Spectral Analysis, Editing, and Resynthesis: Methods and Applications

Michael Kateley Klingbeil

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Musical Arts
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2009

ABSTRACT

Spectral Analysis, Editing, and Resynthesis: Methods and Applications

Michael Kateley Klingbeil

This document describes the design, development, and applications of cross-platform software for audio analysis, editing, and resynthesis. Analysis is accomplished using a variation of the McAulay-Quatieri technique of peak interpolation and partial tracking. Linear prediction of the partial amplitudes and frequencies is used to determine the best continuations for sinusoidal tracks. A high performance user interface supports flexible selection and immediate manipulation of analysis data, cut and paste, and unlimited undo/redo. Hundreds of simultaneous partials can be synthesized in real-time and documents may contain thousands of individual partials dispersed in time without degrading performance. A variety of standard file formats, including the Sound Description Interchange Format (SDIF), are supported for import and export of analysis data. Specific musical and compositional applications, including those in works by the author, are discussed.

Contents

List of Figures

List of Tables

Acknowledgements

I wish to thank my advisor Tristan Murail and committee members Brad Garton and Fred Lerdahl for their insight and support throughout my time at Columbia University (and beyond). Thanks also goes to James Beauchamp for introducing me to the details of signal processing, to Rick Taube for revealing the realm of the metalevel, and to Karl Klingbeil for opening my ears to the world of contemporary music. Special gratitude goes to Anita Buckmaster for her unwavering encouragement and assistance.

This dissertation is dedicated to my parents, Laura Kateley and Jack Klingbeil.

September 17, 2008

1. Introduction

1.1 Timbre and the Computer

Considerable energy and effort has been applied to further the understanding of timbre, one of music's most elusive features. The American National Standards Institute defines timbre in a purely exclusionary manner: "... that attribute of sensation in terms of which a listener can judge that two sounds having the same loudness and pitch are dissimilar." (ANSI S1.1-1960(R1976)-12.9) This rather unsatisfying description essentially relegates all aspects of sound other than pitch and loudness to a single perceptual category. Since at least the beginning of the twentieth-century, the conception of timbre has taken on an increasingly important role in Western compositional practice. In 1922 Arnold Schoenberg asserted, "The evaluation of tone color, the second dimension of tone, is in a much less cultivated, much less organized state than is the aesthetic evaluation of pitch. Nevertheless, we go right on boldly connecting sounds with one another, contrasting them with one another, simply by feeling... " (Schoenberg 1983). More than eight decades later, this statement still seems regrettably accurate. Schoenberg goes on to call for a theory of timbre, analogous to pitch theories, that will enable the composition of "tone color melodies." While falling far short of a unified theory, scientific and artistic efforts of recent decades have achieved a greatly expanded understanding of timbre. For example, while Schoenberg describes tone color as a singular "second dimension" of tone, there is now strong evidence suggesting that timbre is in some sense multi-dimensional (Grey 1977). It is now theorized that sensations of pitch and timbre are closely intertwined, and that we likely experience different classes of pitch sensation (spectral pitches and virtual pitches) (Terhardt et al.

1982). What is undeniable, whether a composer grounds her work in theory, or spirituality, or chooses to work "simply by feeling," is that technological resources have had a profound impact on recent musical practice, and that an understanding of timbre is essential to effectively harnessing these resources.

The application of the digital computer, in particular, has done much to expand the practical range of sonic possibilities and also to enable the theoretical and empirical study of timbre. In a prophetic lecture given in 1936, Edgard Varèse proposed a vision of music made possible by machine:

> ... the new musical apparatus I envisage, able to emit sounds of any number of frequencies, will extend the limits of the lowest and highest registers ... Not only will the harmonic possibilities of the overtones be revealed in all their splendor, but the use of certain interferences created by the partials will represent an appreciable contribution. ... The time will come when the composer, after he has graphically realized his score, will see this score automatically put on a machine that will faithfully transmit the musical content to the listener. (Varèse and Chou 1966, 12)

Varèse was hardly the first, nor last, to make such predictions, and yet the strength of his conviction and the precision of his forecast remain striking. By 1953, when Varèse would begin his first compositional experiments with magnetic tape, numerous composers, technicians, and researchers were already at work in studios throughout the world exploring the musical applications of electronic instruments. By 1957, with the first musical applications of general purpose digital computers at Bell Labs, the stage was set for the eventual realization of Varèse's predictions.

It was in fact the advent of digital sound synthesis that made this vision possible. Computer music pioneer Max V. Matthews understood the power and generality of digital synthesis which he described in his 1963 article in *Science*: "With the aid of suitable output equipment, the numbers which a modern digital computer generates can be directly converted to sound waves. The process is completely general, *any perceivable sound can be so produced*.[1]" (Matthews 1963, 553). For the composer seeking to explore a world of timbral possibilities, the notion is seductive. And yet the computer music composer, whether

[1] My emphasis

working in 1967 or forty years later, still faces a number of formidable challenges. We can summarize these particular problems as follows: the time/cost problem, the synthesis problem, the control problem, and the compositional problem. Let us briefly consider each of these issues in turn.

Digital music production distinguished itself from all previous musical-mechanical approaches in that sound creation could occur on a time scale completely distinct from the temporal evolution of the sound itself. At first these computations were ponderous (many orders of magnitude slower than real-time), but as computational power increased over the decades, reasonably complex and credible sounds could be computed faster than real-time. The implication of this temporal decoupling is that sound complexity is no longer bound by space constraints (the number of musicians one can fit on a stage, or the number of inputs on a mixing board, for example) but by limitations of time (and/or expense). The composer now has the capability, as Varèse envisioned, to "emit sounds of any number of frequencies," and yet at the same time now grapples with a continual "compromise between interest, cost, and work" (Matthews 1963, 555). One could argue that this problem is being tackled with continual efficiency improvements in computer hardware. Compared to the IBM 704 of 1957, the laptop computer of the early twenty-first century provides a five-hundred-thousand fold increase in processing power at at fifteen-hundred times less cost (Sandowsky 2001). Nevertheless, computer musicians seem to have a particular penchant for using spare processing power.

With decreased costs and staggering increases in computing speed, significant additional challenges still remain. Although "any conceivable sound" is certainly possible, in reality, the range of potential sounds is constrained by the development of appropriate synthesis algorithms. The ability to formulate these procedures is severely limited by our understanding of timbre and psychoacoustics (Matthews 1963). This can be classified as "the synthesis problem."

The control problem is a closely associated concern. Although digital synthesis "allows composers freedom of expression at the timbral level" it also entails the "additional burden of specifying the microstructure of sound, which was formerly the domain of the

performing musician" (Loy 1989, 326). Because interesting musical sounds are rarely static, nor entirely stochastic, the precise control parameters must be specified at every instant. If this control information is not a natural result of the synthesis procedure in use, the result is an unwieldy data explosion with separate detailed control streams for each sonic parameter of interest.

Finally, the composer is faced with all of the traditional problems of composition itself — matters of intention and aesthetics, as well as possible questions of melody, counterpoint, rhythm, harmony, and/or form.

The following list summarizes the challenges and some possible approaches:

	Challenge	**Possible Approaches**
I.	Time and Cost	faster computers, higher processing density, more efficient algorithms
II.	Synthesis problem	new synthesis methods, flexible software systems
III.	Control problem	new user interfaces (visual, gestural), computer assisted composition, machine learning/listening
IV.	Compositional problem	continued exploration, theory

It is furthermore recognized that these problems are inherently interrelated. A given composition aesthetic, one that involves indeterminacy, for example, would invite a particular approach to the specification of control information. Particularly with regard to the compositional "problem," it is certain that no ultimate "solution" exists. If it did, such a discovery would spell the end of creative artistic production and individual expression. The act of composition must be seen as a continual search, for which each artist may be free to define their own goals and strategies.

The chapters that follow are concerned in large part with particular approaches to the synthesis problem, with a discussion of the resulting implications for the cost and control problems. The motivation will always be oriented towards musical goals, which will be addressed in the penultimate chapter.

1.2 Synthesis Methods

Regardless of compositional approach the practitioner of computer music always faces the same question: What sounds should I produce? Although current computer music systems offer tremendous flexibility, the choices are constrained by current theories and their implementations. In order to contextualize the work described in subsequent chapters, it may prove instructive to consider a particular taxonomy of digital synthesis techniques developed by Smith (1992). Table 1.1 maintains the original categories while adding some more recent methods. A significant development in the past decade is the emergence of hybrid approaches.

<table>
<tr><th>Time Domain Methods</th><th>Spectral Models (Frequency Domain)</th><th>Physical Models</th><th>Abstract Methods</th></tr>
<tr><td>Sampling
Concrète
Granular
Scanned Synthesis</td><td>Additive
Phase Vocoder
Sinusoidal (MQ/PARSHL)
SMS (Smith & Serra)
Subtractive
Source-filter (LPC)
UPIC (Xenakis)
FOF/FOG/PSOLA</td><td rowspan="3">Waveguide
Finite Element
(CORDIS-ANIMA)
Karplus-Strong
Modal</td><td rowspan="3">FM
AM
Waveshaping
Analog simulation
(VCO, VCA, VCF)
Chaotic systems</td></tr>
<tr><td colspan="2">Wavetable
Particles (Roads)
Concatenative
Audio Mosaicing (MPEG-7)</td></tr>
<tr><td colspan="3">Physically Informed Sonic Modeling (PhISM)</td></tr>
</table>

Table 1.1: Taxonomy of current digital synthesis techniques (after Smith 1992).

Let us first consider some of the most commonly used synthesis methods. Since the early 1990s, sampling synthesis has been the predominant sound production method used in commodity hardware and software. Sampling offers very high fidelity results (limited only by the quality of the input), but is restricted in terms of expressive and transformational possibilities. A typical sampling synthesis architecture limits transformations to amplitude envelopes, filtering, and loop control. Commercial developers are attacking the limitations of sampling in a brute force manner. It is not uncommon to find multi-gigabyte sample libraries containing complex mappings, cross-faded overlays,

and keyswitch programs. For example, a complex violin bank might include samples for legato, marcato, spiccato, staccato, and tremolo bowings all recorded at a variety of different dynamic levels. Fast secondary storage allows these massive sample banks to be streamed from disk in real-time.

Granular synthesis may be viewed as a particular specialization of sampling synthesis. The granular approach overlaps many short sound segments (generally less than 50 msec. duration) which may be derived from recorded sources or produced via other synthetic means. Usually, each sound segment is modulated by a corresponding amplitude envelope that quickly fades-in, reaches a steady state, and then fades out. Granular synthesis offers the possibility to decouple the time and frequency evolution of sound, as well as impart particular characteristics modulating between rough, smooth, or stochastic textures.

Wavetable synthesis, one of the oldest techniques, can be viewed as a type of hybrid approach. The time domain view concentrates on the particular wave shape of the table, which may be either derived mathematically, drawn interactively, or (most commonly) extracted from a recorded sample. In the frequency domain perspective, the contents of the wavetable are specified in terms of the amplitude, frequency, and phase of a set of harmonically related partials. With both approaches, wavetable interpolation is necessary to produce lively and dynamic sounds.

Particle synthesis may be viewed as a hybrid extension of granular synthesis that attempts to build up specific frequency domain content using a variety of elementary sound types which might include grains, impulses (trainlet or pulsar synthesis), or chirps (glisson synthesis). Convolution may be used to impart specific frequency domain characteristics to the "pulslet" texture. For a more detailed discussion of these methods, consult Roads's *Microsound* (2001, ch. 4).

Frequency domain models are concerned first and foremost with the spectral content of sounds. This is a particularly powerful technique because the perception of timbre is closely related to spectral content. Auditory research shows that the ear behaves as a sort of spectrum analyzer with precise physical positions along the basilar membrane of the inner ear attuned to specific frequencies. Frequency domain methods may thus be

seen as way to model synthesis on the processes of audition. Frequency domain models usually follow an analysis/synthesis paradigm in which control parameters are derived by analyzing pre-existing recordings.

Additive synthesis is one of the oldest and still most flexible of these frequency domain methods. Oscillators tuned to specific frequencies, each with time varying amplitudes, are mixed (summed) together to form a composite timbre. Typically, each oscillator produces a sine-like waveform. In some applications the frequency and/or phase[2] of each oscillator may also be time variant. While theoretically capable of generating almost any type of sound, additive synthesis is particularly unattractive in light of the control problem. A complex timbre may easily require 100 individual oscillators that, in turn, each require their own amplitude and frequency functions. This problem is typically solved by deriving synthesis data from an analysis stage. The major advantage of additive synthesis, and frequency domain methods in general, is that they are highly amenable to transformations.

The phase vocoder has been particularly favored for its support for independent time and frequency modification. Andy Moorer's *Lions Are Growing*, a work from 1978 that time stretches, compresses, and transposes speech sounds,[3] offers a clear demonstration of the artistic potential of this approach. Beginning in the early 1990s, the technique became possible on commodity computing platforms and received widespread use in both commercial sound design and music composition. Spectral models, with particular emphasis on sinusoidal models, will be taken up in further detail in subsequent chapters.

Concatenative synthesis, a category that encompasses a variety of specific techniques, may be viewed as a hybrid of spectral and time domain methods. For example, attack transients might be most effectively handled with sampling, while steady states could be executed with a spectral model. By carefully concatenating[4] sound segments derived from a large database, more convincing and dynamic note-to-note transitions and timbral modulations can be achieved than with sampling or spectral synthesis alone. A number of

[2] The phase of any non-zero frequency oscillator is by definition time varying — what is emphasized here is that phase may be modulated directly rather than being a result of a fixed or time varying frequency.

