FRACTUS I FOR TRUMPET IN C AND ELECTRONIC SOUND:
A CRITICAL EXAMINATION OF THE COMPOSITIONAL PROCESS

Eli Mulvey Fieldsteel, B.A.

Thesis Prepared for the Degree of

MASTER OF MUSIC

UNIVERSITY OF NORTH TEXAS

May 2010

APPROVED:

Jon C. Nelson, Major Professor
Eugene M. Corporon, Minor Professor
David S. Bithell, Committee Member
Joseph Klein, Chair, Division of Composition Studies
Graham H. Phipps, Director of Graduate Studies
James C. Scott, Dean of the College of Music
Michael Monticino, Dean of the Robert B. Toulouse School of Graduate Studies

*Fractus I* is a composition for trumpet in C and live electronic sound. The electronics were primarily created using SuperCollider, an environment and programming language for real time audio synthesis. This project investigates SuperCollider’s *pattern* and *task* functionality as a means of supporting and enriching the compositional process. *Fractus I* develops several different code architectures in order to randomize as well as synchronize various musical elements. The piece exploits SuperCollider as both an audio synthesis tool and a performance conduit. Additionally, the nature of SuperCollider’s patterns and tasks influences the form and content of the composition. The project underscores SuperCollider as a powerful, versatile and open-ended tool for musical composition and examines future directions and improvements.
Copyright 2010

by

Eli Mulvey Fieldsteel
ACKNOWLEDGMENTS

I would like to extend sincere gratitude to Dr. Jon C. Nelson, Dr. Cindy McTee, and Dr. David Bithell for their indispensable feedback on the composition and formal essay, Grant Harbison and Alan Lara for lending their talent to the project, Nathan Fieldsteel for introducing me to SuperCollider, L. Scott Price for his patient assistance with SuperCollider debugging, Ilya Rostovtsev and Katherine Costa for proofreading, and Ryan Taycher for facilitating percussion recording sessions.
TABLE OF CONTENTS

ACKNOWLEDGMENTS .................................................................................................................. iii

PART I: CRITICAL ANALYSIS

1. Introduction .......................................................................................................................... 2

2. Pre-Compositional Activity ............................................................................................... 3
   Influences
   Recordings

3. SuperCollider ..................................................................................................................... 11
   Introduction
   Playback Engine
   Patterns and Pseudo-Granular Synthesis
   Tasks
     Isolated Tasks
     Regulatory Tasks

4. The Composition ................................................................................................................. 32
   Section I
   Section II
   Section III
   Section IV
   Section V

5. Conclusions and Future Directions .................................................................................. 43

Appendix A: Preliminary Recorded Gestures ......................................................................... 46

Appendix B: Final Recorded Gestures ................................................................................... 47

Reference List .......................................................................................................................... 50

PART II: THE COMPOSITION

SuperCollider Code .................................................................................................................. 52

Performance Score ................................................................................................................ 75
PART I

CRITICAL ANALYSIS
Chapter 1

Introduction

Having composed extensively for wind ensemble, I became interested in exploring the musical potential of a single wind instrument. The trumpet was particularly inviting for both timbral variety resulting from different mutes as well as melodic capacity. Moreover, having previously composed one fixed media composition and two fixed media studies, I was interested in expanding my work with electronic media. Specifically, I was intrigued by the concept of using electronic sound to augment the timbral and gestural capabilities of the instrument.

These ideas prompted *Fractus I*, a thirteen-minute composition for trumpet in C and electronic sound. The electronic component was primarily created using SuperCollider, an environment and programming language for audio synthesis. I developed several compositional techniques using *Tasks* and *Patterns* within SuperCollider, which allowed for open-ended manipulation of tonal and rhythmic elements. As a result, pitch activity and pulse regularity are major focal points in *Fractus I*. This paper details the electronic processes, scrutinizes the compositional process, and speculates on future artistic directions.
Chapter 2
Pre-Compositional Activity

Influences

There exists a growing body of works for solo instrument and electronic sound, in which the electronics attempt to augment the sonic capabilities of the instrument. I studied a variety of these scores and recordings in preparation for composing the piece. Primarily, I examined a collection of compositions for solo trumpet, trumpet and tape, and a handful of pieces with closely related instrumentations (such as trumpet duet, solo unspecified brass instrument, etc.). However, a number of works with seemingly unrelated instrumentations also guided my compositional decisions. I found works by Berio, Bryant, Davidovsky, McTee, and Phillips to be most influential. There are a number of composers whose works were less directly influential but who nevertheless provided interesting perspectives, including Harvey, Kagel, Melby, Morrill, Persichetti, Stockhausen, and Takemitsu.

By far, the piece that most heavily influenced my compositional decisions was Luciano Berio’s Sequenza X for trumpet and piano. The piano part is distinctive in that the pianist never directly produces any notes; rather, the pianist depresses certain keys while the trumpeter plays into the piano, causing the strings to vibrate sympathetically. Although the technique is simple, the effect is extremely convincing; the piano acts as a reverberator, expanding both the performance space and the expressive capabilities of the trumpet.
The manner in which Berio handles pitch and gesture in *Sequenza X* was also of great interest to me. Throughout the first forty-five seconds of the piece, Berio places great emphasis the minor third between pitch classes D and F. The music gradually departs from this subject, but periodically returns to this pair of notes while emphasizing the minor third throughout the piece. Moreover, Berio calls for flutter-tonguing, doodle-tonguing, and valve tremolo in order to create timbral variety within gestures. *Fractus I* draws heavily from these techniques.

Berio’s *Sequenza VII* for oboe is similarly influential. One characteristic feature of this piece is a B-natural, which not only regularly appears in the oboe part, but also continuously sounds offstage. Due to this persistent stasis in pitch content, much of the piece’s development arises from timbral variation. Berio drives the piece forward through the use of alternate fingerings, ornamentation, frequently changing dynamics, and a host of extended techniques. This technique is very effective in prolonging various sections, particularly the opening. By primarily relying on mutes, dynamics, articulation, and ornamentation, *Fractus I* similarly explores timbre as a means of maintaining and evolving thematic material.

Mario Davidovsky’s *Synchronisms No. 6* for piano and electronic sound also made a notable impression on *Fractus I*. The piece begins, innocently enough, with a single note being articulated and sustained on the piano. However, after a few seconds, the electronic sound becomes audible, and the tone seems to become louder without any rearticulation. Once again, although
the technique is simple, the effect is surprisingly impressive. This concept, in conjunction with the central tone idea of *Sequenza VII*, helped shape the opening section of *Fractus I*, which is examined in sections below.

The three pieces mentioned above each contain rhythmic elements that are complex enough to conceal any indication of a regular rhythmic pulse. As a result, while the larger thematic sections retain a sense of direction, the individual gestures tend to sound somewhat independent. I wanted to adopt this technique in order to mask any semblance of regularity and use the trumpet to convey the initial abstract melodic statements. This idea raised questions regarding the best notational practices to specify rhythmic structure. This work is primarily notated using traditional meter, while utilizing spatial notation when purposeful irregularity is desired. In such cases, I employed a technique found in Cindy McTee’s *Einstein’s Dream*. Instead of notating meter with a number representing the beat value (such as 3/4, with the 4 representing a quarter note), McTee indicates 1/X, 2/X, etc., in which the necessary durations are indicated in seconds above the measure. This choice retains a high degree of precision, even when the melodic lines are fairly amorphous. Furthermore, this approach facilitates natural transitions between regular and irregular sections through a simple meter change.

As the larger sections of the composition began to take shape, I became interested in exploiting my decision to notate the music traditionally. More specifically, I wanted to construct a large-scale transition from an arrhythmic
texture to a rhythmic texture. In detailing this idea more closely, I took into consideration two considerably different pieces. The first of these pieces was Mark Phillips' *T-Rex* for trombone and electronics. The fourth movement of Phillips' work opens with a haphazard assortment of pseudo-pitched trombone sounds, which progressively lock into place with one another and become a driving rhythmic pattern. Similarly, Steven Bryant's *Ecstatic Waters* embodies a short, percussive sound and sequences it into a relentless texture of sixteenth notes, which persists throughout most of the third movement. With the support of this rhythmic pulse, the music builds intensity and culminates in a swath of sound.

Bryant's piece also provides a model for a large-scale textural crescendo, in which essentially all musical parameters become more and more chaotic. In composing his piece, Bryant finds inspiration in *The Second Coming* by William Butler Yeats, written in the aftermath of World War I. The third movement draws influence from the line in Yeats' poem, "the center cannot hold,"¹ in which the music "ultimately dominates, subsum[es] everything, spiral[s] out of control, and explod[es]."² *Fractus I* includes a number of large-scale gestures that attempt to emulate this concept.

In addition to studying and borrowing ideas from the established body of repertoire, *Fractus I* is influenced by my most recent composition for fixed media, *Displacement*. This piece was composed using Spear, which is a program for

---

¹ Yeats.
² Bryant.
spectral analysis and resynthesis. Through restricting the source materials and processing techniques, *Displacement* presents a unified sound world. *Fractus I* also preserves this sense of material economy. With only several minor exceptions, the exclusive use of trumpet source material establishes a homogenous sonic environment without sacrificing timbral variety.

*Recordings*

Trumpet samples were collected via three recording sessions with Grant Harbison and Alan Lara. During the second session Alan recorded the relatively short melodic phrases in Appendix A. Although these recordings were generally not used in the final version of the composition, they helped generate more concrete melodic ideas for the instrumental part. In particular, the primary set of pitches in the opening of *Fractus I* can be seen in Figure 1.

Figure 1. Recorded melodic unit and basis for pitch material in *Fractus I*.

During the final recording session, Alan recorded the collection of extremely simple gestures in Appendix B. These simple units of sound provided the basic building blocks from which the audio materials in *Fractus I* were
created. In particular, the following examples proved to be the most useful in synthesizing a variety of larger ideas.

Figure 2. Primary recorded samples used for synthesizing larger gestures.

The following three excerpts also feature prominently in creating faster, more realistic sounding double-tonguing, creating a “cloud” of arrhythmic sounds, and generating the driving rhythmic force inspired by Bryant and Phillips’ music.

Figure 3. Secondary samples used for synthesizing larger gestures.
Other samples used in *Fractus I* include a collection of percussion samples recorded during April 2009. All of the percussive samples are metallic (cymbals, pitched gongs, vibraphone, and crotales). These instruments were selected in order to obtain long, sustained, and timbrally rich sounds. Bowed suspended cymbals, placed upside-down on a timpani head while manipulating the pedal are the most prominent of the samples in the composition. This gesture was useful in synthesizing more complex sounds during later stages in the compositional process. In particular, I incorporated these sounds as convolution impulse responses that “color” the recorded trumpet, thereby maintaining focus on the trumpet as the primary vessel for acoustic and electronic material. These sounds appear in the performance score as “metallic swells” one measure after Cue 08 and at Cue 17.

Figure 4. Graphical representation of trumpet and cymbal convolution.

This work also includes samples of a desk with a creaky folding table and fingers dragging across stretched plastic wrap. Using a few dozen short,
percussive sounds from these sessions, I created percussive textures that could be prolonged for significant periods of time without becoming stagnant. Such textures appear at Cue 11 and Cue 16.

Figure 5. Graphical representations of creak textures and plastic wrap textures.

Time-stretching the recorded desk created deep percussive thuds that emphasize climactic moments. These gestures appear periodically throughout the performance score, including mm. 12, 56, 105, and 175.
Chapter 3
SuperCollider

Introduction

SuperCollider played a pivotal role in the composition of Fractus I. Many of the gestural, timbral, and textural ideas result from the technical capabilities of the software. Specifically, I developed a number of versatile structures with the use of SuperCollider’s Task and Pattern functionality. Moreover, these approaches have a direct and discernable influence on the formal structure of the piece. To supplement the capabilities of SuperCollider, I relied on a combination of Logic Pro for reverb, mastering, and temporal positioning and SoundHack for convolution.

Playback Engine

Using only a few lines of code, SuperCollider is capable of loading a sound file into an internal buffer and playing it back:

```~myBuffer=Buffer.read(s,“/Users/elifieldsteel/Desktop/MySound.aif”);
{PlayBuf.ar(2,~myBuffer,BufRateScale.kr(~myBuffer),doneAction:2)}.play;
```

Figure 6. Basic syntax of SuperCollider buffer playback.

In the example above, SuperCollider searches for the sound file entitled MySound.aif via the specified path name and loads it onto the server (in this case the server has been given the variable name s). SuperCollider then loads this sound file into a buffer, to which it assigns the variable name ~myBuffer. It
should be noted that variable names preceded by a tilde are referred to as environment variables, which are essentially global variables. These variables can be called from anywhere within SuperCollider, as opposed to local variables, which are specific to the function in which they are defined. For convenience purposes, nearly all the variables in *Fractus I* are environment variables. The second line of code instructs SuperCollider to read through the buffer and output the data as sound. 2 specifies the number of channels, ~myBuffer indicates the buffer to be read, BufRateScale.kr(~myBuffer) ensures that the buffer will be read at the original sample rate, and doneAction:2 terminates the process after playback is complete in order to conserve CPU cycles.

From this basic code excerpt, it is fairly straightforward to see that sound files can be easily manipulated within SuperCollider. In particular, it is possible to alter the playback rate of a sound file by multiplying BufRateScale.kr(~myBuffer) by a number. For instance, BufRateScale.kr(~myBuffer)*2 would transpose the buffer up by one octave. The warm, brass-like chords at Cue 17 exemplify this technique. For each desired note in a given chord, there is a PlayBuf whose rate argument is “tuned” to the correct number. This approach is also the source of the low drones in the bass clef from mm. 12-17 and mm. 105-198.
Because the electronic score of *Fractus I* is rooted in recorded sound instead of synthesized sound, the piece makes extensive use of SuperCollider’s buffer playback capabilities. In addition to being the basis for sound design, buffer playback also served as a performance conduit. Employing a line-by-line approach, the finalized electronic sounds are broken into smaller pieces and loaded into a series of `PlayBuf` commands (See Executable Audio Files in the SuperCollider Code section).

