3.553	0.140	0.575	5.833	0.371
i2	3.701	0.133	0.600	6.000	0.298

```

i2  3.828  0.126    0.625    6.167    0.339
i2  3.927  0.119    0.650    6.333    0.479
i2  3.960  0.112    0.675    6.500    0.311
i2  4.025  0.106    0.700    6.667    0.025
i2  4.196  0.099    0.725    6.833    0.290
i2  4.269  0.092    0.750    7.000    0.103
i2  4.355  0.085    0.775    7.167    0.416
i2  4.448  0.078    0.800    7.333    0.269
i2  4.617  0.071    0.825    7.500    0.079
i2  4.758  0.064    0.850    7.667    0.365
i2  4.866  0.057    0.875    7.833    0.410
i2  4.969  0.050    0.900    8.000    0.197
e

```

```

;pvocedits1.gen
rs(1533)
i1 = 5 0 25
{
;start
  p2  mo(25, 1.E [.05 .1] [.3 .5])
;duration
  p3  mo(25, 1.E [400.01 400.05] [400.1 400.4])
;skip time
  p4  ra(25, 1. [.1 8.9])
;pan
  p5  ra(25, 1. [.5 1])
}

```

```

rs(1534)
i2 = 5 0 25
{
;start
  p2  mo(25, 1.E [.05 .1] [.3 .5])
;duration
  p3  mo(25, 1.E [400.01 400.05] [400.1 400.4])
;skip time
  p4  ra(25, 1. [.1 8.9])
;pan
  p5  ra(25, 1. [0 .5])
}

```

Over the course of 25 seconds, the note events gradually become slower (.05-.1 seconds apart to .3-.5 seconds apart) but also longer (.01-.05 seconds long to .1-.4 seconds long). The notes read from the previously-generated file, randomly selecting starting points between .1 and 8.9 seconds into the file. To create more density, there are 2 instruments, one controlling events panned generally to the left, and one controlling events panned generally to the right (the exact pan placements are randomly selected).

```

;pvocedits2.gen
rs(1535)
i1 = 5 0 25
{
;start
  p2  ms(25, 1. [.01 .25] [.5 1])
;duration
  p3  mo(25, 1. 400.05 400.5)
;skip time
  p4  mo(25, 1. 0 9.5)
}

```

```

;pan
  p5  ra(25, 1. [.5 1])
}

rs(1536)
i2 = 5 0 25
{
;start
  p2  ms(25, 1. [.01 .25] [.5 1])
;duration
  p3  mo(25, 1. 400.05 400.5)
;skip time
  p4  mo(25, 1. 0 9.5)
;pan
  p5  ra(25, 1. [0 .5])
}

```

This nGen file works in a similar manner to pvocredits1, except that the start times and durations become greater, and the durations move from exactly .05 seconds to exactly .5 seconds, rather than moving from one range to another. Most importantly, the skip times gradually read through the file instead of randomly selecting start points at all times.

```

;ringmod.gen
>f1 0 16384 10 1

rs(1981)
i1 = 6 0 25
{
;start
  p2  ms(25, 1. [1.5 2 3 4] [.25 .5 1])
;duration
  p3  401
;amp
  p4  ms(25, 1. [.3 .4 .5] [.5 .6 .75])
;frequency
  p5  ra(15, 1. [100 6000])
      ra(10, 1. [100 1000])
;pan
  p6  ra(25, 1. [.3 .7])
}

```

For the ringmod instrument, the start times begin at intervals between 1.5 and 4 seconds, gradually moving faster, to intervals between .25 and 1 second. The duration always remains the same, as this is ring modulating the original 1-second sound. The amplitude gradually increases, and the modulating frequencies begin with a fairly wide range (between 100 and 6000 Hz), moving to a more narrow range (100 to 1000 Hz).

```

;subtractive.gen
>i1 0 25

rs(8558)
i2 = 10 0 25
{
;start
  p2  .75, .75, 1.5, 2.3, 3.8, 6.2
;duration