[3] The source sound is a reading of Richard Brautigan's poem of the same name. The work employs LPC-style resynthesis coupled with phase vocoding.

[4] Generally trying to preserve phase continuity

recently released commercial products such as Synful (orchestral instrument synthesis) (Lindemann 2007) and Vocaloid (singing voice synthesis) (Bonada and Serra 2007) have demonstrated the viability of this technique. A more detailed overview of concatenative methods may be found in Schwarz (2005).

In the early 1990s, Julius O. Smith predicted "abstract-algorithm synthesis seems destined to diminish in importance," (Smith 1992) and although this proved true to some extent, a resurgent interest in classic analog and digital hardware has demonstrated that these methods retain a strong following. As a result, considerable efforts in both the commercial and academic sectors have gone toward the software emulation of classical voltage controlled oscillators and filters.[5] In recent years the "vintage emulation" trend has extended to early commercial digital synthesizers, including the venerable Yamaha DX series which has been replicated in a number of software packages.[6]

It seems certain that no particular technique will ever disappear entirely from the toolbox, rather the collection will expand with the exigencies of fashion dictating the most popular methods of the time. The flexibility and modularity of current software systems suggests that hybrid synthesis methods will become increasingly important. The somewhat arbitrary boundaries between approaches will blur as practitioners explore possible combinations and concatenations. After all, sampling, wavetable, granular, and pulsar synthesis differ solely on their operative time scales (and approach to periodicity) and thus could be subsumed under a single methodology.

Physical modeling represents a considerably different approach to sound production. While sampling, spectral, and abstract models are all understood in terms of their effect on the sound consumer (the ear), physical modeling is concerned with the details of the sound producer (the instrument) (Smith 1992). The physics of musical instruments serve as the starting point. Generally a set of differential equations are used to describe the possible motion (vibrations) in response to physical input (such as the motion of a bow or the stroke of a mallet). A virtual sound pressure wave can then be computed. A significant

[5] The interested reader might wish to consult Välimäki and Huovilainen (2006) as well as Stilson (2006).
[6] Most notably with Native Instruments FM-7 series (now FM-8 in its 2007 incarnation)

benefit is that a very rich and sonically convincing output can result from simple control inputs.

Despite many compelling advantages, physical modeling has yet to become pervasive in computer music composition. The lack of generality in many physical modeling implementations may be one impediment to their acceptance. Even with a pre-designed algorithm, the parameter space may be undesirably abstract and difficult to control. For example, in a woodwind model the output pitch may be a result of multiple inputs. Particular physical modeling methods (for example waveguides) may be limited to modeling certain restricted classes of instruments (for more detail see Smith 2004). Finite-difference time-domain methods (Cadoz et al. 1993) represent a more general approach to physical modeling, although the computational demands are still somewhat prohibitive for many systems of musical interest. It is certain, however, that with continued research and development and inevitable increases in processing speed, both this and other physical modeling methods will find a following. After all, one must consider that even in the relatively short history of computer music, these methods are still fairly new.

1.3 Theory Versus Implementation

Regardless of the technique of interest, the computer music practitioner is always limited by the tools and implementations at hand. In some cases the constraints of the tools may prove compositionally beneficial in their circumscription of the available musical materials. At other times, the realities of the tools (missing features, bugs, awkward interface, sluggish operation) may prove merely frustrating. Too often one finds a lack of available implementations of some of the most compelling synthesis and signal processing techniques described in the literature. In such situations computer music composers may find it necessary to build their own tools—be it in software or hardware. Computer music is an inherently multi-disciplinary endeavor involving researchers, mathematicians, programmers, engineers, theoreticians, composers, improvisers, and performers, and in most cases one individual may be closely tied to a number of these disciplines.

Over the past several decades, many such individuals have devoted considerable attention to the area of spectral modeling. The proliferation of various spectral modeling implementations attests to its appeal. Some of the technique's advantages (noted above) are especially attractive to composers — in particular the possibility to explore timbre in a manner that is directly related to real-world experience (unlike abstract algorithm synthesis). The transformative potential of spectral modeling synthesis (largely unavailable with sampling) opens up a vast space between purely mimetic or purely synthetic sounds.

Sinusoidal modeling has a proven track record of high quality resynthesis while offering numerous possibilities for novel sonic transformations (Pampin 2004). The technique (which will be described in detail in section 2.4) derives a detailed additive synthesis model from an input sampled sound signal. Something that closely resembles the original input sound (a resynthesis) can be generated by adding together a sufficient number of individual sinusoidal waves. In most cases the resynthesis will not be identical to the original sound, although it is possible to get very close and, with certain extensions to the basic technique, it is possible to recreate the original exactly.

1.3.1 Implementation History

A complete implementation of sinusoidal modeling was first detailed in paper by Robert J. McAulay and Thomas F. Quatieri (1986), who were working at MIT on the analysis and synthesis of speech signals. The basic technique is often referred to as the MQ method. This work was closely followed by the development of the PARSHL system for analysis-resynthesis of musical tones by Julius O. Smith and Xavier Serra (1987) at Stanford University. Serra extended this into the "Spectral Modeling Synthesis" (SMS) method which added residual noise modeling to the basic sinusoidal analysis (Serra 1989). At the same time Robert C. Maher (1989), working with James Beauchamp at the University of Illinois, was using sinusoidal modeling for polyphonic voice separation. Much of Maher's code was subsequently integrated into the SNDAN suite (Beauchamp 1993).

Maher's work also served as the basis for Lemur by Kelly Fitz, Lippold Haken, and Bryan Holloway (Fitz et al. 1995). Lemur, which ran on the Macintosh, was one of the first graphical user interfaces (GUIs) available on commodity hardware for interactively viewing and editing sinusoidal models. Its release was closely followed by the availability of other GUI editors including SMSTools from Xavier Serra (running on Windows), InSpect and ReSpect from Sylvain Marchand and Roberty Strandh (running on UNIX/X Windows), and a number of tools from IRCAM including Diphone (Rodet and Lefèvre 1997), Sview, and XSpect. Sadly, Lemur was never updated for newer MacOS versions. The original version of SMSTools was discontinued and then relaunched as part of the cross-platform CLAM project (Amatriain et al. 2002). Some of the IRCAM tools were made commercially available as part of the IRCAM Forum software suite. In 1999, Juan Pampin developed the ATS system which implemented sinusoidal and noise modeling in Common Lisp. A GUI editor, ATSH by Oscar Pablo Di Liscia and Pete Moss soon followed.

1.3.2 Present Work

These packages are important predecessors of the software that will be detailed in this document: SPEAR (Sinusoidal Partial Editing Analysis and Resynthesis). SPEAR was created out of a desire for software that was faster, more flexible, and easier to install and use than other options.

The following goals were kept in mind throughout the design and development phases: editing should be as fast and as easy to understand as in a time domain waveform editor; listening to transformations should be possible with no intermediate synthesis or processing stage; and high quality analyses should require only a few parameter settings. Additionally, it was desired to have interoperability with other analysis-resynthesis software by providing both SDIF (Wright et al. 1999a) and native format data exchange.

In order to offer a familiar and comfortable interface, SPEAR was written to run using the native graphics of the host operating system. Development was founded on the principle that the interface should be as close as possible to that of a first-class native

application, with little or no evidence of compatibility or GUI emulation layers. Portability is made possible using wxWidgets (http://www.wxwidgets.org), a C++ library that allows the software to be compiled for MacOS, Windows, and GTK Linux (Zeitlin 2001). SPEAR was first demonstrated at the Columbia University Computer Music Center on February 23, 2004. As of this writing, builds have been created MacOS 9, MacOS X, and Windows.

The following chapters will detail the design and evolution of this software. Chapter 2 will discuss the basic techniques of spectral and sinusoidal modeling, while chapter 3 will focus specifically on sinusoidal modeling and its implementation in SPEAR. Chapter 4 details the user interface design and implementation, and chapter 5 explores some composition applications of sinusoidal modeling.

2. Spectral Modeling

2.1 The Frequency Domain

Spectral modeling is concerned primarily with the *frequency domain* content of audio signals, as opposed to the *time domain* content. In the time domain, an audio signal may be viewed as a function of time, $x(t)$, with instantaneous amplitude as the function's range. Typically such a function is graphed with time on the abscissa (horizontal axis) and amplitude on the ordinate (vertical axis), which results in the familiar waveform representation. In the frequency domain, the same signal is represented as a new function, $X(f)$, where f corresponds to frequency. Given any frequency value f_k, the value $X(f_k)$ can be used to determine relative strength of that frequency in the given signal.[1] The function $X(f)$ defines the *spectrum* of the sound.

2.1.1 Sinusoids

In order to study frequency domain representations in more detail, the notion of frequency must be made more explicit. Any function $x(t)$ that has a precisely repeating shape is said to be periodic. In musical applications, and signal processing applications in general, the periodic functions of greatest interest are the trigonometric functions sine and cosine. A sine wave signal with a frequency of f hertz is given by the following: $x(t) = \sin(2\pi ft)$ (where t is in seconds).

Because of their relation to the unit circle, the sine and cosine functions have a periodicity of 2π. Furthermore, the cosine function may be viewed as a phase shifted form of the

[1] As we will define it, $X(f_k)$ is a complex value, so it does not directly give the amplitude of component with frequency f_k. However it is straightforward to derive it from the complex value.

sine function. We use the term *sinusoid* to denote any any signal that can be described by a function $x(t)$ of the following form:

$$x(t) = A\sin(2\pi ft + \phi) \tag{2.1}$$

where A = amplitude, f = frequency (in hertz), t = time (in seconds), and ϕ = initial phase (in radians). Note that this function is periodic with a period of $\frac{1}{f}$ seconds.

2.1.2 The Fourier Transform

The Fourier transform provides an explicit method for converting from the time domain to the frequency domain. The following defines the Fourier transform of $x(t)$:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-i2\pi ft}dt \tag{2.2}$$

A complete examination of the Fourier transform is beyond the scope of this document, but we shall consider the transform with an eye toward an intuitive and pragmatic understanding.

Recall that $x(t)$ represents a given time domain signal, and that evaluating the integral for any desired frequency value f allows us to find the contribution of frequency value f in $x(t)$. In order to understand how frequency is defined, we must explore more closely the inner portion of the integral, $e^{-i2\pi ft}$. By definition, e is the base of the natural logarithm ($= 2.7182818\ldots$) and i is the imaginary number $\sqrt{-1}$. Euler's relation is a remarkable mathematical identity that relates these constants to the sine and cosine functions:

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{2.3}$$

Note that $\cos\theta + i\sin\theta$ is a complex value (it has both real and imaginary parts). The expression $e^{-i2\pi ft}$ is a *complex exponential* that defines a *complex sinusoid* with frequency f. More generally, a complex sinusoid is given by the following (compare with the definition of a real sinusoid in equation 2.1):

$$x(t) = Ae^{i(2\pi ft+\phi)} \tag{2.4}$$

Complex sinusoids are particularly useful because they are able to encode phase shifts in a manner that is compact and easy to manipulate (Puckette 2007, 189). For example, given the complex sinusoid of equation 2.4, the properties of exponents allow us to encode both the amplitude and the initial phase shift ϕ as a single *complex amplitude* (also known as a *phasor*). Any complex amplitude A' can take the following form (for some constant phase value ϕ):

$$A' = Ae^{i\phi} \tag{2.5}$$

and our complex sinusoid now simplifies as follows:

$$\begin{aligned} x(t) &= Ae^{i(2\pi ft+\phi)} \\ &= Ae^{i2\pi ft}e^{i\phi} \\ &= Ae^{i(2\pi ft+\phi)} \\ &= A'e^{i2\pi ft} \end{aligned}$$

Other important results of Euler's formula are the following relations:

$$\sin\theta = \frac{e^{i\theta} - e^{-i\theta}}{2i} \qquad\qquad \cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

This tells us that real valued sinusoids are in fact a sum of two complex sinusoids, one with positive frequency and the other with negative frequency. Complex sinusoids are therefore simpler to manipulate because they have only a single frequency (either positive or negative).

Given Euler's identity, we may now re-express the Fourier transform as:

$$X(f) = \int_{-\infty}^{\infty} x(t)(\cos 2\pi ft - i\sin 2\pi ft)dt \tag{2.6}$$

To gain some intuitive sense of what this implies, consider the evaluation of $X(f)$ for some value f_k. Informally, we may consider $\cos 2\pi f_k t$ and $\sin 2\pi f_k t$ to be "reference" or "basis" sinusoids (each with freqeuncy f_k) that are multiplied by the time domain signal. The multiplication of $x(t)$ with one of the reference function results in a new function of time, which can be plotted as a curve. We can think of the integral as giving the total area under

the curve. The area indicates the amount of the reference sinusoid present in the signal. $X(f_k)$ is in fact a complex amplitude that encodes amplitude and initial phase (as shown in equation 2.5).