This piece requires a human score follower, who activates each line of code according to the indications in the score. Although human error may be a factor, this implementation addresses a number of performance-related concerns. Specifically, with a human score follower controlling the pace of the sample playback, the trumpet player is no longer under such a dire obligation to count seconds with the utmost accuracy. If the trumpeter plays a section too quickly, the score follower can execute the next cue slightly earlier, and vice-
versa. Although the trumpet player and score follower must work together, this approach gives both performers a great deal of flexibility and gives the piece a degree of interactivity.

*Patterns and Pseudo-Granular Synthesis*

Many of the electronic sounds in *Fractus I* were creating using *Patterns*. According to the SuperCollider help documentation,

Patterns describe calculations without explicitly stating every step. They are a higher-level representation of a computational task. While patterns are not ideally suited for every type of calculation, when they are appropriate they free the user from worrying about every detail of the process. Using patterns, one writes what is supposed to happen, rather than how to accomplish it.

In SuperCollider, patterns are best for tasks that need to produce sequences, or streams, of information. Often these are numbers, but they don't have to be -- patterns can generate any kind of object.³

Using patterns, I was able to create effects and textures that would have been logistically impossible while using `PlayBufs` alone. For example, patterns facilitated the sequencing of thousands of instances of a single buffer while manipulating the playback arguments of each individual instance.

Patterns are inherently linked to SuperCollider streams. A stream simply represents a sequence of values, such as [1, 2, 3]. A pattern, on the other hand, is capable of combining streams and specifying stream behavior. According to the SuperCollider help documentation on patterns and streams:

³ McCartney.
The difference between a pattern and a stream is similar to the difference between a score and a performance of that score... \(^4\)

In other words, it makes sense to think of a stream as a series of notes, while a pattern dictates the way those notes are played; the pattern determines how loud each note is played, how long each note is held, how long to wait after a certain note terminates, in what order the notes should be played, and so forth.

SuperCollider provides an extensive list of pattern commands. For instance, one of the simplest patterns is known as \(\text{Pseq}\). The following code provides an instructive example:

```plaintext
a = \text{Pseq}([1, 2, 3], 2).\text{asStream};
a.next;
```

Figure 8. Simple implementation of pattern functionality using \(\text{Pseq}\).

After the first line of code is executed, SuperCollider defines the variable \(a\) as a pattern. In response to the suffix \(\text{.asStream}\), the pattern creates a stream from the bracketed numbers. The 2 that occurs outside of the brackets indicates that the stream is to be iterated twice. The second line of code simply sends the message \text{next} to the stream. Each time this line is executed, SuperCollider will output the next element in the pattern.

\textit{Fractus I} uses a number of slightly more advanced patterns, particularly \texttt{Pseries}, \texttt{Pgeom}, and \texttt{Prand}. \texttt{Pseries} takes a start value, an addend, and a total number of produced values. For example, the following pattern,

\(4\) McCartney.
a = Pseries(1, 2, 5)

Figure 9. Pattern implementation using Pseries.

if executed in the same manner as the Pseq above, would produce the values [1, 3, 5, 7, 9]. Pgeom is similar to Pseries, but the second argument is a multiplicative increment instead of an additive one. For example,

a = Pgeom(1, 2, 5)

Figure 10. Pattern implementation using Pgeom.

would produce [1, 2, 4, 8, 16]. This particular pattern was useful in controlling frequency-related parameters and amplitude-related parameters as a result of their exponential nature. Lastly, Prand randomly chooses from a stream of values for a given number of iterations:

a = Prand([1, 2, 3], 5)

Figure 11. Pattern implementation using Prand.

This code randomly selects a value from the set [1, 2, 3] for a total of 5 selections.

Patterns are extremely versatile in that they can manipulate much more than numbers; in fact, patterns can manipulate any type of data, including audio files. However, as patterns cannot directly interface with buffers, I first defined a series of SuperCollider-specific instruments, or Synths, using the SynthDef
command. `SynthDef` is an object that instantiates a synth, which can in turn be accessed by patterns, routines, tasks, and other data processing structures. In some cases, `SynthDef` instruments are defined by synthesized sound, as in the following example:

```plaintext
SynthDef("sineTone", {
    arg frequency=440, phase=0, amplitude=0.1, duration=1;
    Out.ar(0,
        SinOsc.ar(frequency, phase, amplitude, 0) * Line.kr(1, 0, duration, doneAction:2)
    );
});
```

Figure 12. Syntax of `SynthDef` defined by a sinusoidal oscillator.

Here, SuperCollider defines a synth named `sineTone`. The synth declares four arguments (frequency, phase, amplitude, and duration). The output of the synth, encapsulated within `Out.ar`, is a sinusoidal oscillator multiplied by a line envelope. Again, `doneAction:2` conserves CPU cycles. Without `doneAction:2`, `Line.kr` will continuously output zeros, even after the envelope has finished.

Alternatively, `SynthDefs` can be defined using recorded sound. The following is a simplified example of a `SynthDef` from the final patch of *Fractus I*:
SynthDef("creakChoose", {
    arg amplitude = 1, playbackRate = 1, whichBuffer = 0;
    Out.ar(0,
        PlayBuf.ar(2,
            whichBuffer,
            playbackRate * BufRateScale.kr(whichBuffer),
            doneAction:2)
    );
}).send(s);

Figure 13. Syntax of SynthDef defined by buffer playback.

In this case, SuperCollider defines a synth entitled creakChoose, in which the SinOsc generator from the previous example is replaced with a PlayBuf. As a result, the output originates from an external audio file that has been loaded into a buffer, rather than code within SuperCollider itself. There is an additional creation argument here, named whichBuffer. This argument allows patterns to use a single synth to call upon multiple buffers, which became useful for particular textural effects.

In order to take full advantage of SuperCollider’s pattern functionality, I relied almost exclusively on the Pbind function. Pbind, according to the help documentation, “combines several value streams into one event stream.”

Practically speaking, Pbind allowed me to access a predefined synth and simultaneously manipulate its arguments. An example of this functionality as applied to creakChoose is as follows:

---

5 McCartney.
Pbind(
    \text{\texttt{\textbackslash instrument, \textbackslash creakChoose,}}
    \text{\texttt{\textbackslash amplitude, Pgeom(0.1, 1.01, 20),}}
    \text{\texttt{\textbackslash playbackRate, Pgeom(1.5, 0.95, 20),}}
    \text{\texttt{\textbackslash whichBuffer, Prand([\text{-\texttt{buffer1, \text{-\texttt{buffer2, \text{-\texttt{buffer3}}}}}], 20),}}
    \text{\texttt{\textbackslash dur, Pseq([Pseq([0.2], 10), Pseries(0.2, 0.1, 10)], 1})}}
).play

Figure 14. Combination of several patterns using \texttt{Pbind}.

As illustrated above, \texttt{Pbind} is capable of merging several patterns into a single event. The items on the left hand side (\texttt{\textbackslash instrument, \textbackslash amplitude, etc.}) are arguments, while the corresponding data and patterns on the right specify values for the arguments. \texttt{\textbackslash instrument} is an argument that is native to the \texttt{Pbind} command and is used to specify a \texttt{SynthDef}. \texttt{\textbackslash amplitude, \textbackslash playbackRate, and \textbackslash whichBuffer}, were defined upon instantiation of the \texttt{SynthDef}, and \texttt{\textbackslash dur}, like \texttt{\textbackslash instrument}, is native to \texttt{Pbind} and specifies the duration between events during a sequence. In this case, \texttt{\textbackslash amplitude} increases by a factor of 1.01 for twenty iterations, \texttt{\textbackslash playbackRate} decreases from 1.5 by a factor of 0.95, and \texttt{Pbind} randomly selects one of three buffers upon each individual event. The \texttt{\textbackslash dur} argument is of particular interest, as it demonstrates that patterns can be embedded within other patterns, making them extremely powerful. Here, the wait time remains at a fifth of a second (0.2) for ten iterations, then increases by an addend of 0.1 for another ten iterations. It is worth noting that \texttt{Pbind} requires a great deal of careful calculation; if an addend or factor is too large, it is possible for the playback rate to slip below zero, or
amplitude to exceed one. These errors can cause clipping, CPU overloads, and other undesired artifacts.

By exploiting patterns in this manner, I was able to develop a pseudo-granular synthesis, which is perpetually present throughout Fractus I. The reason I consider this technique to be "pseudo"-granular is mainly that the process does not involve the segmentation and windowing of a longer sound into minute grains; rather, this approach manipulates the entirety of an already small audio file. As a result, the effects often sound similar to that of traditional granular-synthesis, but the granular window is generally larger.

One of the most prominent instances of this technique involves the manipulation of a short, staccato trumpet note. The amplitude of the sound is quite low, the duration between instances is randomly selected from a set of very small values, and the playback rate is unchanged. The resulting sound has a faint, shimmering quality to it, much like heavy reverberation, although this particular sound does not fade over time. The code for this pattern is as follows:

```plaintext
~d4Shimmer = Pbind(  \instrument, \d4grain,  \dur, Prand([0.01, 0.011, 0.0093, 0.0113, 0.027, 0.00892, 0.00834, 0.0218], 2000),  \amplitude, 0.1,  \playbackRate, Pseq([1], 2000) ).play
```

Figure 15. Syntax of pseudo-granular synthesis applied to a short trumpet tone using Pbind.
This gesture appears at the very beginning of the piece and at the very end of the piece, in the form of a sustained D. Because of its reverberant quality, this gesture is often used to accompany the live trumpet and provide a spatial effect, similar to that in Berio's *Sequenza X*.

This technique is also applied to desk creaks and plastic wrap sounds. Most often, the results are massive, sweeping percussive gestures that provide transitional material or brief interjections. Gestures such as these appear at Cue 06, m. 55 (three measures before Cue 10), Cue 20, Cue 22, and periodically from Cue 24 through the end. The code for the `Pbind` at Cue 22 appears as follows:

```plaintext
~cue22 = Pbind(  
   \instrument, \pwGrain,  
   \dur, Pseq([Pgeom(0.75, 0.888, 25), Pseries(0.0385, 0, 120),  
   Pgeom(0.0385, 1.165, 25)], 1),  
   \amplitude, 0.1,  
   \playbackRate, Pgeom(5, Pseq([Pseq([1], 25), Pseq([1], 120),  
   Pseries(1, Pseries(-0.0024, -0.0023, 25))], 170))  
).play
```

Figure 16. `Pbind` syntax of percussive gesture.

Figure 17. Graphical representation of percussive gesture.

Perhaps one of the more complicated structures, ~cue22 involves a
complex interplay of patterns that alter both playback rate and duration. As shown in the `dur` argument in Fig. 16, the time between individual notes begins at 0.75 seconds and becomes progressively shorter over the next 25 iterations via a `Pgeom` with a factor of 0.888. The duration then remains static for 120 iterations via a `Pseries` with an addend of zero. Finally, the duration is elongated for another 25 iterations through a second `Pgeom` with a factor of 1.165. The `playbackRate` argument undergoes a simultaneous transformation. First, this value remains fixed at five times its normal speed via two `Pseqs`. Subsequently, the playback rate is decreased via a `Pseries` nested within another `Pseries`. These sequences of events are visible in Fig. 17.

**Tasks**

For instances in which patterns provide a clumsy or indirect route, *Fractus* relies on SuperCollider tasks. A task, according to SuperCollider, is simply "a pauseable process." Generally speaking, a task is simply a list of commands and a method of execution, such as `start`, `pause`, `resume`, `stop`, `play`, `loop`, etc. SuperCollider provides the following example of a simple task:

---

6 McCartney.
t = Task({
    50.do({ arg i;
        i.squared.postln;
        0.5.wait
    });
});
t.start;
t.pause;
t.resume;
t.reset;
t.stop;

Figure 18. Simple task implementation and control structures.

In this example, task t declares an argument, i. As i travels from zero to fifty, it is squared and sent to the post window with a half-second wait time in between calculations. The commands below the task definition alter the flow of the task and are self-explanatory. In the process of developing a task architecture appropriate for Fractus I, two fundamental types of tasks emerged, which I have decided to call *Isolated Tasks* and *Regulatory Tasks*.

*Isolated Tasks*

An isolated task consists of two tasks, one of which is encapsulated within the other. The inner task includes two sets of curly braces; the inner set of braces is appended by a loop command, while the outer set of braces is appended by a play command. The outer task, on the other hand, has no command at the end; instead, the outer task is given start and stop commands at appropriate points throughout a performance of the piece. The overall result of this architecture is an infinite loop that can be stopped and started at will. A simplified example from the final patch of Fractus I is as follows:
Figure 19. Basic isolated task architecture.

This task, along with most isolated tasks in *Fractus I*, has only two commands. The first, beginning with `Synth.new`, creates an instance of the synth called `creakChoose` (created earlier with a `SynthDef`), while the second command causes the task to pause for a randomly determined amount of time between half a second and two seconds. Upon creation, the overall task does not start playing immediately as a result of the lack of a `play` or `start` command at the end of the outermost task. Rather, this isolated task can be activated with the command `~creakPlay.start` and can be similarly deactivated with `~creakPlay.stop`.