```

```

    p3      400.8, 401.6, 401.6, 402.4, 403.9, 406.2, 410.2
;amp
    p4      2
;frq1
    p5      se(-1 .9 [11.00 11.01 11.02 11.03] .1 [11.06 11.07 11.08 11.09])
            se(-1 .9 [10.06 10.07 10.08 10.09] .1 [11.00 11.01 11.02 11.03])
            se(-1 .9 [10.00 10.01 10.02 10.03] .1 [10.06 10.07 10.08 10.09])
            se(-1 .9 [9.06 9.07 9.08 9.09] .1 [10.00 10.01 10.02 10.03])
            se(-1 .9 [9.00 9.01 9.02 9.03] .1 [9.06 9.07 9.08 9.09])
            se(-1 .9 [8.06 8.07 8.08 8.09] .1 [9.00 9.01 9.02 9.03])
;frq2
    p6      se(-1 .9 [11.02 11.03 11.04 11.05] .1 [11.08 11.09 11.10 11.11])
            se(-1 .9 [10.08 10.09 10.10 10.11] .1 [11.02 11.03 11.04 11.05])
            se(-1 .9 [10.02 10.03 10.04 10.05] .1 [10.08 10.09 10.10 10.11])
            se(-1 .9 [9.08 9.09 9.10 9.11] .1 [10.02 10.03 10.04 10.05])
            se(-1 .9 [9.02 9.03 9.04 9.05] .1 [9.08 9.09 9.10 9.11])
            se(-1 .9 [8.08 8.09 8.10 8.11] .1 [9.02 9.03 9.04 9.05])
;frq3
    p7      se(-1 .9 [11.04 11.05 11.06 11.07] .1 [11.10 11.11 12.00 12.01])
            se(-1 .9 [10.10 10.11 11.00 11.01] .1 [11.04 11.05 11.06 11.07])
            se(-1 .9 [10.04 10.05 10.06 10.07] .1 [10.10 10.11 11.00 11.01])
            se(-1 .9 [9.10 9.11 10.00 10.01] .1 [10.04 10.05 10.06 10.07])
            se(-1 .9 [9.04 9.05 9.06 9.07] .1 [9.10 9.11 10.00 10.01])
            se(-1 .9 [8.10 8.11 9.00 9.01] .1 [9.04 9.05 9.06 9.07])
;frq4
    p8      se(-1 .9 [11.06 11.07 11.08 11.09] .1 [12.00 12.01 12.02 12.03])
            se(-1 .9 [11.00 11.01 11.02 11.03] .1 [11.06 11.07 11.08 11.09])
            se(-1 .9 [10.06 10.07 10.08 10.09] .1 [11.00 11.01 11.02 11.03])
            se(-1 .9 [10.00 10.01 10.02 10.03] .1 [10.06 10.07 10.08 10.09])
            se(-1 .9 [9.06 9.07 9.08 9.09] .1 [10.00 10.01 10.02 10.03])
            se(-1 .9 [9.00 9.01 9.02 9.03] .1 [9.06 9.07 9.08 9.09])
;frq5
    p9      se(-1 .9 [11.08 11.09 11.10 11.11] .1 [12.02 12.03 12.04 12.05])
            se(-1 .9 [11.02 11.03 11.04 11.05] .1 [11.08 11.09 11.10 11.11])
            se(-1 .9 [10.08 10.09 10.10 10.11] .1 [11.02 11.03 11.04 11.05])
            se(-1 .9 [10.02 10.03 10.04 10.05] .1 [10.08 10.09 10.10 10.11])
            se(-1 .9 [9.08 9.09 9.10 9.11] .1 [10.02 10.03 10.04 10.05])
            se(-1 .9 [9.02 9.03 9.04 9.05] .1 [9.08 9.09 9.10 9.11])
;frq6
    p10     se(-1 .9 [11.10 11.11 12.00 12.01] .1 [12.04 12.05 12.06 12.07])
            se(-1 .9 [11.04 11.05 11.06 11.07] .1 [11.10 11.11 12.00 12.01])
            se(-1 .9 [10.10 10.11 11.00 11.01] .1 [11.04 11.05 11.06 11.07])
            se(-1 .9 [10.04 10.05 10.06 10.07] .1 [10.10 10.11 11.00 11.01])
            se(-1 .9 [9.10 9.11 9.00 9.01] .1 [10.04 10.05 10.06 10.07])
            se(-1 .9 [9.04 9.05 9.06 9.07] .1 [9.10 9.11 9.00 9.01])
}

```

The start times and durations of each sonority are predetermined, as is the general downward trajectory. However, for each sonority, each filter chooses between weighted sets of a particular range of pitches. These were chosen to cover an octave, with overlaps so there could be a variety of intervals. The downward choices are weighted at 90%, but the filter could move upward, giving more variety to the sonorities. Below is an example score file:

```

i1 0 25
;I-block #1 (i2):
i2 0.000 0.800      2.000      11.020      11.020      11.100      11.080      11.100      11.110

```

i2	0.750	1.600	2.000	10.070	10.110	11.010	11.000	11.040	11.040
i2	1.500	1.600	2.000	10.020	10.020	10.060	10.080	10.080	11.010
i2	3.000	2.400	2.000	9.070	9.090	10.010	10.060	10.030	10.050
i2	5.300	3.900	2.000	9.000	9.050	9.040	10.020	9.110	9.110
i2	9.100	6.200	2.000	8.060	8.100	8.110	9.020	9.030	9.060
i2	15.300	10.200	2.000	8.060	8.100	8.110	9.020	9.030	9.060
i2	21.500	10.200	2.000	8.060	8.100	8.110	9.020	9.030	9.060

e

4 – Conclusion

The Sonic Generator explores several techniques which I have been employing in my music lately, most notably algorithmic editing and subtractive and granular synthesis. As a generational tool, the Sonic Generator allows for either highly randomized or highly deterministic renderings of material. This work is a documentation of one of my first forays into programming, but I hope that it can also serve as an introduction into batch programming, Csound, and nGen for other interested composers.