2.1.3 The Discrete Fourier Transform

In the given definition of the Fourier transform, the limits of integration are $\pm\infty$, implying signals of infinite duration. In real-world situations, we work with audio signals that, by necessity, are of finite duration and are discretely sampled. Rather than working with the continuous Fourier transform, we can calculate a discretely sampled frequency spectrum using the discrete Fourier transform (DFT). Given a sampled signal $x(n)$ with a duration of N samples, the DFT of $x(n)$ is:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi \frac{k}{N} n} \qquad k = 0, 1, 2, \ldots, N-1 \tag{2.7}$$

$$= \sum_{n=0}^{N-1} x(n) \left[\cos\left(2\pi \frac{k}{N} n\right) - i \sin\left(2\pi \frac{k}{N} n\right) \right] \tag{2.8}$$

Note that frequency is defined in terms of k, which takes on values from 0 to $N-1$. We can think of k as measuring the number of sinusoidal cycles per N samples. As a concrete example, consider a signal that is sampled 44100 times per second. Let N (the length of our signal) be 1024. If $k = 1$, then we have 1 cycle per 1024 samples. Converting to cycles per second we have:

$$f_1 = \frac{1 \text{ cycle}}{1024 \text{ samples}} \times \frac{44100 \text{ cycles}}{1 \text{ second}} \approx 43.066 \text{ Hertz} \tag{2.9}$$

Successive values of k correspond to 0 Hz, 43.07 Hz, 86.13 Hz, 129.20 Hz, etc. Thus, the frequency spectrum is sampled every 43.07 Hz. Each value of k is said to refer to a specific *frequency bin*. Note that the DFT gives us signal $X(k)$ that, like the time domain signal $x(n)$, is N samples long.

One very useful property of the Fourier transform and the DFT is that the transform is invertible. If we have the DFT $X(k)$, we may recover the time domain signal $x(n)$ as

follows:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{i2\pi\frac{k}{N}n} \qquad n = 0, 1, 2, \ldots, N-1 \tag{2.10}$$

Note that the inverse DFT is very similar to the DFT, differing only in a change of sign and a scaling factor of $\frac{1}{N}$.

As noted above, $X(k)$ is complex valued. In general, the time domain and frequency domain signals may be either real or complex valued. In audio applications the time domain signal is typically real valued only, which yields complex values in the frequency domain. Each complex amplitude $X(k) = A_k$ encodes both the amplitude and initial phase shift ϕ_k. From our definition of complex amplitude in equation 2.5 and from Euler's identity (equation 2.3), we have the following:

$$\begin{aligned} X(k) &= A_k e^{i\phi_k} \\ &= A_k \cos\phi_k + iA_k \sin\phi_k \end{aligned}$$

Since A_k and ϕ_k are constants, we can represent our complex amplitude as a complex number of the form $a + bi$ where $a = A_k \cos\phi_k$ and $b = A_k \sin\phi_k$. We can visualize this as a point (a, b) on the complex plane. The amplitude A_k of the sinusoid is the distance from the origin to the point (a, b), and the phase ϕ_k is the angle of rotation about the origin. The amplitude A_k is defined as the *magnitude* of the complex point.

$$A_k = |X(k)| = \sqrt{a^2 + b^2} \tag{2.11}$$

The phase angle ϕ_k is given by the inverse tangent function.

$$\phi_k = \tan^{-1}\frac{b}{a} \tag{2.12}$$

Equations 2.11 and 2.12 give an explicit method to convert from a complex frequency bin value to magnitude (amplitude) and phase. Note that for a real valued signal, phase is measured relative to a cosine function. Phases may be expressed in sine phase via a phase shift of $-\frac{\pi}{2}$ radians.

2.1.4 The Short-Time Fourier Transform

While almost all musical signals have spectra that vary significantly over time, many sounds exhibit spectra that are locally stable. Such signals may be well-represented by a series of spectral "snapshots" or "frames" that capture the average spectral content and, in sequence, represent the time varying spectrum of the sound. Such an analysis is called a short-time Fourier transform (STFT). Each spectral frame is computed by means of a DFT of length N and is centered on a particular sample n. Successive frames are computed by advancing some number of samples H (the hop size) and computing another DFT.

As noted above, the DFT is a discrete sampling of the frequency spectrum. When a signal consists of a sinusoid with a period that is an integer factor of the DFT length N, then the sinusoid's frequency is precisely aligned with one of the DFT frequency bins. Figure 2.1 shows the magnitude spectrum of a DFT (length $N = 16$) of a sinusoid with period $\frac{N}{4}$. Note that the magnitude spectrum consists of a single spike at bin 4.

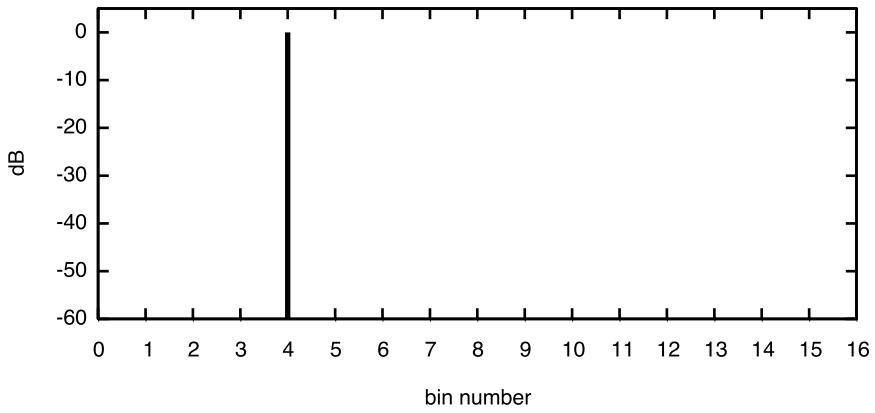


Figure 2.1: DFT magnitude spectrum of a sinusoid with period $\frac{N}{4}$, $(N = 32)$.

Since such precisely constructed signals are unlikely to be found in real-world digital recordings, let us now consider the case of a sinusoid with a period of $\frac{N}{4.21}$ samples, and its corresponding DFT magnitude spectrum (figure 2.2). In this case, there are significant components present in bins 0 through 16. Since the signal is N samples long, but has a sinusoidal period of 4.21 cycles, it cuts off abruptly creating a significant discontinuity.

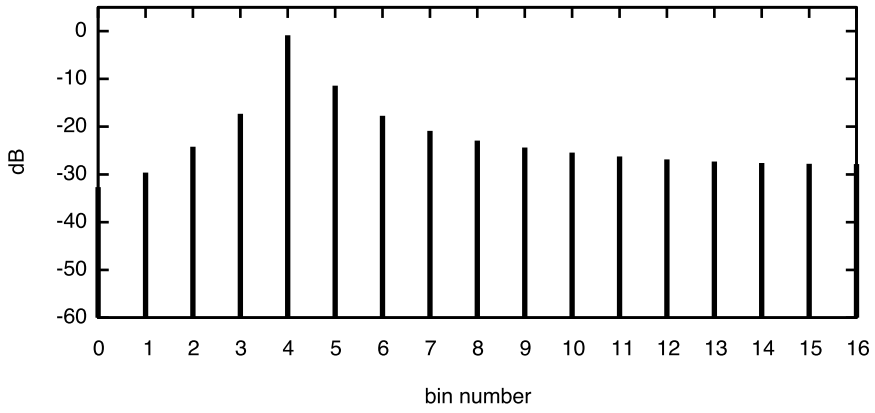


Figure 2.2: DFT magnitude spectrum of a sinusoid with period $\frac{N}{4.21}$, $(N = 32)$.

Figure 2.3 compares two sinusoids, one with period $\frac{N}{4}$ and the other with period $\frac{N}{4.21}$. Note that the sinusoid with period of $\frac{N}{4}$ arrives smoothly at 0 at sample index 32.

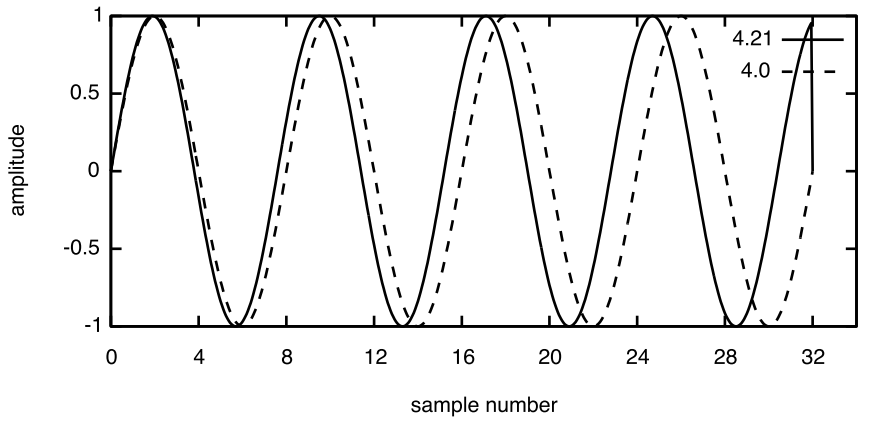


Figure 2.3: Sinusoids of 4.21 cycles and 4.0 cycles per N samples, $(N = 32)$.

Intuitively, one can interpret such discontinuities as additional high frequency components. These additional components are often described as *spectral distortion* or *spectral leakage*. These distortions are a significant issue when one is attempting to detect multiple sinusoids at different frequencies: it may be impossible to differentiate spectral leakage from sinusoids. By applying a smoothing *window function*, $h(m)$, in the time domain, these edge discontinuities can be reduced and the spectral leakage attenuated. The STFT of

length N beginning at sample n and with smoothing window $h(m)$ is defined as follows:

$$X_n(k) = \sum_{m=0}^{N-1} x(m+n)h(m)e^{-i2\pi\frac{k}{N}m} \tag{2.13}$$

A number of different window functions are commonly used in audio applications. The most effective are generally bell-shaped curves. Note that the apparent absence of a smooth window function in fact implies a rectangular window function. The rectangular window function is defined to be 1 on the interval $0 \ldots N-1$ and 0 otherwise. See Harris (1978) and Nuttall (1981) for a complete discussion of window functions, their definitions, and properties.

Before continuing further with our discussion of windowing, we must introduce one other important signal processing concept. The convolution operation $*$ for two signals $x(n)$ and $y(n)$ is defined as follows:

$$f(n) = x(n) * y(n) = \sum_{m=0}^{N-1} x(m)y(n-m) \tag{2.14}$$

The convolution of $x(n)$ and $y(n)$ can be thought of as a new signal, $f(n)$, where each sample of x is multiplied by a time shifted copy of all the samples in y and the results summed. Thus, $f(n)$ consists of the scaled shifted sums of copies of $y(n)$.

The convolution theorem (which we will not prove here) states that the spectrum of the convolution of two signals is equivalent to the multiplication of the signals each expressed in the frequency domain:

$$\begin{aligned} \text{DFT}\left[x(n) * y(n)\right] &= \text{DFT}\left[x(n)\right]\text{DFT}\left[y(n)\right] \\ &= X(n)Y(n) \end{aligned}$$

The converse is also true: the spectrum of the multiplication of two signals in the time domain is equivalent to the convolution of the signals each expressed in the frequency domain:

$$\begin{aligned} \text{DFT}\left[x(n)y(n)\right] &= \text{DFT}\left[x(n)\right] * \text{DFT}\left[y(n)\right] \\ &= X(n) * Y(n) \end{aligned}$$

By the convolution theorem, the multiplication of a signal $x(n)$ and a window $h(n)$ is equivalent to their convolution in the frequency domain: DFT $[x(n)h(n)] = X(n) * H(n)$. Since the spectrum of a sinusoid is a single impulse, it follows that the spectrum of a windowed sinusoid will be a scaled copy of the spectrum of the window function, centered on the frequency of the sinusoid. Returning to the example of a sinusoid with a period of 4.21 cycles per window, we can get a sense of the effects of windowing by inspecting the underlying continuous spectrum.[2]

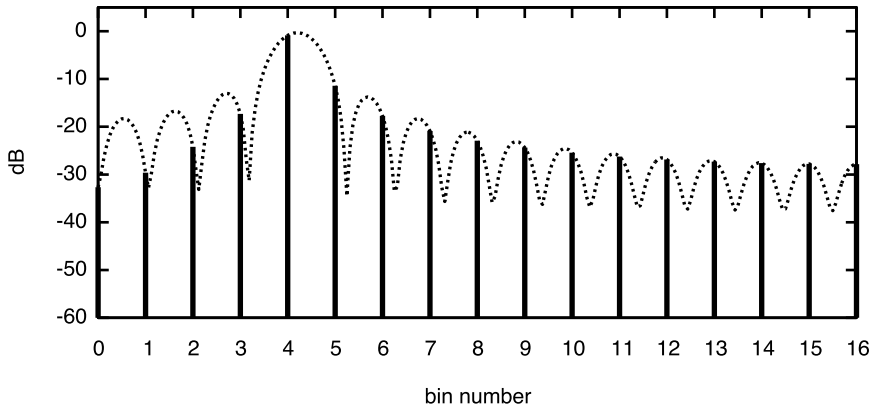


Figure 2.4: DFT magnitude spectrum and the interpolated underlying magnitude spectrum of a sinusoid with period $\frac{N}{4.21}$, $(N = 32)$.

The dotted curve in figure 2.4 shows the continuous underlying spectrum which consists of a *main lobe* centered at the sinusoid frequency and many *side lobes* of lower amplitude arranged around the main lobe. Different window functions exhibit unique main lobe shapes and side lobe levels. The case of a sinusoid with an integer period and a rectangular window is quite special. Figure 2.5 shows that the although there are significant side lobes in the underlying spectrum, the DFT samples align precisely with nulls in the continuous spectrum. The result is a single impulse in frequecy bin 4.