As demonstrated in this example, isolated tasks can be extremely conducive to producing irregular textures. `~creakChoose` randomly calls on a set of three buffers and reads them while varying panning, amplitude, and wait times. Although a similar effect could have been created using patterns, patterns lend themselves more to sequential gestures. Even considering the availability of patterns like `Prand`, SuperCollider selects from a set of random values when using patterns, rather than a range of random values, as in the `rrand` command above. Overall, isolated tasks are more open-ended and have better capabilities.
of producing nonuniform and unstructured textures. Isolated tasks such as these, which usually involve either desk creaks or plastic wrap, appear at Cue 11, Cue 14, Cue 15, Cue 16, and Cue 20.

In some cases, isolated tasks were useful in emulating pattern-produced sounds, but additionally took advantage of the ability to select from a range of random values in order to expand the sonic possibilities. For instance, the following task replicates the sound created by the \textsc{\textasciitilde d4Shimmer} example in the previous section:

\begin{verbatim}
~d4Smear = Task({
x = 1.0;
y = 1.0001;
~d4SmearInner = Task({
  {Synth.new("d4Grain", [
    \amplitude, 0.01,
    \playbackRate, rrand(x, y), ],
    s, \addBefore );

  rrand(0.008, 0.01).wait;
  x = max(0.46, 2 - (x * 1.0005));
  y = min(2.1, y * 1.0005);
}.loop
}.play);
});
\end{verbatim}

Figure 20. Modified isolated task architecture resulting in gradual pitch expansion.

This task uses the same \textsc{d4Grain} synth and selects from a range of very small wait times, therefore creating a very similar effect as the \textsc{\textasciitilde d4Shimmer} pattern. However, the primary difference between the two lies in the creation and manipulation of the \textit{x} and \textit{y} arguments shown above. Playback rate was fixed in
the ~d4Shimmer example, but here, these variables dictate the minimum and maximum values of a randomly selected playback rate. Upon each loop of this task, \( x \) is slightly decreased and \( y \) is slightly increased, thereby expanding the playback range. The \texttt{max} and \texttt{min} functions prevent the variables from becoming too small or too large. The overall result is a gradual tonal smear, in which a distinct pitch evolves into a more chaotic cloud of tones. This gesture occurs at Cue 02, Cue 09, m. 158 (two measures after Cue 32), and throughout the coda beginning at m. 175.

![Figure 21. Graphical representation of the tonal smear produced using isolated tasks.](image)

There are a few isolated tasks that expand on the architecture presented above in order to communicate with other isolated tasks. More specifically, some of the tasks in \textit{Fractus I} are capable of activating other tasks and deactivating themselves.

The pair of isolated tasks, \texttt{~shortTptPlayDense} and \texttt{~shortTptPlaySparse}, provide an instructive example. As these tasks involve a relatively large amount of code, they can be found in the SuperCollider Code section. These two tasks primarily provide alterations between dense and sparse textures of randomized short trumpet samples. This section begins at Cue 20.
However, in order to facilitate the score follower’s job, the piece includes several environment variables and counters (such as ~counterShortTptDense, ~counterShortTptSparse, ~trumpetCounter, ~shortTptDenseNumberOfGestures, ~shortTptSparseNumberOfGestures) that allow these two tasks to interact with one another. Furthermore, these environment variables allow these two tasks to communicate with the aforementioned desk creak tasks, activating and deactivating them according to a given set of rules.

Before every instance of a short trumpet sample, the task in question reviews a series of conditional statements. The task checks the values of these environment variables and acts accordingly. For example, before each individual trumpet note, ~shortTptPlayDense checks whether ~trumpetCounter is greater than or equal to three. If so, it deactivates an isolated task of dense creaks, preventing another instance of “denser” creaks at m. 99. Next, ~shortTptPlayDense checks ~trumpetCounter against a value of six. If they are equal, both trumpet tasks are terminated and the trumpet counter is reset. Note that there are a total of six measures of trumpet tasks from mm. 95-100, and that the electronic trumpet part stops at the end of the sixth measure. ~trumpetCounter corresponds to the current measure, and is increased by one each time a measure consisting of these trumpet tasks is completed. Lastly, ~shortTptPlayDense checks the current number of sounded trumpet notes against a predetermined total. If the number of notes has exceeded this total, the
task sets parameters for its sibling task, activates the sibling task, and terminates itself. ~shortTptPlaySparse is nearly identical to ~shortTptPlayDense in that it undergoes a similar process of determining values for its sibling task, ~shortTptPlayDense. Through this interaction, these dense and sparse trumpet textures alternate autonomously.

There is a second pair of isolated tasks in *Fractus I*, ~popPlayRegular and ~popPlaySolid, which expands upon the basic architecture of isolated tasks in a different manner. These two tasks manipulate a collection of mouthpiece pops. What makes these two tasks unique is the use of an all-pass filter in their corresponding SynthDefs in order to create delay lines:

```plaintext
SynthDef("handPopSynthRegular", { 
    arg amplitude = 1, playbackRate = 1, pan = 0, whichPop = 0, 
    maxDelTime = 0.065, delTime = 0.065, decTime = 5, bandpassFreq = 0.75;
    Out.ar(0, 
        BPF.ar(
            Pan2.ar(
                AllpassC.ar(
                    PlayBuf.ar(1, 
                        whichPop, 
                        playbackRate * BufRateScale.kr(whichPop), 
                        doneAction:2), 
                        maxDelTime, 
                        delTime, 
                        decTime), 
                        pan, 
                        amplitude), 
                    SinOsc.kr(bandpassFreq, 0, 1200, 1300), 0.5 
                )
            )
        )
    ).send(s);

    Figure 22. Syntax of SynthDef incorporating an all-pass filter.
```
The resulting sounds are significantly more repetitive than the isolated tasks mentioned above.

The construction of ~\texttt{popPlayRegular} begins by assigning numerical values to six variables, which serve as six randomly selected delay times:

\begin{verbatim}
b = 0.093;
c = 0.1422;
d = 0.1916;
e = 0.2701;
f = 0.313;
g = 0.3917;
\end{verbatim}

Figure 23. Declaration of variables representing delay times.

As the task progresses, these variables are gradually increased or decreased,

\begin{verbatim}
b = b * rrand(0.975, 0.99);
c = c * rrand(0.986, 0.999);
d = d * rrand(1.01, 1.013);
e = e * rrand(0.993, 0.995);
f = f * rrand(1.016, 1.021);
g = g * rrand(1.009, 1.014);
\end{verbatim}

Figure 24. Mathematical manipulation of delay time variables.

and eventually converge on even divisions of the second. In order to accomplish this convergence, the delay time argument relies on a collection of \texttt{max} and \texttt{min} operators, which prevent the delay time variables from exceeding a given division:

\begin{verbatim}
\texttt{\\delTime\, [max(0.0625, b), max(0.125, c), min(0.25, d),}
\texttt{ max(0.25, e), min(0.5, f), min(0.5, g)]}
\end{verbatim}

Figure 25. \texttt{max} and \texttt{min} functions holding values to even divisions of the second.
In *Fractus I*, `~popPlaySolid` is activated as soon as `~popPlayRegular` converges on a fixed pulse at m. 105. Here, the delay times are fixed at even divisions of the second and remain so for the rest of the piece. Among the parameters being manipulated are wait time, amplitude, and second division; hand pops sometimes occur as repeating quarter notes, eighth notes, sixteenth notes, or thirty-second notes. However, despite these parameters being constantly altered, the lack of a randomized wait time gives these tasks a distinct pulse and places them in stark contrast to the aforementioned isolated tasks. The presence and absence of this pulse is a central thematic idea in *Fractus I*.

![Figure 26. Graphical representation of eighth- and sixteenth-note hand pop pattern.](image)

*Regulatory Tasks*

This persistent pulse prompted the development of regulatory tasks. In previous sections of the piece, the score follower is not required to execute cues with machine-like precision. However, with a prominent pulse in the music, it is impractical to require the human score follower to synchronize sample playback.

Regulatory tasks are fundamentally different from isolated tasks in that there is no `loop` command. Instead, a regulatory task consists of a series of
instructions that are executed once, according to the order in which they appear. These tasks each include some combination of PlayBufs, wait times, numerical reassignments to environment variables, and start/stop instructions to isolated tasks. The following is an example of a regulatory task from the final patch of Fractus I:

```super
~playGesture28 = Task({
  {PlayBuf.ar(2, ~g24, BufRateScale.kr(~g24), doneAction:2)}.play;
  ~popSolidAmp = 3.0;
  4.wait;
  ~dblConvPlayQuiet.stop;
  9.wait;
  ~popSolidFadeMultiplier = 0.62;
  ~fadePopPlaySolid.start;
  3.5.wait;
  ~cymbalBowPlay.stop;
});
```

Figure 27. Basic syntax of regulatory tasks.

When given the command ~playGesture28.start, SuperCollider executes a PlayBuf, then immediately sets ~popSolidAmp to a value of three, then waits for four seconds, and so forth. Requiring only a few minor calculations, these regulatory tasks provided a means of seamlessly connecting several pulse-oriented sections while requiring very little work from the score follower.
Chapter 4
The Composition

Buffer playback, patterns, and tasks served as the basic tools with which I composed *Fractus I*. More importantly, however, these techniques had a discernable impact on form and gesture. Further still, numerous sections of the piece are heavily influenced from aforementioned pieces in the contemporary repertoire. An examination of the formal structure of *Fractus I* highlights these influences.

*Fractus I* is composed of five distinct sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Type</th>
<th>Minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section I:</td>
<td>Initial Statement</td>
<td>mm. 1 – 60</td>
</tr>
<tr>
<td>Section II:</td>
<td>Lyric</td>
<td>mm. 61 – 82</td>
</tr>
<tr>
<td>Transition:</td>
<td></td>
<td>mm. 83 – 94</td>
</tr>
<tr>
<td>Section III:</td>
<td>Aleatoric</td>
<td>mm. 95 – 102</td>
</tr>
<tr>
<td>Transition:</td>
<td></td>
<td>mm. 103 – 104</td>
</tr>
<tr>
<td>Section IV:</td>
<td>Pulse-driven</td>
<td>mm. 105 – 174</td>
</tr>
<tr>
<td>Section V:</td>
<td>Coda</td>
<td>mm. 175 – 198</td>
</tr>
</tbody>
</table>

Large-scale pitch and rhythmic manipulation are central ideas in *Fractus I*. Over the course of the entire work, there is a gradual pitch expansion and contraction. This technique is a reflection of SuperCollider’s ability to manipulate a single pitch into a cloud of tones. Additionally, the piece utilizes a variety of rhythmic structures, ranging from completely aleatoric to pulse-driven. The variety of
rhythmic ideas in *Fractus I* is similarly influenced by the interplay of rhythmically unstructured isolated tasks and synchronized regulatory tasks.

*Section I*

The focal point of the first section is the initial melodic statement, which is conveyed through the live trumpet. The electronic component largely operates in the background, where it provides support for the trumpet and occasionally interrupts with percussive or textural gestures. In order to achieve a convincing and logical structure, this section relies heavily on organic development and timbral variety.

One of the most salient characteristics of the opening section is the prevalence of the pitch class D. During the first forty-five seconds, both the electronic and acoustic components strictly adhere to this pitch. Although there is a microtonal degree of pitch variation, these departures are present in order to create timbral diversity. The electronic component, while appearing fairly static in the score, is composed of a variety of processed and non-processed sounds. Included among these sounds is the staccato trumpet *Pattern* mentioned above, and long, reverberant trumpet samples. The combination of granular textures and heavy reverberation creates an irregular and timbrally complex tone. Simultaneously, the live trumpet explores this pitch with a number of different dynamic, articulation, ornamentation, and mute indications.
The idea of establishing a section by prolonging a single tone is primarily taken from the prominent B in Berio’s Sequenza VII, as well as the opening gesture from Davidovsky’s Synchronisms No. 6. Moreover, the timbral variety in the live trumpet is a rough acoustic adaptation of the technical nature of SuperCollider Patterns.

At m. 12, the electronic part presents the first percussive interruption, at which point the live trumpet begins to explore new pitches while maintaining a sense of timbral variation. The trumpet moves to F in m. 13 and C-sharp in m. 18. At this point, the piece establishes this combination of pitches D, F, and C-sharp as a thematic unit, similar to the introductory pitch materials in Sequenza X. The electronic component rarely deviates from these three pitches throughout the remainder of Section I.

The music approaches a local climax at Cue 06, at which point the electronic component interjects the first of several creak patterns and recedes into the background once more. From here, the trumpet continues to wander...
through new pitches while gravitating back toward D, F, and C-sharp. There is a brief repose from mm. 37-40, in which the electronic part disappears altogether and the trumpet becomes somewhat more lyrical. This segment attempts to maintain the tension of Section I while foreshadowing Section II.

The final portion of Section I, from Cue 09 to Cue 10, is an extended crescendo of pitch, dynamics and rhythmic activity. This is, in part, a manifestation of the musical ideas in the third movement of Bryant’s *Ecstatic Waters*. The electronic part begins by stating each of the three predominant pitches according to the order in which they originally appeared. However, in this final segment, these pitches are patterned in SuperCollider and gradually expand outward. This technique is a prolongation of the small tonal expansion at Cue 02. Furthermore, the electronic part gradually adds new elements, including the sound of fingernails on the trumpet bell at (indicated as “percussive clinks”) in m. 44, a time-stretched trumpet drone in m. 49, and a flutter-tongued tone in m. 55. The live trumpet ascends to a flutter-tongued A-flat, which is the highest pitch thus far and timbrally distant from the original D. A long creak pattern begins in m. 55, which extends into Section II and transforms into a low rumbling sound. As with Bryant’s piece, “the center cannot hold” and the music dissipates in an aggressive exhalation of sound.
The entirety of Section I is, in essence, a large-scale realization of the gradual “tonal smear” idea developed with patterns; the piece begins with a single pitch, and both the tonal palette and rhythmic activity increase over the course of the section.