Examining the spectra of various window functions can provide a good sense of their practical capabilities. Figures 2.6 through 2.9 show various window functions in both the time domain and frequency domains. Note that as the center area of the window

[2]The continuous spectrum can be approximated by zero-padding the signal and then performing the DFT. The result is a spectrum that is oversampled by the zero-padding factor.

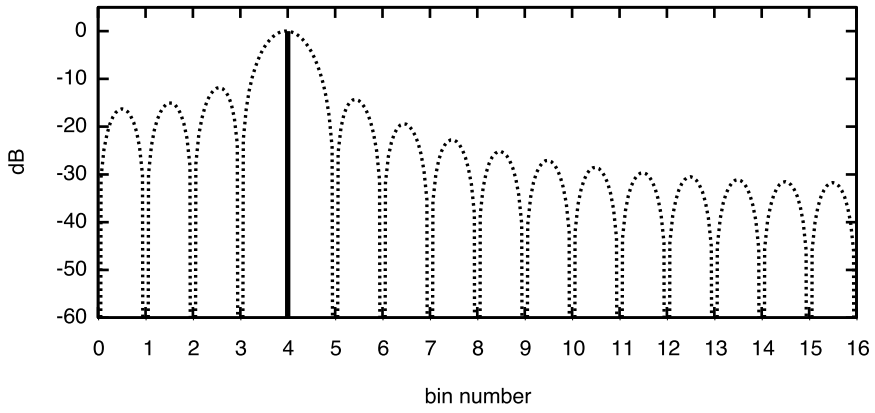


Figure 2.5: DFT magnitude spectrum and the interpolated underlying magnitude spectrum of a sinusoid with period $\frac{N}{4}$, $(N = 32)$.

function becomes narrower in the time domain, the main lobe of the window spectrum becomes wider. This demonstrates an important tradeoff in the design and use of window functions.

By using the STFT and appropriate windowing, we can successfully model many types of musical signals and more significantly, we can use these models to effect a number of different sonic transformations.

2.2 STFT Analysis and the Phase Vocoder

The phase vocoder is one of the most widely utilized STFT techniques for time and frequency scale modification. The technique makes use of phase information in successive analysis frames to make a precise estimate of the sinusoidal frequency present at a particular analysis bin. By default, the frequency resolution of the STFT is limited by the bin spacing, which in turn is controlled by the DFT length N. For example, as was observed in equation 2.9, a DFT of length 1024 at a sampling rate of 44.1 kHz results in a bin resolution of 43.07 Hz. Practical DFTs are computed by means of the fast Fourier transform which limits N to powers of 2. Therefore, there are some significant restrictions on the frequency resolution. Moreover, the center frequencies of each STFT bin will be limited to harmonics of the center frequency of bin 1.

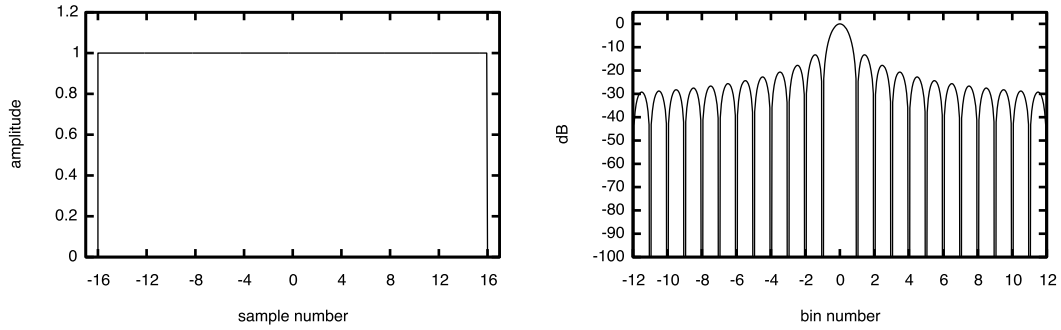


Figure 2.6: Time domain plot and magnitude spectrum of the rectangular window.

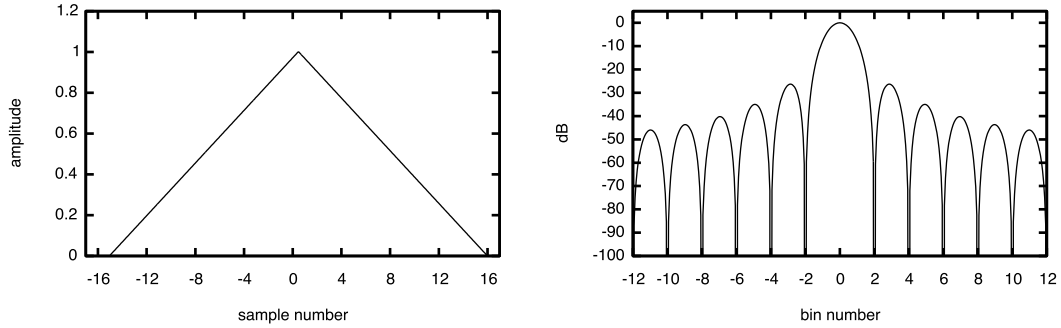


Figure 2.7: Time domain plot and magnitude spectrum of the triangular window ($N = 31$).

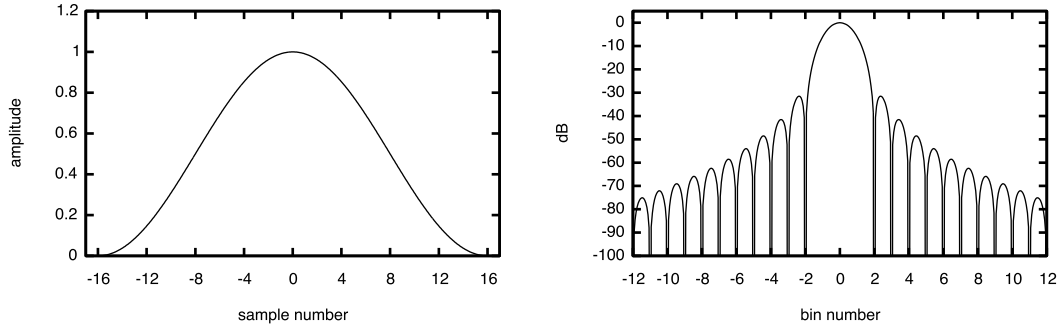


Figure 2.8: Time domain plot and magnitude spectrum of the Hann window.

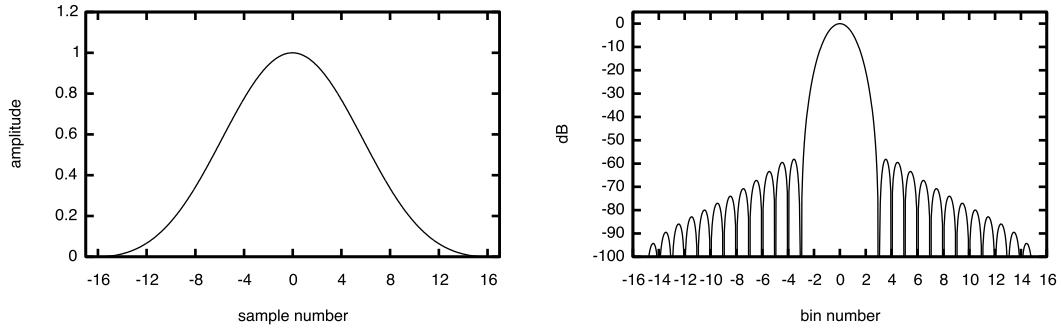


Figure 2.9: Time domain plot and magnitude spectrum of the Blackman window.

Purely harmonic sounds may be analyzed by resampling the signal in the time domain so that the length of the fundamental period is an integer factor of the DFT length N (Beauchamp 1993). This assures that each harmonic in the input will be aligned with the center of an analysis bin. However, this does not accommodate inharmonic sounds, nor does it allow the measurement of frequency deviations (including vibrato) that are present in almost all real-world acoustic signals.

2.2.1 Putting Phase to Good Use

Thus far we have been looking at the magnitude spectrum, but each analysis frame also has a phase spectrum and this information is useful. Since frequency is defined as the amount of phase change per time unit, the change in phase between successive analysis frames can be used directly to estimate frequency. We define the phase deviation for frequency bin k and analysis frame n as follows:

$$\Delta\phi_{k,n} = \phi_{k,n} - \left[\phi_{k,n-1} + H\frac{2\pi k\Delta f}{f_s}\right] \tag{2.15}$$

where $\phi_{k,n}$ is the phase for bin k, frame n, H is the analysis hop size in samples, Δf is the frequency bin spacing in hertz, and f_s is the sampling rate. Equation 2.15 expresses the idea of phase unwrapping, where the term $\phi_{k,n-1} + H\frac{2\pi k\Delta f}{f_s}$ represents the unwrapped phase of a sinusoid with frequency centered precisely on analysis bin k. Positive phase deviation $\Delta\phi_{k,n}$ indicates a sinusoid with a frequency higher than the bin center frequency, negative phase deviation indicates lower frequency.

In practice, the phase deviations must be re-wrapped to the half-open interval $[-\pi, \pi)$ which constrains the allowable frequency deviation and locates the estimated frequency around analysis bin center frequency. If $\Theta\phi_{k,n}$ represents the re-wrapped phase deviation, then the estimated frequency (in hertz) for bin k and analysis frame n is

$$f_{k,n} = \left[\frac{\Theta\phi_{k,n}}{2\pi}\right]\left[\frac{f_s}{H}\right] + k\Delta f \tag{2.16}$$

Note that decreasing the hop size increases the range of possible frequency deviation. Small hop sizes will allow the phase vocoder to more closely track the rapid phase deviations of non-stationary (noisy) spectral components.

For quasi-stationary (sinusoidal) components, we note that there is an important relationship between the choice of window function and hop size. As mentioned earlier, a single sinusoidal component will result in a spectrum that is a scaled translation of the spectrum of the analysis window. As such, we desire that all bins in the main lobe of the window should contribute to the energy of the sinusoid, i.e. their frequencies should "lock" to the frequency at the center of the main lobe. For a main lobe width of W bins and window size of M, a hop size H such that $H \leq \frac{M}{W}$ will be sufficient (Puckette 1995). Thus, the wider the main lobe, the smaller the allowable hop size. This also makes intuitive sense from a time domain perspective. Since wider main lobes result from narrower bell-like window shapes in the time domain, the decreased width of the window shape must be compensated for by more dense sampling in the time domain.

2.2.2 Resynthesis and Transformation

Following analysis, the phase vocoder data can be used to perform a resynthesis of the original sound. Depending on the desired result, the analysis data may be modified to effect different sonic transformations. Two different resynthesis methods are in common use.

The first is the STFT overlap-add technique. With this method, the inverse DFT is applied to the data in each analysis frame. Since the analysis hop size H_a is typically less than the DFT length N, a time domain overlapping of each DFT window results. Provided that the analysis window function sums to unity upon overlap (which can be assured by choosing the appropriate window function and normalization factor), the original signal can be recovered exactly. Setting the hop size so that $H_s \neq H_a$ gives a time warped resynthesis.

The second resynthesis method is the summing oscillator bank. Each bin of the phase vocoder data is associated with a sinusoidal oscillator and each phase vocoder frame provides frequency, amplitude, and phase information that can be used to drive the oscillators. Since each analysis frame is separated by a hop of H_a samples, linear interpolation of frequency and amplitude is often used to fill in intermediate samples. With linearly interpolated frequency, the phase function becomes the integral of instantaneous interpolated frequency. As such, the resynthesized phases cannot be guaranteed to match the original phases, and thus resynthesis with the oscillator bank will not recover the original signal. In practice, different frequency interpolation strategies can be used to achieve fairly high fidelity results. For example, the cubic phase interpolation method commonly used in sinusoidal modeling can be used to match phase at each frame (see section 3.3.4 for details).

Typically one is interested in transforming the analysis data in order to achieve one or more of the following results:

1. frequency filtering (boosting or cutting certain frequency bands)
2. time expansion or compression
3. transposition (frequency scaling)

Of these three, frequency filtering is the most straightforward, and can easily be implemented by selectively scaling the amplitudes in the desired analysis bins. Time varying filtering may be used in applications such as broadband noise reduction.

2.2.3 Time Expansion/Compression

Time expansion/compression may be achieved in the overlap add case by adjusting H_s and/or H_a. The expansion ratio is given by $\alpha = \frac{H_s}{H_a}$. When performing time expansion or compression, care must be taken to properly update the phases in each synthesis frame. Let $\phi'_k(n)$ denote the resynthesis phase for bin k in frame n. Phase is propagated forward

from each previous frame according to the analysis frequencies as follows:

$$\phi'_k(0) = \alpha\phi_k(0) \tag{2.17}$$

$$\phi'_k(n) = \phi'_k(n-1) + H_s\frac{f_k(n)2\pi}{f_s} \tag{2.18}$$

Note that the initial phase is multiplied by the expansion ratio α. In their 1997 study, Laroche and Dolson show that scaling the initial phases will preserve the original phase relationships when expanding by an integer factor.

Because time expansion/compression no longer guarantees the same phase relationships as the original sound, an additional smoothing synthesis window function is generally applied after each inverse DFT. This serves to reduce the effect of any phase or amplitude discontinuities that result from frequency domain transformations. Since the application of a synthesis window amplitude modulates the signal, proper care must be taken to assure enough overlap to reduce amplitude modulation artifacts.

To explore this issue in more detail, it will prove useful to examine the amplitude behavior of several analysis and synthesis windows under different overlap-add conditions. First let us define the overlap factor $\beta = \frac{N}{H_s}$. When $\beta = 1$ there is no overlap, and the output consists of a signal that is amplitude modulated by the product of the synthesis window with the analysis window. (The original analysis window was recovered from the inverse DFT although the shape may be distorted due to operations performed in the frequency domain.) Figure 2.10 shows the overlap-adding of Blackman analysis and synthesis windows with β values of 1, 3, 4, and 6. Note that un-modulated gain is achieved when $\beta = 6$. In general, windows with a narrower time domain peak will require more overlap.

Additional steps must be taken to assure proper amplitude normalization. For a synthesis window of w' of length N, the normalization factor γ is

$$\gamma = H_s\sum_{i=0}^{N-1} w'(i) \tag{2.19}$$

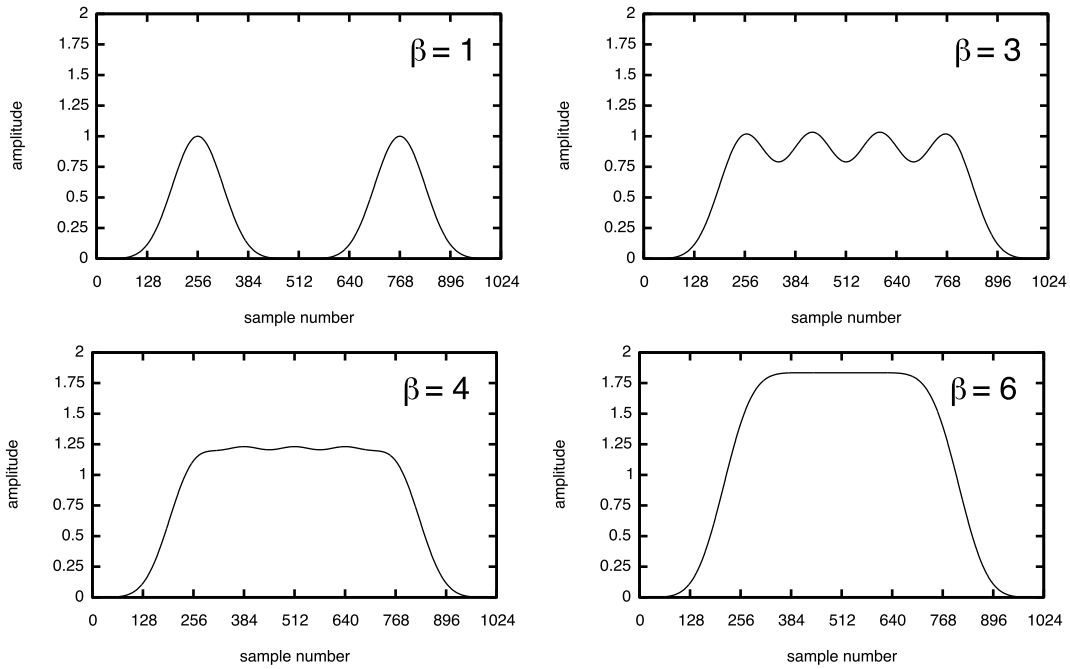


Figure 2.10: Overlap-adding the product of Blackman analysis and synthesis windows with varying overlap factors (β).

Time varying expansion and contraction factors imply a variable synthesis hop size with a corresponding variable normalization factor. Due to the possibility of amplitude modulation artifacts, care must be taken not to increase H_s beyond the point of constant gain. In practice it is best to fix H_s and vary H_a (or linearly interpolate frequency and amplitude from the analysis spectrum) to achieve the desired expansion ratio.

2.2.4 Transposition

Transposition (frequency scaling) with the phase vocoder may be handled in a number of different ways. A simple, though computationally expensive, approach synthesizes each phase vocoder band with a sinusoidal oscillator. Frequency and amplitude may be linearly interpolated from frame to frame. If $\hat{F}_k(n)$ and $\hat{A}_k(n)$ are the piecewise interpolated frequency and amplitude functions for bin k, then the resynthesis is

$$x(n) = \sum_{k=0}^{N/2-1} \hat{A}_k(n) \cos\left(nP\hat{F}_k(n)\frac{2\pi}{f_s} + \phi_k\right) \tag{2.20}$$

where P is the transposition ratio, and ϕ_k is the initial phase of bin k.

A less computationally expensive approach is to resynthesize with the overlap-add method with a time expansion/compression ratio that is the inverse of the transposition ratio. The transposition is then achieved via resampling in the time domain. Another approach is to scale the spectrum in the frequency domain as described in Laroche and Dolson (1999). The spectral magnitude peaks are scaled according to the transposition ratio, and the phases are adjusted to reflect the new transposed frequencies.

2.2.5 Refinements

A number of additional refinements can be added to the basic phase vocoder technique. One drawback that has been observed with time expansion/compression is the problem of "phasiness." This can best be described as a loss of presence in the synthesized signal; the result is less crisp and there is sense of greater reverberation. Much of this can be attributed to a lack of phase coherence between adjacent frequency bins. Although the standard phase vocoder technique interpolates coherent phase between adjacent analysis frames ("horizontal" phase), it does nothing to maintain "vertical" phase alignment. Puckette (1995) proposed a method of phase locking that proceeds by first finding amplitude peaks in each frame. Under the assumption that frequency bins adjacent to the peak contribute to the same sinusoid, the phases of adjacent bins are set equal to the phase of the peak bin. Laroche and Dolson (1997) describe a number of different phase maintenance strategies (all based on this peak detection method) that can provide significant improvements in sound quality. Phase relationships are particularly important for preserving transient and micro-transient events. Robel (2003) describes a technique for reinitializing phases at transients that significantly reduces phasiness and transient smearing under time scale modifications. Although the phase vocoder has its limitations, when carefully implemented it can offer excellent fidelity.

2.3 Spectral Envelopes

Up to this point we have been discussing the precise frequency domain content of audio signals. It has been shown that the ear functions similarly to Fourier analysis, with specific locations along the basilar membrane of the inner-ear corresponding to specific frequencies. However, in addition to frequency content, we are also simultaneously sensitive to the overall general shape of the spectrum. We call the curve that defines the shape of the magnitude spectrum the *spectral envelope*. The dotted curve in figure 2.11 shows one possible spectral envelope for the given spectrum.

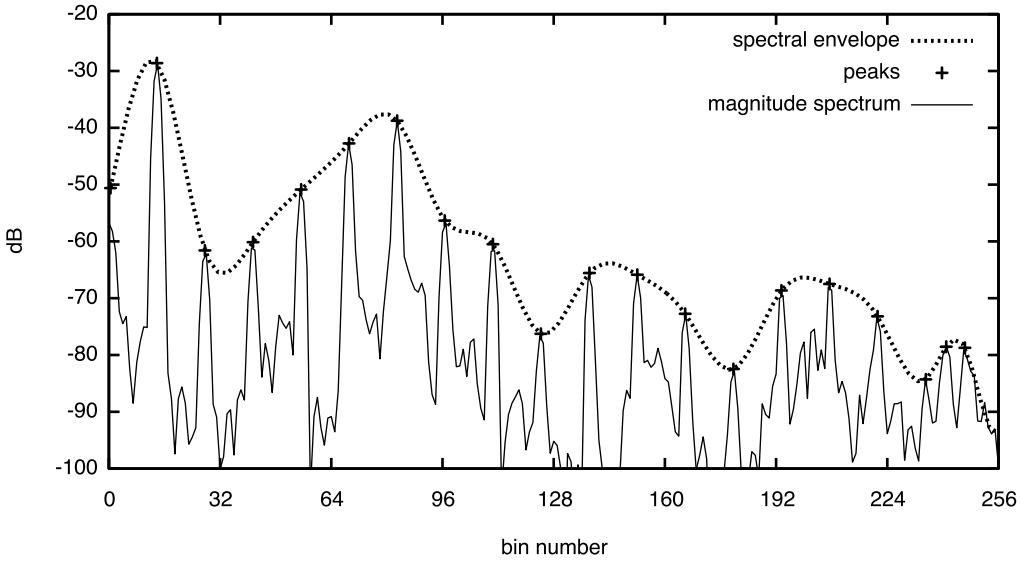


Figure 2.11: Spectral envelope of a soprano voice.

Many vocal and instrumental sounds can be understood in terms of a *source-filter* model, where the resonant body of the instrument or of the vocal tract shape (filter) the input (the source). In voiced human speech, the source is provided by the periodic vibration of the vocal folds. The spectral envelope can be taken as a model of the resonances of the vocal tract. For vowels, the peaks in the spectral envelope trace the *formants*. For any particular speaker and vowel, the spectral envelope stays fixed regardless of pitch.

If a sound is transposed using the phase vocoder, for example, the spectral envelope shape will change along with the transposed spectrum. For an instrumental or vocal

sound, the perceived result is a change in *size* of the resonating body. Thus there is a change in timbre as well as in pitch. The timbre can be preserved by dividing out the spectral envelope prior to transposition (this process of flattening the spectrum is sometimes referred to as *whitening*), and then filtering the transposed magnitude spectrum by the original spectral envelope.

Common methods for determining the spectral envelope include linear predictive coding (LPC), cepstrum, discrete cepstrum, and true envelope methods (Robel and Rodet 2005). For a thorough discussion of spectral envelope estimation methods and applications see Schwarz (1998).

2.4 Sinusoidal Modeling

Like the phase vocoder, sinusoidal modeling is an STFT based analysis/resynthesis technique. It attempts to represent an audio signal as a generalized collection of sinusoids with time varying frequency, amplitude, and phase. Unlike the phase vocoder, the number of sinusoids present at a given time is not fixed by the DFT size. As a consequence, the sinusoids need not necessarily be centered around a fixed frequency band, and sinusoids may vary arbitrarily in frequency. The following plots of a frequency sweep (chirp) first analyzed with the phase vocoder (figure 2.12) and then with sinusoidal modeling (figure 2.13), clearly illustrate the difference between the two approaches.

The phase vocoder represents the chirp with many frequency bands that each lock to the chirp frequency as it sweeps upward. The result is highly overspecified, with many bands per sinusoidal feature. Manipulation of the chirp (transposition for example) requires the adjustment of many frequency bands. As a corollary, a local adjustment to any one frequency band will degrade the integrity of the frequency sweep.

In contrast, a sinusoidal model represents the chirp as a single breakpoint function with time varying frequency. The chirp can be freely transposed and reshaped by the manipulation of a single frequency/amplitude trajectory. Although in most cases perfect

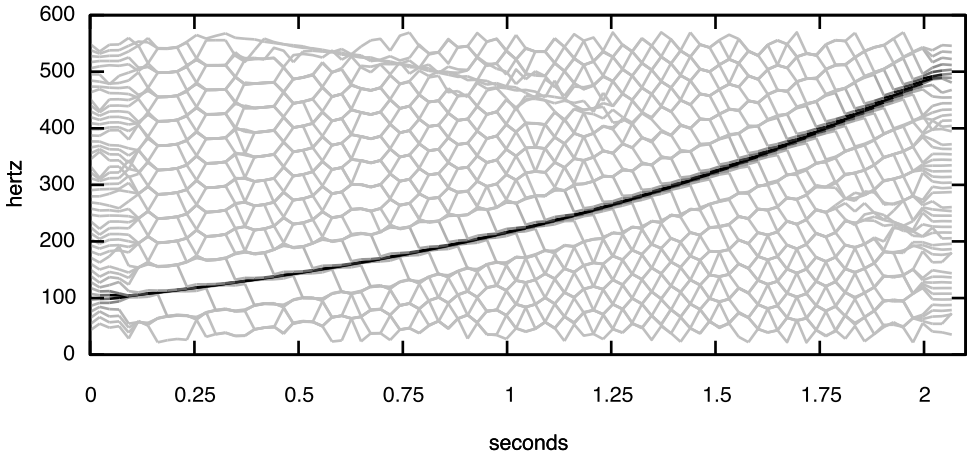


Figure 2.12: Phase vocoder analysis of a frequency sweep from 100 Hz to 500 Hz.

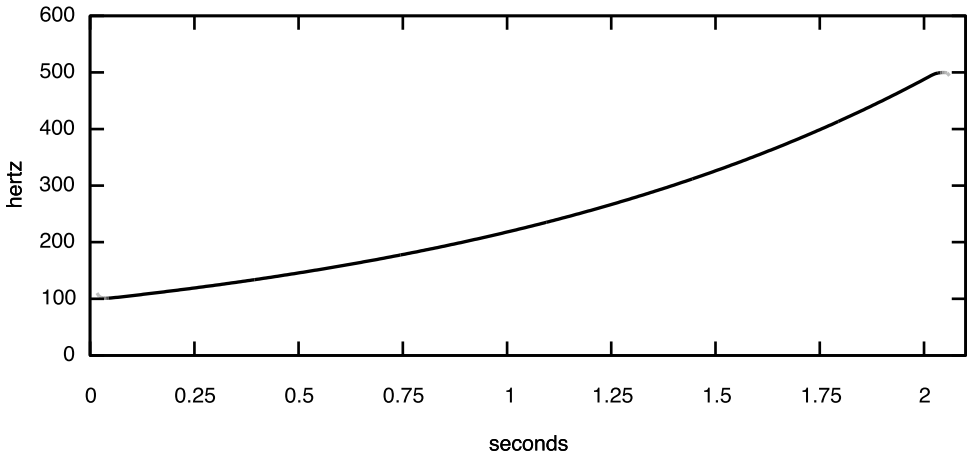


Figure 2.13: Sinusoidal modeling analysis of a frequency sweep from 100 Hz to 500 Hz.

reconstruction is not possible with a purely sinusoidal model, the benefits of the technique may outweigh this drawback.