Section II

Section II, a formal realization of the brief repose in Section I, is considerably more lyrical and relaxed in comparison to the previous section. Here, the live trumpet part is less active; there are more sustained trumpet tones and fewer rhythmic disturbances. Although the electronic part remains somewhat textural, it begins to emerge from the background by providing contrapuntal lines at Cue 13 and chordal accompaniment at Cue 14 and Cue 17. As a result, there
is a more balanced relationship between the acoustic and electronic components.

Figure 30. Excerpt of melodic, chordal and contrapuntal material in Section II.

Each electronic pitch in Section II is the result of one of two processes: convolution with sampled rain, or sequencing within a SuperCollider pattern (similar to some electronic gestures in Section I). In either case, the processed tone has an ethereal quality to it, which contributes to timbral variety, as well as the subdued character of the section.

In contrast to Section I, Section II purposefully avoids excessive tonicization of D. This idea is indicative of overall pitch expansion and contraction; having completed a gradual tonal expansion in Section I, the music now exercises a greater degree of freedom in pitch content. However, the piece continues to place great emphasis on the three intervals established by the
central pitches in Section I; the trumpet melody frequently outlines the minor third, tritone, and major seventh.

Section II also marks the beginnings of a departure from fixed electronic gestures. There are two distinct percussive textures in this section, which are composed of small sounds derived from desk creaks and plastic wrap, occurring at Cue 11 and Cue 16, respectively (indicated in Fig. 5). These textures are controlled by three isolated tasks: ~creakPlaySparse, ~creakPlayAverage, and ~pwPlay. SuperCollider randomly determines the finer details of these textures, but the overall effect remains nearly identical on each execution.

At m. 83, the trumpet and electronic part mutually initiate a transition into Section III. Both components begin to introduce short, accented trumpet notes into the chordal texture. These notes are placed at irregular intervals in order to cohere with the percussive background texture, as well as foreshadow the rhythmic disorder in Section III. These interruptions become more frequent, and reach a climax at Cue 20.

Section III

Structurally, Section III is highly developmental and radically different from the rest of the composition. With only a few exceptions, every sound in this section is determined with some degree of randomness. The percussive textures from Section II become more prevalent, while both the electronic component and the performer freely explore the staccato gestures from the previous transition.
As the section proceeds, various electronic textures emerge from and recede into the background, while the trumpet player repeatedly introduces new gestures and eliminates older ones.

Section III departs from traditional metered notation, using $1/X$ and $3/X$ metric indicators. As a result, both the duration and content of this section can vary significantly from performance to performance.

Section III mirrors Section I, with highly active and volatile gestures evolving into a state of equilibrium. At the beginning of Section III, the electronic score abruptly alternates between dense and sparse textures of staccato trumpet notes, which are paralleled by dense and sparse desk creaks (governed by `~shortTptPlayDense` and `~shortTptPlaySparse`). The dense creaks are abolished at Cue 21, and the electronic trumpet tones disappear entirely at Cue 22. At this point, a long percussive pattern emerges and transforms into a texture of bell-like tones over the course of twenty-three seconds (Fig. 17). With the arrival of these bell tones, the piece reaches equilibrium; the trumpet has transitioned from short, agitated gestures to long, simple tones in a low register, while the electronic component periodically emits clusters of bell tones.
Figure 31. Graphical representation of equilibrium in Section III.

The equilibrium of Section III is disrupted at Cue 23 by a collection of hand pops (governed by $\neg$\texttt{popPlayRegular}), which initiates a transition into the final section. The hand pops gradually converge toward a regular pulse, the trumpet gestures become slightly more agitated, and a massive creak pattern carries the music into Section IV.

\textit{Section IV}

At the onset of Section IV, the trumpet mouthpiece pops create a regularly recurring pulse that is maintained throughout (this pulse is controlled by the isolated task $\neg$\texttt{popPlaySolid}). The presence of this regular pulse gives this section a distinct character, in which the live performer and electronics are synchronized. This pulse-driven section is partly inspired by Bryant’s \textit{Ecstatic Waters} and Phillips’ \textit{T-Rex}. 
Despite the presence of a regular pulse, the structure of Section IV is similar to Section I. At its onset, the trumpet exhibits little rhythmic activity, but the amount of rhythmic activity increases over time. For example, the trumpet introduces both ascending and static sixteenth note gestures at m. 128 and m. 149, respectively, which increase in length and frequency of occurrence.

Figure 32. Ascending and static sixteenth notes gestures.

As in Section I, the electronic part lends support to the trumpet and highlights the trumpet gestures while periodically interrupting with aggressive rushes of sound at Cue 25, Cue 27, and Cue 30. These gestures also become longer, denser, and more aggravated as the section progresses.

The pitch content of Section IV begins with A and A-flat at Cue 24, which refer back to the A-flat at the conclusion of Section I. Over the course of Section IV, the bass line and trumpet part gradually converge on D, much as the hand-pops from section III converge on a steady pulse.
Section V

Section V is a recapitulatory coda that combines several ideas from earlier sections of the piece. Notably, this section is tonally and timbrally similar to the opening forty-five seconds of the piece, although the material here is accompanied by the rhythmic mouthpiece pops. Section V can also be considered a restatement, and perhaps a fuller realization of the long concluding gesture from Section I, in that it consists of an even longer, more dramatic tonal smear (pp. 87-88). The upper register of the electronics include an elongated melody on the notes D – F – D, a restatement of the pervasive minor third, while the final pitch of the live trumpet part is a C-sharp, providing a complete restatement of the original set of three pitches. Despite the magnitude of the final gesture, the piece ends very lightly and perhaps unpredictably. The trumpet player executes the first and only live hand pop, while the electronics respond with a bright, reverberant chime-like sound.
Chapter 5
Conclusions and Future Directions

SuperCollider provides an instructive and versatile environment for electroacoustic composition, which allows for the development of flexible code structures for audio synthesis and performance techniques. These relatively simple code structures were extremely conducive to composing large-scale formal ideas, as well as individual gestural ideas. At the same time, the composition of Fractus I raises a number of noteworthy issues about score following, notation, meter, and indeterminacy.

Although the human score follower implementation addresses several concerns about fixed media and provides the performer with a degree of flexibility, human error remains a considerable factor. If the score follower simultaneously executes two or more cues from a slip of the finger, the electronics cannot be corrected without stopping the electronic sound altogether and restarting from a given cue. Despite the dangers, however, using a human score following method is a direct, simple, and effective approach in the context of this piece.

The combination of traditional meter and spatial meter provides gestural flexibility and works well with SuperCollider’s pattern and task functionality. When working in traditional meter, regulatory tasks are significantly easier to construct and manage, particularly when the conversion factor between time and meter is simple. As such, the tempo is set to 60 or 120 beats per minute in traditionally
metered sections. This approach also facilitates synchronization between gestures and PlayBufs. Spatial notation is convenient in emphasizing amorphous gestural ideas and eliminating pulse. Moreover, isolated tasks were well suited to spatially notated sections. Furthermore, McTee’s metric indications allow for seamless transitions between pulse-oriented gestures and aleatoric gestures.

The pattern and task architecture developed over the course of this project serve as an excellent point of departure for future technical improvements and compositional ideas. I expect to continue this series of Fractus pieces while improving existing code and developing new implementations. Specifically, the ability to code patterns and tasks within other patterns and tasks is a significant point of interest; this idea is conducive to controlling macroscopic formal elements as well as microscopic gestural nuance. Moreover, I expect to expand the sonic potential of future works by incorporating synthesized sound in addition to recorded sound. Using the techniques and ideas uncovered here, I hope to create a series of SuperCollider pieces similar to Berio’s Sequenzas or Davidovsky’s Synchronisms.

The metric and notational concepts in Fractus I have potentially beneficial implications for future acoustic works as well. The union of spatial and traditional metric indications has the potential to greatly facilitate the combination of random and non-random elements in large ensemble acoustic music.
The composition of *Fractus I* has revealed a number of interesting artistic ideas, opportunities and directions. Through a further research and exploration with SuperCollider, I expect to improve on existing practices, develop better compositional tools, and unearth new compositional ideas.
Appendix A

Preliminary Recorded Gestures

1. $J = 60$

2.

3.

4.

5.

6. ($-5$ sec.)

7.
Appendix B

Final Recorded Gestures

1. Very short, aggressive articulation. For recording purposes, allow at least 2 sec. of silence between notes.

2. Still short and somewhat aggressive, but slightly longer than example 1. Again, allow at least 2 sec. of silence between notes.

3. 3 - 4 sec. per note. Leave ≥ 2 sec. of silence between notes.

4. ~ 8 sec. per note. Leave ≥ 2 sec. of silence between notes.

5. Allow at least 2 sec. of silence between pairs of notes in examples 5 - 8.

6. Repeat entire example at mf and pp.

7. Repeat at mf and pp.

8. Repeat at mf and pp.


10. Repeat each measure 4 - 8 times with no pause between repeats. Leave ≥ 2 sec. of silence before moving to a new measure.

Appendix B

Final Recorded Gestures

1. Very short, aggressive articulation. For recording purposes, allow at least 2 sec. of silence between notes.

2. Still short and somewhat aggressive, but slightly longer than example 1. Again, allow at least 2 sec. of silence between notes.

3. 3 - 4 sec. per note. Leave ≥ 2 sec. of silence between notes.

4. ~ 8 sec. per note. Leave ≥ 2 sec. of silence between notes.

5. Allow at least 2 sec. of silence between pairs of notes in examples 5 - 8.

6. Repeat entire example at mf and pp.

7. Repeat at mf and pp.

8. Repeat at mf and pp.


10. Repeat each measure 4 - 8 times with no pause between repeats. Leave ≥ 2 sec. of silence before moving to a new measure.
Repeat each measure 4 - 8 times with no pause between repeats. Leave ≥ 2 sec. of silence before moving to a new measure.

Repeat at \( mf \) and \( pp \).

Allow ≥ 2 sec. of silence between pairs of notes in examples 12 - 13.

Leave ≥ 2 sec. of silence between each set of three notes. Lip the pitch up or down as indicated.

Number of noteheads is approximate and only serves to represent the overall character and shape of the gesture. The first and last notes should each last 1 - 2 sec. The overall gesture should last 15 - 18 sec.

Repeat example 15 with the following pairs of pitches.

For ease of fingering/embouchure, the pitches may be transposed up or down by a small amount, but the interval must not be altered.
16. Flick fingernails on bell

\[ \text{repeat at } mf \text{ and } pp. \]

17. Blow air through horn

\[ \text{as long as is comfortable} \]

\[ \text{quickly, but at a comfortable speed} \]

18. Hand-pops: strike mouthpiece with open palm while fingering indicated pitches

\[ \text{repeat at } mf \text{ and } pp. \]
Reference List


PART II

THE COMPOSITION
SuperCollider Code

/** Server Setup **/
s = Server.local
s.boot
s.freeAll

/** Create/Reset Environment Variables **/

(~trumpetCounter1 = 0;
~counterShortTptDense = 0;
~counterShortTptSparse = 0;
~uselessCounter = 0;
~shortTptDenseNumberOfGestures = 0;
~shortTptSparseNumberOfGestures = 0;
~popCounter = 0;
~popRegularAmp = 0;
~popSolidAmp = 3.2;
~popSolidDecayMin = 10.0;
~popSolidDecayMax = 15.0;
~thirtySecondNoteWeight = 0;
~sixteenthNoteWeight = 0;
~eighthNoteWeight = 0.75;
~quarterNoteWeight = 0.25;
~waitTime1 = 2;
~waitTime2 = 1;
~waitTime3 = 0.5;
~waitTime1Weight = 0.2;
~waitTime2Weight = 0.7;
~waitTime3Weight = 0.1;
~cymbalBowAmp = 0.52;
~popSolidFadeMultiplier = 0.65;
~popSolidPbrLo = 0.43;
~popSolidPbrHi = 1.8;
)