2.4.1 STFT Frequency Interpolation

The basic method of sinusoidal modeling begins with a STFT. As observed in section 2.1.4, the short-time spectrum of a signal is the convolution of the spectrum of the window function with the spectrum of signal $x(n)$. If the signal in question is in fact a single sinusoid at frequency f_k, then its magnitude spectrum is a single impulse peak at frequency f_k, and its windowed magnitude spectrum is simply the spectrum of the window function

with its main-lobe centered at frequency f_k. In the DFT magnitude spectrum, a sinusoid appears as a local maximum near the true maximum of the underlying continuous spectrum. Parabolic interpolation may then be used to approximate the position of the peak of the main lobe.

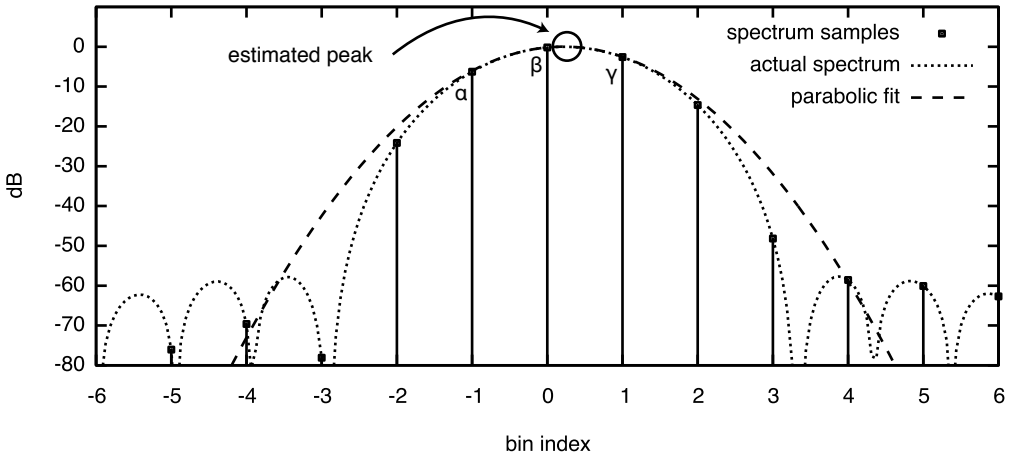


Figure 2.14: Estimating a peak in the spectrum based on a parabolic curve fitting.

Given the DFT $X(k)$, the magnitude spectrum can be searched for bins that are local maxima. Bin n is considered a local maximum if its magnitude is greater than its two neighboring bins, or more precisely $|X(k_{n-1})| < |X(k_n)| > |X(k_{n+1})|$. We can then fit a parabola that passes through the magnitude values of bins $n-1$, n, and $n+1$. The parabola takes the following form:

$$y(x) = a(x-p)^2 + b$$

The known points on the curve are $y(-1)$, $y(0)$, and $y(1)$ and are given by the the bin magnitudes (measured in dB). According to Smith and Serra (1987), frequency estimates are found to be more accurate when interpolating using dB rather than linear magnitude.

$$y(-1) = \alpha = 20\log_{10}|X(k_{n-1})|$$
$$y(0) = \beta = 20\log_{10}|X(k_n)|$$
$$y(1) = \gamma = 20\log_{10}|X(k_{n+1})|$$

With three equations and three unknowns (a, p, and b) we can solve for the parabola peak location p.

$$p = \frac{1}{2}\left(\frac{\alpha - \gamma}{\alpha - 2\beta + \gamma}\right) \tag{2.21}$$

The center frequency (measured in bins) will be $n + p$. The parabola peak location p can then be used to solve $y(p)$ for both the real and imaginary parts of the complex spectrum. For the real part, we set $\alpha_{\Re}$, $\beta_{\Re}$, and $\gamma_{\Re}$ to the real values at bins $n-1$, n, and $n+1$,

$$\alpha_{\Re} = \Re e[X(k_{n-1})]$$
$$\beta_{\Re} = \Re e[X(k_{n})]$$
$$\gamma_{\Re} = \Re e[X(k_{n+1})]$$

and for the imaginary part

$$\alpha_{\Im} = \Im m[X(k_{n-1})]$$
$$\beta_{\Im} = \Im m[X(k_{n})]$$
$$\gamma_{\Im} = \Im m[X(k_{n+1})]$$

Solving $y(p)$ for the real part a and imaginary part b gives

$$a = y(p)_{\Re} = \beta_{\Re} - \frac{1}{4}p(\alpha_{\Re} - \gamma_{\Re})$$
$$b = y(p)_{\Im} = \beta_{\Im} - \frac{1}{4}p(\alpha_{\Im} - \gamma_{\Im})$$

According to equations 2.11 and 2.12 the magnitude wil be $\sqrt{a^2 + b^2}$ and the phase will be $\tan^{-1}\frac{b}{a}$.

This frequency, phase, and magnitude interpolation method means that the analysis procedure can accurately detect sinusoidal frequencies that are non-integer multiples of the analysis bin spacing frequency. This makes it ideal for analyzing sounds that have inharmonic components, have variable frequency content, or exhibit limited non-stationary behavior (a sum of different harmonic complex tones, sounds with wide vibrato or glissandi, etc.)

2.4.2 Partial Tracking

Once all of the sinusoidal peaks in a particular STFT frame have been detected, the peaks must be organized such that they form continuous, time-varying sinusoidal tracks (partials). This is accomplished by matching the peaks in the current frame with peaks present in previous STFT frames. During resynthesis the frequency, amplitude, and phase can be smoothly interpolated from a peak to its matched successor in the next frame.

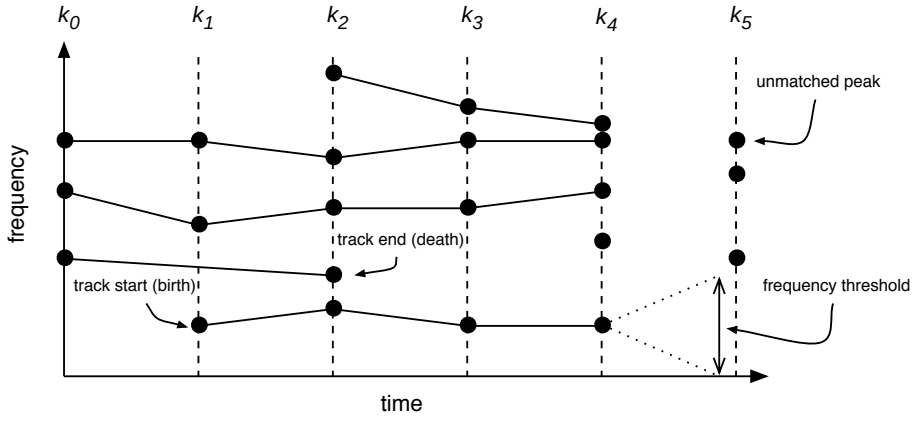


Figure 2.15: Connecting STFT peaks into sinusoidal tracks.

Typically, some peaks that fall below a certain amplitude threshold will be discarded prior to matching. This means that each frame may contain different numbers of peaks and that, over time, individual partials may vanish or appear depending on their amplitude. Amplitude thresholding techniques and tradeoffs will be discussed in more detail in the next chapter. The major implication for this type of sinusoidal model is that the number of sinusoids present at any one instant is variable. By eliminating redundancies and low amplitude components, the result is a model that is efficient to store and manipulate.

Several different techniques have been proposed for partial tracking. The simplest method is a locally optimal greedy algorithm that matches each peak to the peak in the next frame that is closest in frequency. Typically there is a distance threshold that constrains the maximum allowable frequency jump. Each peak is defined in terms of the following data structure:

structure Peak
    Freq
    Amp
    Phase
    Forwardmatch
    Backmatch
end structure

Note that peaks keep track of both their successor (Forwardmatch) and predecessor (Backmatch). Forwardmatch and Backmatch are initialized to Nil, indicating the absence of a matching peak. Listing 2.1 illustrates the greedy peak matching algorithm.

```
1:  for prevpeak ∈ P_{k−1} do
2:      for curpeak ∈ P_k do
3:          distance ← |prevpeak.Freq − curpeak.Freq|
4:          if distance < freq_threshold then
5:              if curpeak.Backmatch ≠ Nil then
6:                  existing_distance ← |curpeak.Backmatch.Freq − curpeak.Freq|
7:              else
8:                  existing_distance ← freq_threshold
9:              end if
10:             if distance < existing_distance then
11:                 curpeak.Backmatch.Forwardmatch ← Nil
12:                 curpeak.Backmatch ← prevpeak
13:                 prevpeak.Forwardmatch ← curpeak
14:             end if
15:         end if
16:     end for
17: end for
```

Algorithm 2.1: Greedy peak matching from frame $k-1$ to frame k.

P_k is the collection of peaks in frame k (the current frame) and P_{k-1} is the collection of peaks in the previous frame. The outer loop iterates through each of the peaks in frame $k-1$. The inner loop looks for the best match in the current frame by iterating through each of frame k's peaks. If the frequency distance between any two peaks is less than *distance_threshold* (line 4), then it is a candidate for matching. Before connecting the peaks, the algorithm checks to see if the candidate peak in frame $k+1$ has already been claimed (line 5). If so, the new distance must be less than distance of the current match (line 10).

Following matching, the Forwardmatch and Backmatch fields indicate the complete sinusoidal tracks. If Backmatch = Nil, a new partial is starting up. If Forwardmatch = Nil, a partial is ending. When a partial begins or ends, a short amplitude fade is usually added in order to avoid a discontinuity.

Given that the greedy algorithm has no explicit model of the underlying sound structure, nor any context beyond the peaks of the current and previous frames, it actually performs reasonably well in practice. However, the greedy algorithm can be easily confused by spurious peaks or significant frequency deviations such as vibrato or glissandi (see figure 2.16). A number of minor refinements can be made, such as incorporating amplitude into the measure of peak proximity or measuring frequency distance on a logarithmic scale.

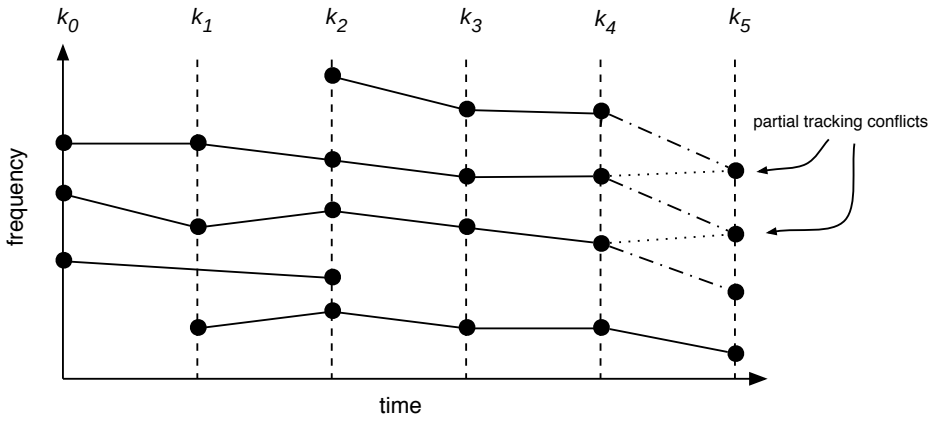


Figure 2.16: Partial tracking conflicts due to glissandi.

2.4.3 Improved Partial Tracking

A number of other improved partial tracking strategies have been proposed. Under the assumption that the sound being analyzed contains relatively stable frequency trajectories (such as stable harmonic or inharmonic partials of a single fundamental), peaks can be matched to a set of *frequency guides* (Serra 1989). Guide based partial tracking proceeds as follows: at frame k we have a set of N peaks P_k with frequencies $f_0, f_1, f_2 \ldots f_{N-1}$, and a set of M frequency guides G with frequencies $f'_0, f'_1, f'_2 \ldots f'_{M-1}$. As in the greedy

algorithm, each guide attempts to claim the peak in P_k that is closest in frequency to the guide frequency ($|f'_m - f_n|$ is minimized for all combinations, subject to constraints on the maximum allowable frequency jump). When a match is found between partial n and guide m, the partial trajectory is continued and a new guide frequency $\hat{f}_m$ is computed according to

$$\hat{f}_m = \alpha(f_n - f'_m) + f'_m \quad \text{for} \quad \alpha \in [0,1]$$

The parameter α is a simple lowpass filter on the guide frequency. When α is zero, the guide frequencies remain constant. As α approaches 1, the guide frequencies more closely track the frequency of the evolving partial. Guides that fail to find a match are set to a dormant state. They can "wake up" in future frames if they find a peak appropriately proximate in frequency. If a guide remains dormant for too long, it is removed from G. Peaks that are not matched to a guide may (optionally) spawn new guides. The number of maximum allowable guides may be limited, if desired.

For harmonic sound, the guide frequencies may be fixed to multiples of the fundamental frequency f_0. In this case the number of guides can remain constant throughout the evolution of the sound. Note that the tracking method works well only if the fundamental frequency estimation is reliable. Guide based tracking has two distinct advantages: it can fill temporal gaps in the frequency trajectory, and it can provide a clean representation for harmonic sounds (with one partial per harmonic). Guide based tracking has been implemented in a number of sinusoidal modeling packages including SMS, Loris, and ATS.