/** Buffers **/

(~g01 = Buffer.read(s, "fractus_i_audio/Output_01_secondDraft.aif");
~g02 = Buffer.read(s, "fractus_i_audio/AngryBees_fast.aif");
~g03 = Buffer.read(s, "fractus_i_audio/Low Tpt Drone 2.aif");
~g04 = Buffer.read(s, "fractus_i_audio/Extremely Low Dual Tpt Drone 3.aif");
~g05 = Buffer.read(s, "fractus_i_audio/Output_02.aif");
~g06 = Buffer.read(s, "fractus_i_audio/Output_03.aif");
~g07 = Buffer.read(s, "fractus_i_audio/Output_04.aif");
~g08 = Buffer.read(s, "fractus_i_audio/Output_05.aif");
~g09 = Buffer.read(s, "fractus_i_audio/Output_06.aif");
~g10 = Buffer.read(s, "fractus_i_audio/Output_07.aif");
~g11 = Buffer.read(s, "fractus_i_audio/Output_08.aif");
~g12 = Buffer.read(s, "fractus_i_audio/Output_09.aif");
~g14 = Buffer.read(s, "fractus_i_audio/Output_11a_revised.aif");
~g15 = Buffer.read(s, "fractus_i_audio/Output_12.aif");
~g16 = Buffer.read(s, "fractus_i_audio/Output_13a.aif");
~g17 = Buffer.read(s, "fractus_i_audio/Raygun_up_accel.aif");
~g18 = Buffer.read(s, "fractus_i_audio/Low_Drone_Ab1_fadeout_reverb.aif");
~g19 = Buffer.read(s, "fractus_i_audio/Fast_Db1_Tongue_A3.aif");
~g20 = Buffer.read(s, "fractus_i_audio/Output_14.aif");
~g21 = Buffer.read(s, "fractus_i_audio/Low_Drone_A1_fadeout.aif");
~g22 = Buffer.read(s, "fractus_i_audio/Output_16.aif");
~g23 = Buffer.read(s, "fractus_i_audio/Output_17.aif");
~g24 = Buffer.read(s, "fractus_i_audio/Output_18.aif");
~g25 = Buffer.read(s, "fractus_i_audio/Output_15.aif");
~g26 = Buffer.read(s, "fractus_i_audio/Output_19.aif");
~g27 = Buffer.read(s, "fractus_i_audio/Output_20.aif");
~g28 = Buffer.read(s, "fractus_i_audio/Output_21.aif");
~g29 = Buffer.read(s, "fractus_i_audio/Output_22.aif");
~g30 = Buffer.read(s, "fractus_i_audio/Output_23.aif");
~g31 = Buffer.read(s, "fractus_i_audio/Output_24.aif");
~g32 = Buffer.read(s, "fractus_i_audio/Output_25.aif");
~creak0 = Buffer.read(s, "fractus_i_audio/Creak0.aif");
~creak1 = Buffer.read(s, "fractus_i_audio/Creak1.aif");
~creak2 = Buffer.read(s, "fractus_i_audio/Creak2.aif");
~creak3 = Buffer.read(s, "fractus_i_audio/Creak3.aif");
~creak4 = Buffer.read(s, "fractus_i_audio/Creak4.aif");
~creak5 = Buffer.read(s, "fractus_i_audio/Creak5.aif");
~creak6 = Buffer.read(s, "fractus_i_audio/Creak6.aif");
~creak7 = Buffer.read(s, "fractus_i_audio/Creak7.aif");
~creak8 = Buffer.read(s, "fractus_i_audio/Creak8.aif");
~creak9 = Buffer.read(s, "fractus_i_audio/Creak9.aif");
~creak10 = Buffer.read(s, "fractus_i_audio/Creak10.aif");
~creak11 = Buffer.read(s, "fractus_i_audio/Creak11.aif");
~creak12 = Buffer.read(s, "fractus_i_audio/Creak12.aif");
~pw0 = Buffer.read(s, "fractus_i_audio/PlasticWrap0.aif");
~pw1 = Buffer.read(s, "fractus_i_audio/PlasticWrap1.aif");
~pw2 = Buffer.read(s, "fractus_i_audio/PlasticWrap2.aif");
~pw3 = Buffer.read(s, "fractus_i_audio/PlasticWrap3.aif");
~pw4 = Buffer.read(s, "fractus_i_audio/PlasticWrap4.aif");
~pw5 = Buffer.read(s, "fractus_i_audio/PlasticWrap5.aif");
~pw6 = Buffer.read(s, "fractus_i_audio/PlasticWrap6.aif");
~pw7 = Buffer.read(s, "fractus_i_audio/PlasticWrap7.aif");
~pw8 = Buffer.read(s, "fractus_i_audio/PlasticWrap8.aif");
~pw9 = Buffer.read(s, "fractus_i_audio/PlasticWrap9.aif");
~drone1 = Buffer.read(s, "fractus_i_audio/CymbalTrumpetDrone1.aif");
~drone2 = Buffer.read(s, "fractus_i_audio/CymbalTrumpetDrone2.aif");
~drone3 = Buffer.read(s, "fractus_i_audio/CymbalTrumpetDrone3.aif");
~drone4 = Buffer.read(s, "fractus_i_audio/CymbalTrumpetDrone4.aif");
~dblConvTpt1 = Buffer.read(s, "fractus_i_audio/Ethereal_Tpt_Eb3_doubleConvolved_short.aif");
~dblConvTpt2 = Buffer.read(s, "fractus_i_audio/Ethereal_Tpt_Eb3_doubleConvolved_2_short.aif");
~creakSeqAlea0 = Buffer.read(s, "fractus_i_audio/acce11.aif");
~creakSeqAlea1 = Buffer.read(s, "fractus_i_audio/acce12.aif");
~creakSeqAlea2 = Buffer.read(s, "fractus_i_audio/acce13.aif");
~creakSeqAlea3 = Buffer.read(s, "fractus_i_audio/acce14.aif");
~creakSeqAlea4 = Buffer.read(s, "fractus_i_audio/acce15.aif");
~creakSeqAlea5 = Buffer.read(s, "fractus_i_audio/dece11.aif");
~creakSeqAlea6 = Buffer.read(s, "fractus_i_audio/dece14.aif");
~creakSeqAlea7 = Buffer.read(s, "fractus_i_audio/dece10.aif");
~creakSeqAlea8 = Buffer.read(s, "fractus_i_audio/dece14.aif");
~creakSeqAlea9 = Buffer.read(s, "fractus_i_audio/dece15.aif");
~shortTpt0 = Buffer.read(s, "fractus_i_audio/00_F3_reverb.aif");
~shortTpt1 = Buffer.read(s, "fractus_i_audio/01_Ab3_reverb.aif");
~shortTpt2 = Buffer.read(s, "fractus_i_audio/02_B3_reverb.aif");
~shortTpt3 = Buffer.read(s, "fractus_i_audio/03_D4_reverb.aif");
~shortTpt4 = Buffer.read(s, "fractus_i_audio/04_F4_reverb.aif");
~shortTpt5 = Buffer.read(s, "fractus_i_audio/05_Ab4_reverb.aif");
~shortTpt6 = Buffer.read(s, "fractus_i_audio/06_Bb4_reverb.aif");
~shortTpt7 = Buffer.read(s, "fractus_i_audio/07_D5_reverb.aif");
~shortTpt8 = Buffer.read(s, "fractus_i_audio/08_F5_reverb.aif");
~shortTpt9 = Buffer.read(s, "fractus_i_audio/09_Ab5_reverb.aif");
~shortTpt10 = Buffer.read(s, "fractus_i_audio/10_B5_reverb.aif");
~shortTpt11 = Buffer.read(s, "fractus_i_audio/11_F3_littlereverb.aif");
~shortTpt12 = Buffer.read(s, "fractus_i_audio/12_Ab3_littlereverb.aif");
~shortTpt13 = Buffer.read(s, "fractus_i_audio/13_B3_littlereverb.aif");
~shortTpt14 = Buffer.read(s, "fractus_i_audio/14_D4_littlereverb.aif");
~shortTpt15 = Buffer.read(s, "fractus_i_audio/15_F4_littlereverb.aif");
~shortTpt16 = Buffer.read(s, "fractus_i_audio/16_Ab4_littlereverb.aif");
~shortTpt17 = Buffer.read(s, "fractus_i_audio/17_Bb4_littlereverb.aif");
~shortTpt18 = Buffer.read(s, "fractus_i_audio/18_D5_littlereverb.aif");
~shortTpt19 = Buffer.read(s, "fractus_i_audio/19_F5_littlereverb.aif");
~shortTpt20 = Buffer.read(s, "fractus_i_audio/20_Ab5_littlereverb.aif");
~shortTpt21 = Buffer.read(s, "fractus_i_audio/21_B5_littlereverb.aif");
~shortTpt22 = Buffer.read(s, "fractus_i_audio/22_F3_lotsreverb.aif");
~shortTpt23 = Buffer.read(s, "fractus_i_audio/23_Ab3_lotsreverb.aif");
~shortTpt24 = Buffer.read(s, "fractus_i_audio/24_D4_lotsreverb.aif");
~shortTpt25 = Buffer.read(s, "fractus_i_audio/25_Bb4_lotsreverb.aif");
bellTail0 = Buffer.read(s, "fractus_i_audio/bell_tail0.aif");
bellTail1 = Buffer.read(s, "fractus_i_audio/bell_tail1.aif");
bellTail2 = Buffer.read(s, "fractus_i_audio/bell_tail2.aif");
bellTail3 = Buffer.read(s, "fractus_i_audio/bell_tail3.aif");
bellTail4 = Buffer.read(s, "fractus_i_audio/bell_tail4.aif");
handPop0 = Buffer.read(s, "fractus_i_audio/01_Bb_ff_withSilence.aif");
handPop1 = Buffer.read(s, "fractus_i_audio/02_A_ff_withSilence.aif");
handPop2 = Buffer.read(s, "fractus_i_audio/03_Ab_ff_withSilence.aif");
handPop3 = Buffer.read(s, "fractus_i_audio/04_G_ff_withSilence.aif");
handPop4 = Buffer.read(s, "fractus_i_audio/05_Gb_ff_withSilence.aif");
handPop5 = Buffer.read(s, "fractus_i_audio/06_F_ff_withSilence.aif");
handPop6 = Buffer.read(s, "fractus_i_audio/07_E_ff_withSilence.aif");
handPop7 = Buffer.read(s, "fractus_i_audio/08_Bb_f_withSilence.aif");
handPop8 = Buffer.read(s, "fractus_i_audio/09_Ab_mf_withSilence.aif");
handPop9 = Buffer.read(s, "fractus_i_audio/10_G_mf_withSilence.aif");
handPop10 = Buffer.read(s, "fractus_i_audio/11_Gb_f_withSilence.aif");
handPop11 = Buffer.read(s, "fractus_i_audio/12_F_mf_withSilence.aif");
handPop12 = Buffer.read(s, "fractus_i_audio/13_F_p_withSilence.aif");
handPop13 = Buffer.read(s, "fractus_i_audio/14_E_p_withSilence.aif");
handPop14 = Buffer.read(s, "fractus_i_audio/15_D_ff_withSilence.aif");
handPop15 = Buffer.read(s, "fractus_i_audio/16_D_p_withSilence.aif");
handPop16 = Buffer.read(s, "fractus_i_audio/17_hiD_ff_withSilence.aif");
handPop17 = Buffer.read(s, "fractus_i_audio/Key_Pop_withSilence.aif");
cymbalBowUnit = Buffer.read(s, "fractus_i_audio/Cymbal_Bow_seq_reverb_quiet.aif");
)

/** SynthDefs **/

{SynthDef("creakChoose",
    {arg amp = 1, pbr = 1, pan = 0, whichBuffer = 0;
     Out.ar(0,

55
Pan2.ar(
    PlayBuf.ar(
        1,
        whichBuffer,
        pbr * BufRateScale.kr(whichBuffer),
        doneAction:2),
        pan,
        amp
    );
);)
}.send(s);

SynthDef("tpttexturechoose",
    {arg pbr = 1, bufnum = 0, amp = 0.75;
    Out.ar(
        [0,1],
        LPF.ar(
            PlayBuf.ar(
                1,
                bufnum,
                pbr * BufRateScale.kr(bufnum),
                doneAction:2),
                9000,
                amp)
        );
    }
).send(s);

SynthDef("pwChoose",
    {arg amp = 1, pbr = 1, pan = 0, whichBuffer = 0;
    Out.ar(
        0,
        Pan2.ar(
            PlayBuf.ar(
                1,
                whichBuffer,
                pbr * BufRateScale.kr(whichBuffer),
                doneAction:2),
                pan,
                amp
            );
        }
    ).send(s);

SynthDef("dblConvSynth",
    {arg pbr = 1, bufnum = 0, amp = 0.75;
    Out.ar(
        [0,1],
        LPF.ar(
            PlayBuf.ar(
                1,
                bufnum,
                pbr * BufRateScale.kr(bufnum),
                doneAction:2),
                pan,
                amp
            );
        }
    ).send(s);
SynthDef("creakSeq",
{
  arg pbr = 1, amp = 0.01, whichCreak = 0, pan = 0;
  Out.ar(0,
    Pan2.ar(PlayBuf.ar(1,
        whichCreak,
        pbr * BufRateScale.kr(whichCreak),
        doneAction:2
    ),
      pan,
      amp
  )
  );
}).send(s);

SynthDef("shortTptChoose",
{
  arg amp = 1, pbr = 1, pan = 0, whichBuffer = 0, bandfreq = 2000, rq = 0.5;
  Out.ar(0,
    Pan2.ar(BPF.ar(PlayBuf.ar(1,
        whichBuffer,
        pbr * BufRateScale.kr(whichBuffer),
        doneAction:2),
        bandfreq,
        rq,
        1),
        pan,
        amp
    ));
}).send(s);

SynthDef("shortTptBursts",
{
  arg amp = 1, pbr = 1, pan = 0, whichBuffer = 0, bandfreq = 2000, rq = 0.5;
  Out.ar(0,
    Pan2.ar(
BPF.ar(
    PlayBuf.ar(
        1,
        whichBuffer,
        pbr * BufRateScale.kr(whichBuffer),
        doneAction:2),
        bandfreq, rc, 1),
        pan, amp);
    }
).send(s);

SynthDef("bellTailChoose",
    {arg pbr = 1, bufnum = 0, amp = 0.75;
        Out.ar(
            [0, 1],
            PlayBuf.ar(
                1,
                bufnum,
                pbr * BufRateScale.kr(bufnum),
                doneAction:2) * amp;
        )
    ).send(s);

SynthDef("handPopSynthRegular",
    {arg amp = 1, pbr = 1, pan = 0, whichPop = 0, maxDelTime = 0.065, delTime = 0.065, decTime = 5, bpffreq = 0.75;
        Out.ar(
            0, BPF.ar(
                Pan2.ar(
                    AllpassC.ar(
                        PlayBuf.ar(
                            1, whichPop, pbr * BufRateScale.kr(whichPop),
                            doneAction:2),
                            maxDelTime, delTime, decTime),
                            pan, amp
                        ),
                        SinOsc.kr(bpffreq, 0, 1200, 1300),
                        0.5
                    )
                )
            )
        ).send(s);
SynthDef(
  "handPopSynthSolid",
  {arg amp = 1, pbr = 1, pan = 0, whichPop = 0, maxDelTime = 0.065, delTime = 0.065, decTime = 5, bpffreq = 0.75;
   Out.ar(0,
     Mix.new([PlayBuf.ar(1, whichPop, pbr * BufRateScale.kr(whichPop), doneAction:2) * 0.035,
       BPF.ar(Pan2.ar(AllpassC.ar(PlayBuf.ar(1, whichPop, pbr * BufRateScale.kr(whichPop), doneAction:2),
                       maxDelTime, delTime, decTime, 1),
                      pan, 1),
       SinOsc.kr(bpffreq, 0, 1200, 1300), 0.5)
     ) * amp;
   )
  )}.send(s);

SynthDef(
  \cymbalBowSeq,
  {arg pbr = 1, amp = 1;
   Out.ar([0, 1],
     PlayBuf.ar(1, \cymbalBowUnit, pbr * BufRateScale.kr(\cymbalBowUnit), doneAction:2)
   ) * amp
  )}.send(s);
)

/** Tasks **/

~creakPlaySparse = Task({
  ~creakPlaySparseInner = Task({
    ...

59
Synth.new(
    "creakChoose",
    \pan, [rrand(-0.75, 0.75), -0.75, 0.75, 0].wchoose([0.72, 0.08, 0.08, 0.12]),
    \amp, rrand(0.42, 0.78),
    \whichBuffer, [~creak1, ~creak2, ~creak3, ~creak5, ~creak10, ~creak12].choose,
    \pbr, (rrand(0.08, 1.3)), s,
    \addBefore
);}

y = 11.rand;
if (y < 9,
    {rrand(0.7, 2.0)).wait},
    {rrand(0.1, 0.15)).wait});
}.loop
}.play);

~creakPlaySparseQuiet = Task({
~creakPlaySparseQuietInner = Task({
    Synth.new(
        "creakChoose",
        \pan, [rrand(-0.75, 0.75), -0.75, 0.75, 0].wchoose([0.72, 0.08, 0.08, 0.12]),
        \amp, rrand(0.175, 0.5),
        \whichBuffer, [~creak1, ~creak2, ~creak3, ~creak5, ~creak10, ~creak12].choose,
        \pbr, (rrand(0.08, 1.3)), s,
        \addBefore
    );
    y = 11.rand;
    if (y < 9,
        {rrand(0.7, 2.0)).wait},
        {rrand(0.1, 0.15)).wait});
    }.loop
    }.play);
});

~creakPlayAverage = Task({
~creakPlayAverageInner = Task({
    Synth.new(
        "creakChoose",
        \pan, [rrand(-0.75, 0.75), -0.75, 0.75, 0].wchoose([0.72, 0.08, 0.08, 0.12]),
        \amp, 0.7,
        \whichBuffer, [~creak0, ~creak1, ~creak2, ~creak3, ~creak4, ~creak5, ~creak6, ~creak7, ~creak8, ~creak9, ~creak10, ~creak11, ~creak12].wchoose([0.12, 0.065, 0.065, 0.065, 0.065, 0.065, 0.07, 0.07, 0.07, 0.07, 0.1, 0.07, 0.1, 0.07]),
        ...)});
\pbr, (rrand(0.12, 1.75)), s, 
\addBefore

); y = 10.rand;
if ( y < 8, 
{(rrand(0.33, 0.9)).wait}, 
{(rrand(0.07, 0.13)).wait}
);}.loop
}.play);

~creakPlayAverageQuiet = Task{
~creakPlayAverageQuietInner = Task{
  Synth.new(
    "creakChoose",[
    \pan, [rrand(-0.75, 0.75), -0.75, 0.75, 0].wchoose([0.72, 0.08, 0.08, 0.12]),
    \amp, rrand(0.2, 0.32),
    \whichBuffer, [~creak0, ~creak1, ~creak2, ~creak3, ~creak4, ~creak5, ~creak6, ~creak7, ~creak8, ~creak9, ~creak10, ~creak11, ~creak12].wchoose([0.12, 0.065, 0.065, 0.065, 0.065, 0.07, 0.07, 0.07, 0.07, 0.1, 0.07, 0.1, 0.07]),
    \pbr, (rrand(0.12, 1.75))], s, 
    \addBefore
  ); y = 10.rand;
if ( y < 8, 
{(rrand(0.33, 0.9)).wait}, 
{(rrand(0.07, 0.13)).wait}
);}.loop
}.play);

~creakPlayDense = Task{
~creakPlayDenseInner = Task{
  Synth.new(
    "creakChoose",[
    \pan, [rrand(-0.75, 0.75), -0.75, 0.75, 0].wchoose([0.72, 0.08, 0.08, 0.12]),
    \amp, rrand(0.62, 0.75),
    \whichBuffer, [~creak0, ~creak1, ~creak2, ~creak3, ~creak4, ~creak5, ~creak6, ~creak7, ~creak8, ~creak9, ~creak10, ~creak11, ~creak12].wchoose([0.12, 0.065, 0.065, 0.065, 0.065, 0.07, 0.07, 0.07, 0.07, 0.1, 0.07, 0.1, 0.07]),
    \pbr, (rrand(0.12, 1.75))], s, 
    \addBefore
  ); y = 9.rand;
if (}
y < 8,
   {{rrand(0.04, 0.21)}.wait},
   {{rrand(0.04, 0.21)}.wait}
);  
}.loop
}.play);

~creakPlayDenseQuiet = Task({
~creakPlayDenseQuietInner = Task({
{
  Synth.new(
"creakChoose",
  \pan, [rrand(-0.75, 0.75), -0.75, 0.75, 0].wchoose([0.72, 0.08, 0.08, 0.12]),
  \amp, 0.14,
  \whichBuffer, [~creak0, ~creak1, ~creak2, ~creak3, ~creak4, ~creak5, ~creak6, ~creak7, ~creak8, ~creak9, ~creak10, ~creak11, ~creak12].wchoose([0.12, 0.065, 0.065, 0.065, 0.065, 0.065, 0.07, 0.07, 0.07, 0.1, 0.07, 0.07, 0.07, 0.1, 0.07]),
  \pbr, (rrand(0.12, 0.175))], s,
  \addBefore
  
  y = 9.rand;
  if (  
    y < 8,
    {{rrand(0.04, 0.29)}.wait},
    {{rrand(0.04, 0.29)}.wait}
  );
  
  }.loop
  }.play);
});

~pwPlay = Task({
~pwPlayInner = Task({
{
  Synth.new(
"pwChoose",
  \pan, [rrand(-1.0, 1.0), -1, 1, 0].wchoose([0.72, 0.08, 0.08, 0.12]),
  \amp, rr(0.15, 0.46),
  \whichBuffer, [~pw0, ~pw1, ~pw2, ~pw3, ~pw4, ~pw5, ~pw6, ~pw7, ~pw8, ~pw9].wchoose([0.11, 0.11, 0.11, 0.11, 0.11, 0.12, 0.11, 0.11, 0.11, 0.11, 0.11, 0.11, 0.11, 0.11]),
  \pbr, (rrand(0.32, 1.6))], s,
  \addBefore
  
  x = 8.rand;
  if (  
    x < 8,
    {{rrand(0.2, 0.9)}.wait},
    {{rrand(1, 1.55)}.wait}
  );
  
  }.loop
  }.play);
});
~dronePlay1 = Task({
    ~dronePlayInner = Task({
        { Synth.new("tpttexturechoose",{
            \amp, 0.13,
            \bufnum, [-~drone2, -~drone4].choose,
            \pbr, (rrand(0.4, 1.25)), s,
            \addBefore
        });
        (rrand(2.5, 4.8)).wait
    }.loop
} ).play);
});

~dblConvPlay = Task({
    ~dblConvPlayInner = Task({
        { Synth.new("dblConvSynth",
            \amp, rrand(0.28, 0.35), \bufnum, [-~dblConvTpt1,
            -~dblConvTpt2].choose, \pbr, (rrand(0.5, 1.0)), s, \addBefore);
        (rrand(6.0, 7.0)).wait
    }.loop
} ).play);
});

~dblConvPlayQuiet = Task({
    ~dblConvPlayQuietInner = Task({
        { Synth.new("dblConvSynth",
            \amp, rrand(0.11, 0.14), \bufnum, [-~dblConvTpt1,
            -~dblConvTpt2].choose, \pbr, (rrand(0.5, 1.0)), s, \addBefore);
        (rrand(6.0, 7.0)).wait
    }.loop
} ).play);
});

~creakSeqAleaPlay = Task({
    ~creakSeqAleaPlayInner = Task({
        { Synth.new("creakSeq",
            [\pan, [rrand(-0.85, 0.85), -0.85, 0.85, 0].wchoose([0.4, 0.03,
            0.03, 0.54]),
            \amp, rrand(0.3, 0.46),
            \whichCreak, [-~creakSeqAlea0, -~creakSeqAlea1, -~creakSeqAlea2,
            -~creakSeqAlea3, -~creakSeqAlea4, -~creakSeqAlea5, -~creakSeqAlea6,
            -~creakSeqAlea7, -~creakSeqAlea8, -~creakSeqAlea9].choose,
            \pbr, rrand(0.95, 1.07),
        ], s, \addBefore);
        (rrand(3.6, 5.5)).wait
    }.loop
} ).play);
});

~shortTptPlayDense = Task({
    ~shortTptPlayDenseInner = Task({

if ( //precautionary "if" to stop dense creaks if trumpetCounter1 is large enough 
  (~trumpetCounter1 >= 3)
  {
    ~creakPlayDense.stop
  };
  
  ~uselessCounter = ~uselessCounter + 1;
); 

if ( //to stop both sparse and dense short trumpet tasks once trumpetCounter1 is large enough
  (~trumpetCounter1 == 6)
  {
    ~shortTptPlayDense.stop;
    ~trumpetCounter1 = 0
  };
  
  ~uselessCounter = ~uselessCounter + 1;
); 

if ( //what to do when all individual gestures are played and trumpetCounter is large enough 
  (~counterShortTptDense >= ~shortTptDenseNumberOfGestures).and(~trumpetCounter1 >= 4)
  {
    ~shortTptPlayDense.stop;
    ~counterShortTptDense = 0;
    ~counterShortTptSparse = 0;
    ~trumpetCounter1 = ~trumpetCounter1 + 1;
    ~shortTptSparseNumberOfGestures = rrand(11, 14);
    ~shortTptPlaySparse.start;
    ~creakPlayDense.stop;
  };
  
  ~counterShortTptDense = ~counterShortTptDense + 0.5 // adjusted from 1 to 0.5 because it was being added twice
  };
); 

if ( //what to do when all individual gestures are played and
trumpetCounter1 is NOT large enough yet
{
  (~counterShortTptDense >=
~shortTptDenseNumberOfGestures).and(~trumpetCounter1 < 4)
},
{
  ~shortTptPlayDense.stop;
  ~counterShortTptDense = 0;
  ~counterShortTptSparse = 0;
  ~trumpetCounter1 = ~trumpetCounter1 + 1;
  ~shortTptSparseNumberOfGestures = rrand(11, 14);
  ~shortTptPlaySparse.start;
  ~creakPlayAverageQuiet.start;
  ~creakPlayDense.stop;
},
{
  ~counterShortTptDense = ~counterShortTptDense + 0.5
  // here is the second addition. this is a stupid way
  // to get counterShortTptDense to increase by a total of 1 upon a full loop of
  // the task...but it works!
};
}
rrand(0.07, 0.35).wait;
Synth.new("shortTptBursts",
{
  \pan, [rrand(-0.85, 0.85), -0.85, 0.85, 0].wchoose([0.4, 0.03,
0.03, 0.54]),
  \amp, 1.35,
  \whichBuffer, [~shortTpt0, ~shortTpt1, ~shortTpt2, ~shortTpt3,
~shortTpt4, ~shortTpt5, ~shortTpt6, ~shortTpt7, ~shortTpt8, ~shortTpt9,
~shortTpt10, ~shortTpt11, ~shortTpt12, ~shortTpt13, ~shortTpt14, ~shortTpt15,
~shortTpt16, ~shortTpt17, ~shortTpt18, ~shortTpt19].wchoose([0.2, 0.1, 0.1,
0.07, 0.07, 0.07, 0.07, 0.07, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05, 0.05, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005]),
  \pbr, (2**(-1/6), 2**(-1/12), 2, 2**(1/12),
2**(1/6)).wchoose((0.14, 0.24, 0.24, 0.24, 0.14)),
  \bandfreq, rrand(200, 4500),
  \rq, 0.95,
}, s, \addBefore);
}.loop
}.play);
});