For sounds with non-stationary behavior (noise, microtransients, rapid frequency fluctuations, etc.) guide based tracking offers few advantages in comparison to the basic greedy algorithm. Depalle, García, and Rodet proposed a tracking method based on a combinatorial Hidden Markov Model (HMM) that provides more globally optimal frequency trajectories (Depalle et al. 1993). A Hidden Markov Model consists of a set of related states, observations, and parameters. In a typical HMM problem, the goal is to determine a temporal sequence of states based solely on observations that have some

probabilistic connection to the states. The states are the "hidden" part of the model, and the parameters determine the probability of a state (or state transition) given certain observations.

In the application of HMMs to partial tracking, a state consists of a set of connections from frame $k-1$ to k. The goal is to find the most likely sequence of states (connections) given the observations. In this case the observation is simply the number of peaks present in frame $k-1$ and in frame k. If we let S_k model the set of peak connections from frame $k-1$ to k then next state, S_{k+1} will specify the connections from frame k to $k+1$. Thus, a state transition involves the peak trajectories of three frames: $k-1$, k, and $k+1$. The parameters of the HMM specify the probability of a transition from state to state. The parameters are based on a cost function that favors peak connections with continuous frequency slopes. Note that actual frequency distance is not considered. For this reason the HMM method can be used to track crossing partials. Given the parameters of the model (the probability of a particular state transition), the Viterbi algorithm is used to find the most likely sequence of states. The optimal sequence of transitions will be that which minimizes frequency discontinuities.

The computational cost of the HMM method is high since it must consider all combinations of peak connections across all frames. In practice, the number of frames (and therefore the number of state transitions) is limited to some window of T frames. The number of possible peak connections under consideration is further constrained by fixing the number of partials alive during any interval of T. Other options could further constrain the combinations, such as limiting maximum allowable frequency slopes or limiting the number of partial crossings.

A particular weakness of the HMM method described by Depalle et al. is that it cannot fill temporal gaps in the partial trajectories. Gaps may occur (especially in low amplitude partials) due to bin contamination from noise, or if one is analyzing material that has undergone lossy compression (a situation that may be common given current audio distribution methods). Two new promising partial tracking approaches include *future trajectory exploration* (Lagrange et al. 2004) and tracking using *linear prediction* (Lagrange

et al. 2003). SPEAR has implemented the linear prediction method which will be detailed in section 3.2.

2.4.4 Synthesis

The standard resynthesis method for a sinusoidal model is the summing oscillator bank. Each frequency trajectory computed via partial tracking is represented as a breakpoint envelope where where point k of envelope i is represented by the time, frequency, amplitude, and phase values $(t_i^k, f_i^k, a_i^k, \phi_i^k)$. Linear interpolation of amplitude and frequency can be used to resynthesize each breakpoint segment. $F_i^k(n)$ and $A_i^k(n)$ are the linear interpolation frequency and amplitude functions for the segment from breakpoint i to breakpoint $i+1$.

$$F_i^k(n) = f_i^k + \left(f_{i+1}^k - f_i^k\right)\left(\frac{\frac{n}{f_s} - t_i^k}{t_{i+1}^k - t_i^k}\right) \quad \text{for } f_s t_i^k \leq n < f_s t_{i+1}^k \tag{2.22}$$

$$A_i^k(n) = a_i^k + \left(a_{i+1}^k - a_i^k\right)\left(\frac{\frac{n}{f_s} - t_i^k}{t_{i+1}^k - t_i^k}\right) \quad \text{for } f_s t_i^k \leq n < f_s t_{i+1}^k \tag{2.23}$$

where n is the output sample index, f_s is the sampling rate, and t_i^k is breakpoint time in seconds. If we let $\hat{F}_i(n)$ and $\hat{A}_i(n)$ be the piecewise amplitude and frequency functions for partial i defined for all n, then the final resynthesis is given by

$$x(n) = \sum_{i=0}^{N-1} \hat{A}_i(n) \cos\left(n\hat{F}_i(n)\frac{2\pi}{f_s} + \phi_i\right) \tag{2.24}$$

where ϕ_i is the initial phase of partial i.

Note that the instantaneous phase is the integral of the frequency function. For linear frequency interpolation this implies a quadratic phase function. The phase values at all but the first breakpoint are ignored. If we desire a resynthesis that is phase accurate, a different interpolation strategy must be used. McAulay and Quatieri (1986) developed a cubic phase interpolation method that matches frequency and phase at each breakpoint. This will be discussed in detail in section 3.3.4.

2.4.5 Limitations and Extensions

Sinusoidal modeling makes the assumption that the audio signal can be closely approximated as a sum of sinusoids. Although this may hold true for certain classes of signals (harmonic sounds with clear tonal components and slowly varying frequency content) many types of signals are not well represented by sinusoids — broadband noise or transient events such a snare drum hit, for example. In many cases, sounds of interest will exhibit both tonal and noisy components. For example, a flute tone will contain quite a bit of broadband breath noise in addition to a clear fundamental.

Partial tracking becomes particularly difficult in the presence of noise, reverberation, or dense polyphony. Recent research continues to suggest new approaches for partial tracking, noise modeling, and transient preservation. A few of these methods have been implemented in SPEAR and will be discussed in the next chapter.

Even with extensions to the basic sinusoidal modeling method, perfect signal reconstruction is not guaranteed. However, when considering compositional goals, perfect reconstruction is rarely the intent. If lossless reproduction is required, sampling synthesis accomplishes this with far greater ease and efficiency. The power of a sinusoidal model lies in its potential to allow independent transformations in both time and frequency. One can begin with models that closely mimic real-world acoustic events and then, through a series of transformations or interpolations, explore a vast space of synthetic variations. Clearly this is fertile territory for creative musical exploration.

3. Sinusoidal Analysis and Resynthesis

SPEAR provides two different spectral analysis modes — the phase vocoder and sinusoidal modeling. The phase vocoder follows the standard method described in the preceding chapter. This chapter will focus on specific implementation of sinusoidal analysis/resynthesis. Several extensions to the basic techniques, as described in the preceding chapter, will be discussed.

3.1 Analysis Parameters

Sinusoidal analysis requires a number of input parameters including window type, window size, DFT size, analysis hop size, amplitude thresholds, and partial tracking constraints. All of these parameters can have a significant effect on the quality of the analysis. The interaction of amplitude thresholds and partial tracking can be particularly complex. The goal is to provide defaults for all of these parameters that work reasonably well for a wide variety of musical signals. However, even with careful choices, some user-selected parameter inputs are almost always required. Typically, the user might begin analysis using the default settings and then adjust accordingly after viewing the analysis and auditioning the resynthesis.

3.1.1 Analysis Window

One of the most crucial parameter choices is the length of the analysis window. This will control width of the main lobe in the frequency domain and, as a result, will determine the frequency resolution of the analysis. Rather than specifying the window length in samples, the user inputs the frequency resolution f_a in hertz. Adjacent sinusoids that are at least f_a

hertz apart can be properly resolved in the analysis. Additionally, no significant sinusoid with a frequency less than f_a hertz should be present in the input. For a harmonic sound, f_a should correspond to the frequency of the fundamental.

Given f_a, we must determine the appropriate window size and DFT size. In order to resolve sinusoids using the peak picking and quadratic interpolation method, the window size must be large enough to insure separation of the window main lobes. The window length M is given by

$$M = \Delta s \left(\frac{f_s}{f_a} \right) \tag{3.1}$$

where f_s is the sampling rate and Δs is the desired window separation in bins.

SPEAR uses a Blackman window which has a main lobe width of 6 bins. If we require complete separation of the main lobes as shown in figure 3.1, then $\Delta s = 6$. Note that as Δs increases, the window length grows. Since large windows introduce an undesirable loss of temporal resolution, we would like to use a smaller value of Δs that still allows for the resolution of closely spaced sinusoids. Abe and Smith (2004) made an extensive study of the minimum allowable frequency separation (MAFS) for various window types. The MAFS is the smallest separation that allows for peak detection and does not introduce significant bias in the peak interpolation due to interference from a neighboring peak. With a spectral oversampling (zero padding) factor of 2, the MAFS for the Blackman window is 3.53 bins (Abe and Smith 2004, 8, table 9). This is shown in figure 3.2. SPEAR uses the slightly more conservative separation of 4 bins. Given a window length M, the FFT size is $N = 2^{\lceil \lg M \rceil + 1}$, which results in spectrum oversampling by a minimum factor of 2.

With a peak separation of $4f_a$ and spectrum oversampling by a minimum factor of 2, each main lobe peak will be sampled by at least 7 DFT bins. To aid the rejection of spurious peaks we may conservatively impose the restriction that the three magnitudes $|X(k_{n-1})|, |X(k_n)|, |X(k_{n+1})|$ of a parabolic peak must also all exceed the magnitude of either neighboring bin: $|X(k_{n-1})| > |X(k_{n-2})|$ or $|X(k_{n+1})| > |X(k_{n+2})|$.

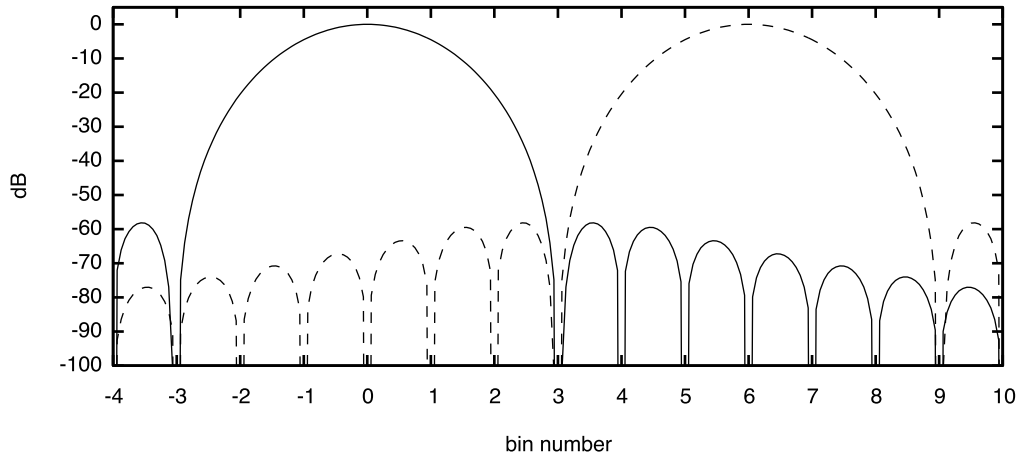


Figure 3.1: Magnitude spectrum of the main lobes of two Blackman windows with a frequency separation $\Delta s = 6$.

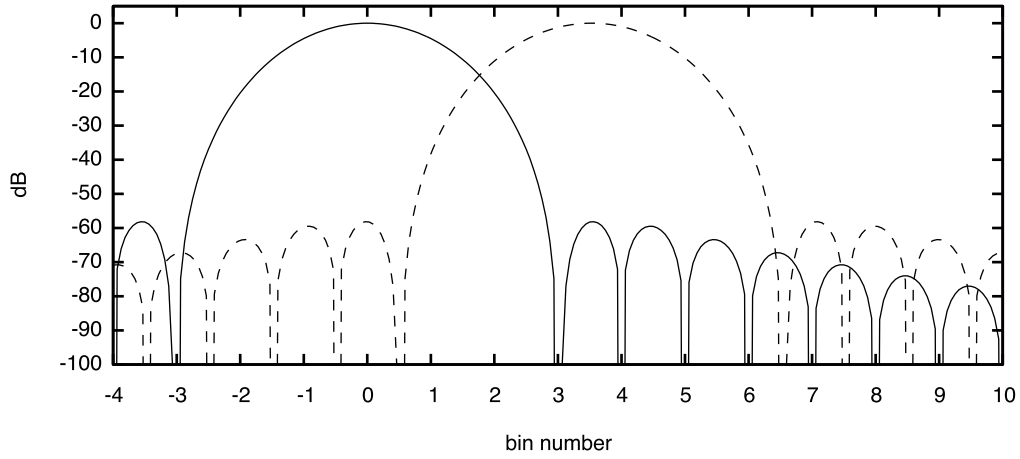


Figure 3.2: Magnitude spectrum of the main lobes of two Blackman windows with the minimum allowable frequency separation $\Delta s = 3.53$.

3.1.2 Amplitude Thresholds

Amplitude thresholds are used to limit the number of peaks in each analysis frame. The desire is to detect and track only the most perceptually significant partials. Because many sounds of interest exhibit a high frequency roll-off, it is helpful to equalize the spectrum. This results in an analysis that gives equal weight to both high and low frequency components and avoids resynthesis that sounds dull and heavily lowpass filtered.

Equalization can be achieved by applying a pre-emphasis high-pass filter in the time domain (Maher 1989), or by applying an emphasis curve to the STFT data (Serra 1989). One particular disadvantage of the applying a filter in the time domain is an alteration of the magnitude and phase spectra, the effects of which can only be easily undone by applying a complementary de-emphasis filter following resynthesis.

SPEAR achieves the effect of pre-emphasis by using a frequency dependent amplitude threshold curve. SPEAR establishes two amplitude thresholds: T_b, which is a variable threshold that must be exceeded to begin a new partial track, and T_d, which is fixed lowest threshold (default value is -90 dB). If peaks are found that exceed T_d, then they may be used to continue an existing partial track. If no appropriate track continuation is found, the partial ends — so T_d may be considered a "death" threshold for possible continuation of a partial.