~shortTptPlaySparse = Task({
  ~shortTptPlaySparseInner = Task({
    if { (~counterShortTptSparse >=
~shortTptSparseNumberOfGestures).and(~trumpetCounter1 >= 3)
{  
  ~shortTptPlaySparse.stop;
  ~counterShortTptSparse = 0;
  ~counterShortTptDense = 0;
  ~trumpetCounter1 = ~trumpetCounter1 + 1;
  ~shortTptDenseNumberOfGestures = rrand(40, 60);
  ~shortTptPlayDense.start 
},
{
  ~counterShortTptSparse = ~counterShortTptSparse + 0.5;
};
);
}

if (  
( ~counterShortTptSparse >= ~shortTptSparseNumberOfGestures).and( ~trumpetCounter1 < 3)  
),
{
  ~shortTptPlaySparse.stop;
  ~counterShortTptSparse = 0;
  ~counterShortTptDense = 0;
  ~trumpetCounter1 = ~trumpetCounter1 + 1;
  ~shortTptDenseNumberOfGestures = rrand(40, 60);
  ~shortTptPlayDense.start;
  ~creakPlayAverageQuiet.stop;
  ~creakPlayDense.start 
},
{
  ~counterShortTptSparse = ~counterShortTptSparse + 0.5;
};
);

rrand(1.3, 2.6).wait;
Synth.new("shortTptBursts",
[  
  \pan, [rrand(-0.85, 0.85), -0.85, 0.85, 0].wchoose([0.47, 0.04, 0.04, 0.45]),
  \amp, 1.46,
  \whichBuffer, [-shortTpt0, -shortTpt1, -shortTpt2, -shortTpt3,  
  -shortTpt24, -shortTpt25].choose,
  \pbr, rrand(0.93, 1.1),
  \bandfreq, rrand(200, 4500),
  \rq, 0.95,
], 8, \addBefore);
}.loop
}.play);
});

~bellTailPlay = Task({
~bellTailPlayInner = Task({
    
    Synth.new("bellTailChoose", [amp, rrand(0.1, 0.15),
        bufnum, [-bellTail0, -bellTail1, -bellTail2, -bellTail3,
        -bellTail4].choose,
        pbr, (rrand(0.4, 1.3))], s, \addBefore);
    
    ([rrand(5.5, 6.0), rrand(0.08, 0.27), rrand(1.0, 1.4)].wchoose([0.13, 0.78, 0.09])).wait
        }.loop
    }
    }.play);
});

~playGesture22 = Task({
    {PlayBuf.ar(2, ~g16, BufRateScale.kr(~g16), doneAction:2)}.play;
    ~dblConvPlay.stop;
    ~dblConvPlayQuiet.start;
    7.wait;
    ~creakSeqAleaPlay.stop;
    8.wait;
    ~creakPlayAverage.stop;
    9.wait;
    ~bellTailPlay.start;
});

~popPlayRegular = Task({
    ~popCounter = 0;
    ~popRegularAmp = 0;
    b = 0.093;
    c = 0.1422;
    d = 0.1916;
    e = 0.2701;
    f = 0.313;
    g = 0.3917;

    ~popPlayInner = Task({
        
        ~popRegularAmp = min(7, ~popRegularAmp + 1);
        Synth.new("handPopSynthRegular",[pan, [rrand(-0.9, 0.9), -0.9, 0.9, 0].wchoose([0.72, 0.06, 0.06, 0.16]),
            amp, ~popRegularAmp,
            whichPop, [-handPop7, -handPop8, -handPop9, -handPop10,
            -handPop11, -handPop12, -handPop13, -handPop15].choose,
            pbr, rrand(0.3, 1.7),
            bpffreq, [0.125/16, 0.125/8, 0.125/4, 0.125/2, 0.5].choose,
            maxDelTime, 1,
            delTime, [max(0.0625,b), max(0.125,c), min(0.25, d),
            max(0.25,e), min(0.5,f), min(0.5, g)].wchoose([0.07, 0.5, 0.11, 0.11, 0.1, 0.011]),
            decTime, rrand(15.0, 30.0)]}, s, \addBefore);
        
        if (}
    );
});
~popCounter >= 4
)
{
  b = b * rrand(0.975, 0.99);
c = c * rrand(0.986, 0.999);
d = d * rrand(1.01, 1.013);
e = e * rrand(0.993, 0.995);
f = f * rrand(1.016, 1.021);
g = g * rrand(1.009, 1.014);
},
{
  ~popCounter = ~popCounter + 1;
}
);

\[1.5, 1, 0.5, 2\].choose.wait;
}.loop
}.play);
});

~popPlaySolid = Task({
  ~popPlaySolidInner = Task({
    Synth.new("handPopSynthSolid",\n      \pan, [rrand(-0.5, 0.5), -0.5, 0.5, 0].wchoose([0.45, 0.05, 0.05, 0.45]),
      \amp, ~popSolidAmp,
      \whichPop, [-handPop0, -handPop1, -handPop2, -handPop3,
      -handPop4, -handPop5, -handPop6, -handPop14].choose,
      \pbr, rrand(~popSolidPbrLo, ~popSolidPbrHi),
      \bpffreq, [0.125/16, 0.125/8, 0.125/4, 0.125/2, 0.5,
      0.75].choose,
      \maxDelTime, 1,
      \delTime, [0.0625, 0.125, 0.25, 0.5].wchoose([~thirtySecondNoteWeight, ~sixteenthNoteWeight,
      ~eighthNoteWeight, ~quarterNoteWeight]),
      \decTime, rrand(~popSolidDecayMin, ~popSolidDecayMax)
    ], s, \addBefore);
    [~waitTime1, ~waitTime2, ~waitTime3].wchoose([~waitTime1Weight, ~waitTime2Weight, ~waitTime3Weight]).wait
  }.loop
  }.play);
});

~cymbalBowPlay = Task({
  ~cymbalBowPlayInner = Task({
    Synth.new("cymbalBowSeq",\n      \amp, ~cymbalBowAmp,
      \pbr, (rrand(0.09, 0.75))
    ], s, \addBefore);
    (rrand(0.5, 1.5)).wait
  }.loop
  )
};
}.play);

~playGesture23 = Task{

  b=0.093;c=0.1422;d=0.1916;e=0.2701;f=0.313;g=0.3917;
  ~popPlayRegular.start;
  19.9.wait;
  {PlayBuf.ar(2, ~g17, BufRateScale.kr(~g17), doneAction:2)*0.81}.play;
  6.5.wait;
  ~bellTailPlay.stop;
  4.2.wait;
  {PlayBuf.ar(2, ~g19, BufRateScale.kr(~g19), doneAction:2)*0.6}.play;
  5.5.wait;
  ~thirtySecondNoteWeight = 0;
  ~sixteenthNoteWeight = 0;
  ~eighthNoteWeight = 1;
  ~quarterNoteWeight = 0;
  ~popSolidAmp = 3.5;
  ~waitTime1 = 2.5;
  ~waitTime2 = 2.0;
  ~waitTime3 = 0.5;
  ~waitTime1Weight = 0.5;
  ~waitTime2Weight = 0.5;
  ~waitTime3Weight = 0;
  ~popSolidDecayMin = 10.0;
  ~popSolidDecayMax = 15.0;
  ~popSolidPbrLo = 0.43;
  ~popSolidPbrHi = 1.8;
  ~popPlaySolid.start;
  ~eighthNoteWeight = 0.85;
  ~quarterNoteWeight = 0.15;
  ~cymbalBowAmp = 0.52;
  ~cymbalBowPlay.start;
  {PlayBuf.ar(2, ~creak1, rrand(0.06, 0.1), doneAction:2)*2.4}.play;
  {PlayBuf.ar(2, ~g18, BufRateScale.kr(~g18), doneAction:2)*0.9}.play;
  6.wait;
  ~thirtySecondNoteWeight = 0;
  ~sixteenthNoteWeight = 0.2;
  ~eighthNoteWeight = 0.5;
  ~quarterNoteWeight = 0.3;

});

~fadePopPlaySolid = Task{

  while (~popSolidAmp > 0.03),
  {
    ~popSolidAmp = max(0.0, ~popSolidAmp * ~popSolidFadeMultiplier);
    0.5.wait;
  }

};

~popPlaySolid.stop;
~fadePopPlaySolid.stop;

});
~playGesture24 = Task({
    {PlayBuf.ar(2, ~g20, BufRateScale.kr(~g20), doneAction:2)}.play;
    ~popSolidAmp = 3.5;
    13.wait;
    ~popSolidFadeMultiplier = 0.67;
    ~fadePopPlaySolid.start;
    4.wait;
    ~cymbalBowPlay.stop;
});

~playGesture25 = Task({
    {PlayBuf.ar(2, ~g25, BufRateScale.kr(~g25), doneAction:2)*0.8}.play;
    5.wait;
    ~thirtySecondNoteWeight = 0;
    ~sixteenthNoteWeight = 1;
    ~eighthNoteWeight = 0;
    ~quarterNoteWeight = 0;
    ~popSolidAmp = 3.3;
    ~waitTime1 = 2;
    ~waitTime2 = 0.5;
    ~waitTime3 = 1;
    ~waitTime1Weight = 0;
    ~waitTime2Weight = 0.8;
    ~waitTime3Weight = 0.2;
    ~popSolidDecayMin = 7.5;
    ~popSolidDecayMax = 10.0;
    ~popSolidPbrLo = 0.43;
    ~popSolidPbrHi = 1.8;
    ~popPlaySolid.start;
    {PlayBuf.ar(2, ~g21, BufRateScale.kr(~g21), doneAction:2)*0.56}.play;
    ~cymbalBowAmp = 0.38;
    ~cymbalBowPlay.start;
    4.5.wait;
    ~thirtySecondNoteWeight = 0;
    ~sixteenthNoteWeight = 0.6;
    ~eighthNoteWeight = 0.25;
    ~quarterNoteWeight = 0.15;
    ~waitTime2 = 1.5;
    ~waitTime1Weight = 0.2;
    ~waitTime2Weight = 0.4;
    ~waitTime3Weight = 0.4;
});

~playGesture26 = Task({
    {PlayBuf.ar(2, ~g22, BufRateScale.kr(~g22), doneAction:2)*1.2}.play;
    ~popSolidAmp = 3.3;
    16.wait;
    ~popSolidFadeMultiplier = 0.5;
    ~fadePopPlaySolid.start;
    2.wait;
    ~cymbalBowPlay.stop;
});

~playGesture27 = Task({
    {PlayBuf.ar(2, ~g23, BufRateScale.kr(~g23), doneAction:2) * 0.7}.play;
5.5.wait;
~thirtySecondNoteWeight = 0;
~sixteenthNoteWeight = 1;
~eighthNoteWeight = 0;
~quarterNoteWeight = 0;
~popSolidAmp = 3.0;
~waitTime1 = 2;
~waitTime2 = 0.5;
~waitTime3 = 1;
~waitTime1Weight = 0;
~waitTime2Weight = 0.9;
~waitTime3Weight = 0.1;
~popSolidDecayMin = 9.0;
~popSolidDecayMax = 12.0;
~popSolidPbrLo = 0.43;
~popSolidPbrHi = 1.8;
~popPlaySolid.start;
~cymbalBowAmp = 0.43;
~cymbalBowPlay.start;
4.wait;
~thirtySecondNoteWeight = 0;
~sixteenthNoteWeight = 0.55;
~eighthNoteWeight = 0.3;
~quarterNoteWeight = 0.15;
~waitTime1 = 1.5;
~waitTime1Weight = 0.15;
~waitTime2Weight = 0.55;
~waitTime3Weight = 0.3;
});
~playGesture28 = Task({
{PlayBuf.ar(2, ~g24, BufRateScale.kr(~g24), doneAction:2)}.play;
~popSolidAmp = 3.0;
4.wait;
~dblConvPlayQuiet.stop;
9.wait;
~popSolidFadeMultiplier = 0.62;
~fadePopPlaySolid.start;
3.5.wait;
~cymbalBowPlay.stop;
});

~playGesture30 = Task({
{PlayBuf.ar(2, ~g27, BufRateScale.kr(~g27), doneAction:2)}.play;
~thirtySecondNoteWeight = 0;
~sixteenthNoteWeight = 1;
~eighthNoteWeight = 0;
~quarterNoteWeight = 0;
~popSolidAmp = 2.0;
~waitTime1 = 0.5;
~waitTime2 = 1;
~waitTime3 = 1;
~waitTime1Weight = 0.33;
~waitTime2Weight = 0.33;
~waitTime3Weight = 0.34;
~popSolidDecayMin = 5.0;
~popSolidDecayMax = 7.0;
4.5.wait;
~popSolidPbrLo = 0.43;
~popSolidPbrHi = 1.0;
~popPlaySolid.start;
4.wait;
~thirtySecondNoteWeight = 0;
~sixteenthNoteWeight = 0.3;
~eighthNoteWeight = 0.4;
~quarterNoteWeight = 0.3;
~waitTime1 = 2;
~waitTime2 = 1;
~waitTime3 = 0.5;
~waitTime1Weight = 0.5;
~waitTime2Weight = 0.35;
~waitTime3Weight = 0.15;
~popSolidDecayMin = 12.0;
~popSolidDecayMax = 18.0;

});

~playGesture32 = Task({
  {PlayBuf.ar(2, ~g29, BufRateScale.kr(~g29), doneAction:2)}.play;
  ~popSolidAmp = 2.0;
  ~popSolidFadeMultiplier = 0.67;
  ~fadePopPlaySolid.start;
  7.wait;
  ~thirtySecondNoteWeight = 1;
  ~sixteenthNoteWeight = 0;
  ~eighthNoteWeight = 0;
  ~quarterNoteWeight = 0;
  ~waitTime1 = 0.5;
  ~waitTime2 = 0.25;
  ~waitTime3 = 3;
  ~waitTime1Weight = 1;
  ~waitTime2Weight = 0;
  ~waitTime3Weight = 0;
  ~popSolidDecayMin = 30.0;
  ~popSolidDecayMax = 40.0;
  ~popSolidPbrLo = 0.6;
  ~popSolidPbrHi = 0.8;
  ~popSolidAmp = 4.2;
  ~popPlaySolid.start;
  1.9.wait;
  ~popSolidAmp = 1.7;
  ~waitTime1 = 0.5;
  ~waitTime2 = 1;
  ~waitTime3 = 1.5;
  ~waitTime1Weight = 0.33;
  ~waitTime2Weight = 0.29;
  ~waitTime3Weight = 0.34;
  ~popSolidPbrLo = 0.45;
  ~popSolidPbrHi = 1.2;
  ~thirtySecondNoteWeight = 0.08;
  ~sixteenthNoteWeight = 0.36;
  ~eighthNoteWeight = 0.36;
  ~quarterNoteWeight = 0.2;
  ~popSolidDecayMin = 10.0;
  ~popSolidDecayMax = 14.0;

});
~playGesture33 = Task(
    {PlayBuf.ar(2, ~g30, BufRateScale.kr(~g30), doneAction:2) * 0.9}.play;
    21.wait;
    ~popSolidAmp = 1.7;
    ~popSolidFadeMultiplier = 0.65;
    ~fadePopPlaySolid.start;
    5.5.wait;
    ~thirtySecondNoteWeight = 0.05;
    ~sixteenthNoteWeight = 0.4;
    ~eighthNoteWeight = 0.4;
    ~quarterNoteWeight = 0.15;
    ~popSolidAmp = 1.7;
    ~waitTime1 = 2.0;
    ~waitTime2 = 1.0;
    ~waitTime3 = 0.5;
    ~waitTime1Weight = 0.2;
    ~waitTime2Weight = 0.4;
    ~waitTime3Weight = 0.4;
    ~popSolidDecayMin = 5.0;
    ~popSolidDecayMax = 8.0;
    ~popSolidPbrLo = 0.48;
    ~popSolidPbrHi = 1.8;
    ~popPlaySolid.start;
    {PlayBuf.ar(2, ~g31, BufRateScale.kr(~g31), doneAction:2)}.play;
    11.5.wait;
    ~cymbalBowPlay.start;
    6.5.wait;
    ~creakPlaySparseQuiet.start;
    12.wait;
    ~popSolidAmp = 1.6;
    ~popSolidFadeMultiplier = 0.65;
    ~fadePopPlaySolid.start;
    ~creakPlaySparseQuiet.stop;
    ~creakPlayAverageQuiet.start;
    8.5.wait;
    ~creakPlayAverageQuiet.stop;
    ~creakPlayDenseQuiet.start;
    ~cymbalBowPlay.stop;
    2.wait;
    ~creakPlayDenseQuiet.stop;
    ~creakPlayDense.start;
    2.wait;
    ~creakPlayDense.stop;

});

/** Executable Audio Files **/

/* 01 */ {PlayBuf.ar(2, ~g01, BufRateScale.kr(~g01), doneAction:2)*1.2}.play
/* 02 */ {PlayBuf.ar(2, ~g02, BufRateScale.kr(~g02), doneAction:2)*1.2}.play
/* 03 */ {PlayBuf.ar(2, ~g03, BufRateScale.kr(~g03), doneAction:2)*0.2}.play
/* 04 */ {PlayBuf.ar(2, ~g04, BufRateScale.kr(~g04), doneAction:2)*0.7}.play
/* 05 */ {PlayBuf.ar(2, ~g05, BufRateScale.kr(~g05), doneAction:2)}.play
/* 06 */ {PlayBuf.ar(2, ~g06, BufRateScale.kr(~g06), doneAction:2)}.play
/* 07 */ {PlayBuf.ar(2, ~g07, BufRateScale.kr(~g07), doneAction:2)}.play
/* 08 */ {PlayBuf.ar(2, ~g08, BufRateScale.kr(~g08), doneAction:2)}.play
/* 09 */ {PlayBuf.ar(2, ~g09, BufRateScale.kr(~g09), doneAction:2)}.play
/* 10 */ {PlayBuf.ar(2, ~g10, BufRateScale.kr(~g10), doneAction:2)}.play
/* 11 */ ~creakPlaySparse.start
/* 12 */ {PlayBuf.ar(2, ~g11, BufRateScale.kr(~g11), doneAction:2)}.play
/* 13 */ {PlayBuf.ar(2, ~g12, BufRateScale.kr(~g12), doneAction:2)}.play
/* 14 */ ~creakPlaySparse.stop;~creakPlayAverage.start
/* 15 */ ~creakPlayAverage.stop;~creakPlaySparse.start
/* 16 */ ~creakPlaySparse.stop;~pwPlay.start
/* 17 */ {PlayBuf.ar(2, ~g14, BufRateScale.kr(~g14),
doneAction:2)*1.1}.play;~dronePlay1.start
/* 18 */ ~dronePlay1.stop;~pwPlay.stop
/* 19 */ {PlayBuf.ar(2, ~g15, BufRateScale.kr(~g15), doneAction:2)*0.87}.play
/* 20 */ ~shortTptDenseNumberOfGestures = rrand(40,
60);~creakPlayDense.start;~shortTptPlayDense.start;~dblConvPlay.start
/* 21 */ ~creakSeqAleaPlay.start;~creakPlayAverageQuiet.stop;~creakPlayAverage.start;
/* 22 */ ~playGesture22.start
/* 23 */ ~playGesture23.start
/* 24 */ ~playGesture24.start
/* 25 */ ~playGesture25.start
/* 26 */ ~playGesture26.start
/* 27 */ ~playGesture27.start
/* 28 */ ~playGesture28.start
/* 29 */ {PlayBuf.ar(2, ~g26, BufRateScale.kr(~g26), doneAction:2)}.play
/* 30 */ ~playGesture30.start
/* 31 */ {PlayBuf.ar(2, ~g28, BufRateScale.kr(~g28), doneAction:2)}.play
/* 32 */ ~playGesture32.start
/* 33 */ ~playGesture33.start
/* 34 */ {PlayBuf.ar(2, ~g32, BufRateScale.kr(~g32), doneAction:2)}.play
FRACTUS I

FOR TRUMPET IN C AND ELECTRONIC SOUND

PERFORMANCE SCORE

ELI M. FIELDSTEEL

©2010
Fractus I is titled after the cloud formation fractus, consisting of small, ragged clouds that have usually broken off from larger clouds and are often distorted by strong winds. Fractus clouds have irregular patterns, change shape frequently, and can form or dissipate rapidly. Fractus I consists of several musical sections, each with its own distinct shape and character. These sections, like the cloud formation, can change abruptly or gradually. The gestures are sometimes aggressive, active and independent, while other times the various components seamlessly blend into a unified idea.

Performance Notes

The electronic part to Fractus I should ideally be performed using SuperCollider, an environment and programming language for real time audio synthesis, freely available for download at http://supercollider.sourceforge.net/. A complete collection of electronic materials may be obtained by contacting Eli Fieldsteel at eli [dot] fieldsteel [at] gmail [dot] com. These materials include a README.txt file, which provides technical information about performing the piece with SuperCollider.

As an alternative, Fractus I may be performed as a fixed media piece by running an audio file (also included in the electronic materials) alongside the performer. This method should be considered inferior to using SuperCollider; part of the electronic score is determined randomly upon each performance with SuperCollider,
therefore much of the overall effect is lost when performing *Fractus I* as a fixed media piece. Furthermore, although performing the piece as a fixed media work sidesteps any SuperCollider-related complications, the trumpet player will have greatly reduced temporal flexibility, as synchronizing the live part with the electronics may be significantly more difficult.

If using SuperCollider, more RAM and a faster processor will ensure a more reliable performance. A 2.0 GHz processor with at least 4 GB of RAM is recommended for optimal performance, although the electronics may run on machines with lower technical specifications.

If performing *Fractus I* with SuperCollider, a human score follower is required to activate each electronic cue as it occurs in the music. This implementation allows for a significant degree of flexibility between the performer and score follower, but also requires that the two individuals work together by actively listening to one another. If the performer plays through a particular section too quickly or too slowly, the score follower can activate the next cue slightly earlier or later, respectively. More technical information on score following can be found in the README.txt file.

The live trumpet should be amplified and sent through the same stereo amplification system as the electronic part. A modest amount of reverb may be added to the live trumpet output if convenient and desirable. It
is advisable to have an additional technician managing output levels in order to maintain a proper balance, as the score follower will most likely be occupied with activating cues.

With the exception of a few moments from m. 105 onward, the piece does not demand a tremendous degree of strict synchronization. Many of the electronic phrases and trumpet phrases overlap with one another (for example, mm. 61 – 94), and the trumpet rhythms are often somewhat amorphous. As a result, it is less important to coordinate the acoustic and electronic components on a microscopic level and more important to coordinate macroscopic sections and arrivals. For example, the trumpet and electronics should prioritize the simultaneous arrival on beat three of m. 56, while the minute rhythmic details approaching this climactic point are less critical.

The tempo of Fractus I is largely governed by seconds; at any given point, the piece is at either 60 beats per minute, 120 beats per minute, or spatially notated. For rehearsal and/or performance purposes, it may be desirable for the performer and/or score follower to use time-keeping devices, although the piece does not required it.

The aleatoric section from mm. 95 – 104 requires some explanation. Unlike traditional notation, the metric indications (1/X, 3/X) do not reference a beat value. Instead, the numerator indicates the number of “blocks” of time within the measure. The durations of these blocks are indicated in brackets above each measure.

This section uses boxed notation to specify gestures for the performer. This notation indicates a pitch range on the staff and a bracketed gesture above the staff. The performer should play the gesture for as long as the
arrow continues, pausing at will, on any of the pitches within the range, in any order. If a double arrow appears on the staff, the performer may choose any pitch for that gesture. If a box contains more than one gesture (as in m. 96), the performer should play a combination of these gestures, in any order, over their respective pitch ranges. Some boxed gestures carry into other measures and are incorporated with new gestures. A dashed arrow indicates that the gesture or gestures are to be gradually eliminated or gradually incorporated, as indicated in the score.

The durations of mm. 95 – 100 are determined by the electronic part, and not by either the score follower or performer. The most predominant sound within these six measures is that of the alternating dense and sparse trumpet notes, indicated in the treble staff of the electronic part. Both the performer and score follower should actively listen for these sounds in order to remain synchronized with the score. Once the sparse trumpet tones have disappeared altogether, measure 101 has begun and the score follower should activate Cue 22 immediately. Measure 102 will continue indefinitely until the score follower activates Cue 23; the score follower should try to estimate 40 seconds as best as possible, although the total duration of the measure need not be absolutely precise. The transition from mm. 103 – 104 is autonomous, but subtle; the trumpet player should make the best approximation possible of when to play the gestures indicated in m. 104, but again, this transition need not be absolutely precise.
There are three measures in *Fractus I* with a time signature indication of 1/4 (mm. 106, 148, and 161). The purpose of these measures is to provide the trumpet player and/or score follower with a window of time in which to hear and mentally synchronize with the pulse created by the electronic hand pops. The repeat indications above these measures are recommendations, however, if the piece is performed as a fixed media piece, they are exact requirements.

The live trumpet player determines the length of measure 106. He or she should give a physical cue for the score follower in order to accurately place Cue 24.

The score follower determines the length of measures 148 and 161. There is no physical cue necessary here, as the trumpet player does not enter until a number of beats after the onset of the cue.

*Fractus I* includes quarter tone accidentals. The symbol ♭ indicates a quarter tone flat and the symbol♯ indicates a quarter tone sharp. These notes should be obtained by lipping the pitch down or up, as indicated in the score.

The feathered beaming (in m. 22, for example) is essentially a written-out *accelerando*. The beamed note durations should become progressively shorter.

The grace note gestures (mm. 4, 6, 9, etc.) are note ornamentations. These gestures should resemble an echo-like effect and can be rhythmically approximated. Some of these grace note gestures incorporate feathered
beaming and quarter tones as well. The parenthesized rests above or below the grace notes indicate the total
duration of the grace note gesture.

*Fractus I* includes one hand pop in the final measure. This is executed by fingering the indicated pitch and
bringing an open palm down on the mouthpiece, while still attached to the trumpet, in a swift and forceful motion.
FRACTUS I.

Tpt.  
- mf  
- p  
- ~ 5"  
- ~ 3"  
- ~ 3"  

Cues  
- ppp  
- cresc.  

Elec.  

Tpt.  
- mp  
- pp  
- p  
- mf  

Cues  
- pitches expand outward  

Elec.  
- percussive clinks  
- thunk  

- 15 mb  
- pp cresc.  

85
50
Tpt.

54
Tpt.

Cues

Elec.

(cresc.)

(cresc.)

clinks

thunk

mf

mf

becomes

low rumble

86
FRACTUS I.

Tpt.

Cues

Elec.


sparse
creaks


denser


FRACTUS I.

Tpt.

Cues

Elec.


FRACTUS I.

Tpt.

Cues

Elec.


FRACTUS I.

Tpt.

Cues

Elec.
FRACTUS I.

Tpt.

remove mute

~ .
rubato, with emotion

p——mf

Cues

16

17

Elec.

small crunches

p

mp

metallic swells

Tpt.

p——mf

Cues

Elec.

accented notes should "jump out" of the texture
improvise on the bracketed gestures within the given range

dense, agitated trumpet gestures

glassy fall

dense creaks

improvise only these two gestures

gradually introduce these gestures

ggradually eliminate this gesture

sparse, more distant

dense, more distant

becomes low rumble

more creaks

improvise on only these two gestures

gradually introduce these gestures

ggradually eliminate this gesture

sparse

more creaks

changes
gradually introduce this gesture

improvise on only these three gestures

improvise on only these two gestures

percussive sounds become more pitched

dense

sparse

straight mute
FRACTUS I.

102
Tpt.

Cues

bell-like clusters

Elec.

improvise on only these three gestures

103
Tpt.

improvise on only these three gestures

Cues

23

Elec.

improvise on only these two gestures

hand pops begin to lock into an 8th- & 16th-note pattern

percussive sounds briefly become pitched

hand pops

percussive sounds

briefly become pitched

improvise on only these two gestures