T_b is computed using a frequency dependent threshold curve, $A_t(f_p)$, in combination with the maximum bin amplitude of the analysis frame, $a_n^{\max}$. Incorporating the maximum bin amplitude allows T_b to track the overall signal level. The threshold curve is given by

$$A_t(f_p) = a_T + a_L + \left(\frac{a_R}{b-1}\right) - \left(\frac{a_R}{b-1}\right) b^{f_p/20000} \tag{3.2}$$

where f_p is the frequency in hertz, b is a parameter controlling the shape of the curve, a_T is user controllable threshold in dB, a_L is the amplitude offset at 0 kHz in dB, and a_R is the range of the curve in positive dB from 0 to 20 kHz. Figure 3.3 shows $A_t(f_k)$ with the default values $b = 0.0075$, $a_T = -60$, $a_L = 26$, and $a_R = 32$.

Given the maximum bin amplitude in dB for a frame, $a_n^{\max}$ and the frequency f_p in Hz of the peak under consideration, the threshold T_b is given by

$$T_b = a_n^{\max} + A_t(f_p) \tag{3.3}$$

T_b constitutes a "birth" threshold for potential new partials. The birth threshold requires low frequency peaks to exceed a greater threshold than high frequency ones. In addition to helping find more high frequency partial tracks, this aids in the rejection of spurious low frequency, low amplitude partial tracks. Although the high frequency emphasis afforded

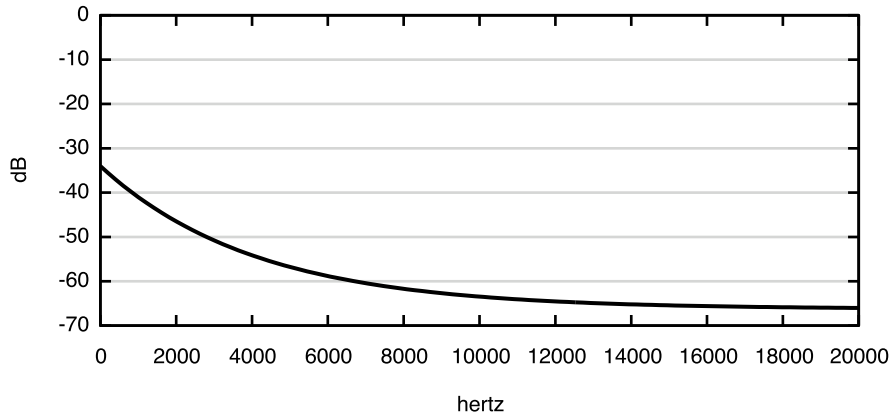


Figure 3.3: Frequency dependent threshold curve with the default values $b = 0.0075$, $a_T = -60$, $a_L = 26$, and $a_R = 32$.

by the variable threshold provides increased resynthesis fidelity for many real world sounds, it can result in many low amplitude, high frequency partials. In many cases, the signal energy above 5 kHz could be more efficiently and robustly modeled as noise bands, rather than clusters of low amplitude sinusoids. Possible approaches to modeling noise will be discussed in section 3.6.

3.2 Partial Tracking Using Linear Prediction

Peaks that are retained following thresholding are joined into breakpoint functions representing individual sinusoidal partials. The linear prediction (LP) method of Lagrange et al. (2003) is used to connect the peaks into tracks. Linear prediction treats each evolving sinusoidal track as a signal. The current signal sample, $x(n)$, is approximated as a linear combination of past samples:

$$\hat{x}(n) = \sum_{k=1}^{K} a(k)x(n-k) \tag{3.4}$$

There are K linear prediction coefficients, $a(k)$, that are calculated by minimizing the error between the predicted value, $\hat{x}(n)$, and the actual value $x(n)$. The K coefficients can then be used to predict successive values $\hat{x}(n+1), \hat{x}(n+2)$, etc. Several different algorithms,

including the Burg method, autocorrelation, and covariance can be used to compute $a(k)$. Lagrange et al. show that the Burg method is most suitable in this application.

LP can be used to compute possible future values for both frequency and amplitude. The frequencies and amplitudes of newly detected peaks are compared to the predicted values, and the closest matches are used to extend the sinusoidal tracks. The method is implemented as follows: at frame k we have detected N candidate peaks and we have M active sinusoidal tracks which extend to at most frame $k-1$. For each of the M active sinusoidal tracks, the Burg method is used to compute two sets of linear prediction coefficients, one on frequency and one on amplitude. An LP model of order 6 computed from a maximum of 64 previous values has worked well in practice. It is critical that enough points are used to capture periodic features such as amplitude modulation or vibrato. For a more thorough discussion of the choice of LP parameters, see Lagrange et al. (2007).

The LP coefficients for track m are used to predict frequency f_m^{pr} and amplitude a_m^{pr} values for frame k. When a new sinusoidal track starts and there are not enough values to compute the LP coefficients, the mean frequency and amplitude are used as the predicted values. The error between predicted values for track m and peak n are given by a measure of Euclidean distance

$$E_{m,n} = \sqrt{\left[12 \lg\left(\frac{f_n^{obs}}{f_m^{pr}}\right)\right]^2 + \left[\alpha\, 20 \log_{10}\left(\frac{a_n^{obs}}{a_m^{pr}}\right)\right]^2} \tag{3.5}$$

where f_n^{obs} and a_n^{obs} are the observed frequency and amplitude values for peak n. The first term measures distance in semitones and the second in dB with a scale factor α. Informal tests have shown $\alpha = \frac{1}{12}$ to offer a reasonable weighting between prediction errors in frequency versus those in amplitude.

Each track m selects the best continuation peak n over all N peaks subject to constraints on the maximum allowable difference in predicted versus actual frequency. More precisely, $|f_n^{obs} - f_n^{pr}| < \Delta f_{max}$ where Δf_{max} is proportional to the analysis frequency f_a. For harmonic sounds, $\Delta f_{max} = \frac{3}{4} f_a$ is a reasonable default value. Δf_{max} is a user-controllable parameter

and can be reduced for sounds with very stable partials or increased for sounds with partials that exhibit wide frequency excursions.

As with guide based tracking methods, linear prediction partial tracking can span temporal gaps in the analysis data. Tracks that fail to find a match either become inactive or lie dormant for several successive frames. In the case of a track that has been dormant for j frames, linear prediction is used to predict the $j + 1^{\text{th}}$ values for matching to candidate peaks.

3.3 Resynthesis

SPEAR offers several different synthesis methods. For real-time playback, the inverse FFT method (abbreviated as IFFT or FFT^{-1}) offers excellent efficiency and very good quality (Rodet and Depalle 1992). Although most synthesis artifacts of the standard IFFT method can be minimized with the use of appropriate overlap-add windows, extremely rapid modulations of frequency or amplitude may not be adequately reproduced. For the highest quality sound, SPEAR can perform oscillator bank synthesis with optional cubic phase interpolation as in the classical McAulay-Quatieri method. SPEAR also supports the resynthesis of Loris RBEP files which include a noise component for each breakpoint (Fitz 1999). These so-called bandwidth enhanced partials can be synthesized with either the IFFT method or noise modulated oscillators.

3.3.1 Inverse FFT Synthesis

Since the DFT is invertible, it is possible to synthesize any desired time domain waveform by constructing the complex spectrum of the desired signal and applying the inverse DFT. In this case we are interested in the synthesis of sinusoids which, as detailed in section 2.1.4, have well understood spectra. A time domain windowed sinusoid can be synthesized by the following procedure:

1. shift the complex spectrum of the window function so that it is centered on the bin frequency of the desired sinusoidal frequency

2. scale the spectrum of the window function according to the desired complex amplitude (magnitude and phase)
3. accumulate the scaled shifted window function into the DFT buffer
4. perform the inverse DFT

Steps 1–3 correspond to convolving the window spectrum with the spectrum of a sinusoid (a line spectrum). Because convolution is a linear operation, steps 1–3 can be performed for each sinusoid to be synthesized. The inverse DFT can then computed as a final step. The final DFT only adds a constant amount of work regardless of the number of sinusoids. Provided that steps 1–3 can be carried out efficiently, we have the basis for an algorithm to quickly synthesize a sum of many sinusoids.

Step 1 can be performed efficiently by pre-computing an oversampled spectrum of the window function. For a DFT/IDFT of length N, we first compute a time domain window function of $h(n)$ of length N. The window function is then zero padded (zeros appended) so that it has length $M = N \times R$, where R is the oversampling factor (R is typically chosen to be a power of 2). Call this zero padded window $h'(n)$.

$$h'(n) = \begin{cases} h(n) & \text{for } 0 \leq n < N \\ 0 & \text{for } n < 0 \text{ or } n \geq N \end{cases} \tag{3.6}$$

The DFT of the zero padded window gives the oversampled interpolated spectrum of the window $H(k)$.

$$H(k) = \sum_{n=0}^{N-1} h'(n) e^{-i2\pi \frac{k}{N} n} \qquad k = 0, 1, 2, \ldots, N-1 \tag{3.7}$$

The window spectrum can then be shifted to any fractional bin position that is a multiple of $\frac{1}{R}$. For bin frequencies that are not a multiple of $\frac{1}{R}$, the window value may be approximated by index truncation, index rounding, linear interpolation, or some higher order interpolation scheme.

Steps 2–3 — scaling and accumulating the complex spectrum of the window function — can be performed with various degrees of efficiency depending on the precision required. To precisely synthesize the desired sinusoid, all N values of the window spec-

trum must be scaled and accumulated into the DFT buffer. Equation 3.8 summarizes the shifting, scaling, and accumulation of the window spectrum $W(n)$ for a sinusoid of amplitude A_n and bin frequency of b_n. Note that to synthesize a real valued-sinusoid, we must represent half of the amplitude as a complex sinusoid, and half as its complex conjugate.

$$X(k) = \frac{A_n}{2} H(\lfloor (k - b_n)R \rfloor) e^{i\phi} + \frac{A_n}{2} H(\lfloor (N - k - b_n)R \rfloor) e^{-i\phi} \tag{3.8}$$

The symmetry of the DFT buffer means that $X(k)$ can be constructed by first accumulating $N/2$ values in the positive frequency half ($0 \leq n \leq N/2$), and then reflecting their complex conjugate in the negative frequency half ($N > n > N/2$).

The procedure implies one table lookup (for the window function), and one complex multiply (four scaler multiplies) for each of the $\frac{N}{2} + 1$ values. Compared to synthesis via a table lookup oscillator, this method is actually less efficient. The oscillator requires one table lookup and one scaler multiply for each of the N samples. However, significant savings can be achieved if we approximate the scaled and shifted window function spectrum.

Although the complete window function spectrum has length N, it should be noted that many of the values are close to zero. Therefore, we can approximate steps 2–3 by scaling and shifting only J window samples immediately surrounding the main lobe (typically between 7–15 samples). Laroche (2000) calls the restricted set of samples the "spectral motif." Accumulating only the spectral motif (rather than the entire window spectrum) reduces the number of table lookups and complex multiplies to a constant J, regardless of the DFT buffer size.

Although any window function can be used, for computational efficiency we desire a window function with a reasonably narrow mainlobe and low sidelobes. When the DFT buffer is filled, only the values of the main lobe and the first few side lobes need to be computed.

The result of the inverse DFT is a sum of sinusoids (each with constant amplitude and frequency) that have been multiplied by the time domain window function $h(n)$.

To synthesize time varying amplitudes and frequencies, an overlap-add process can be used. We begin by multiplying each sample in the DFT buffer by $\frac{1}{h(n)}$. This effectively removes the time domain effect of the window, leaving a constant amplitude sinusoid. We then apply a triangular synthesis window $s(n)$ which gives the desired linear amplitude interpolation when overlap-added together.

Since frequency is constant across each DFT buffer, there will be some modulation artifacts resulting from overlap-adding. These effects can be reduced by carefully matching the phase of each sinusoid from one buffer to the next. For an overlap factor of 2, the phase should be matched at the midpoint of the triangular slope (Rodet and Depalle 1992).

An important implementation detail of the IFFT method concerns the imprecision introduced by the approximation of the window spectrum. By only using J samples centered on the main lobe, the window spectrum has been band-limited. Effectively the window spectrum has been multiplied by a rectangular window. Since multiplication in the frequency domain results in convolution in the time domain, the time domain window is no longer limited to N samples. The result is time domain aliasing which introduces distortion in the resynthesized sinusoid. This distortion is particularly evident at the edges of the time domain buffer where the window function approaches its minimum. When the window function is divided out (multiplied by $\frac{1}{h(n)}$), these distortions are dramatically amplified. It is particularly evident for window functions that taper to zero at the edges (windows in the Blackman-Harris family, for example). Figure 3.4 compares a sinusoid with IFFT synthesized versions using different values of J. Note that even with large values of J, significant error is still present. To avoid this problem, some number of samples D at the beginning and end of the DFT buffer must be discarded. D is typically on the order of $\frac{N}{8}$, so the reduction in synthesis efficiency is negligible.

Inverse FFT synthesis appears to offer some open areas for continued research. More detailed studies of signal distortion in inverse FFT synthesis would be welcome. It would be helpful to have a more precise analysis of the effects of different synthesis windows, window oversampling and interpolation strategies, the choice of J (controlling the size

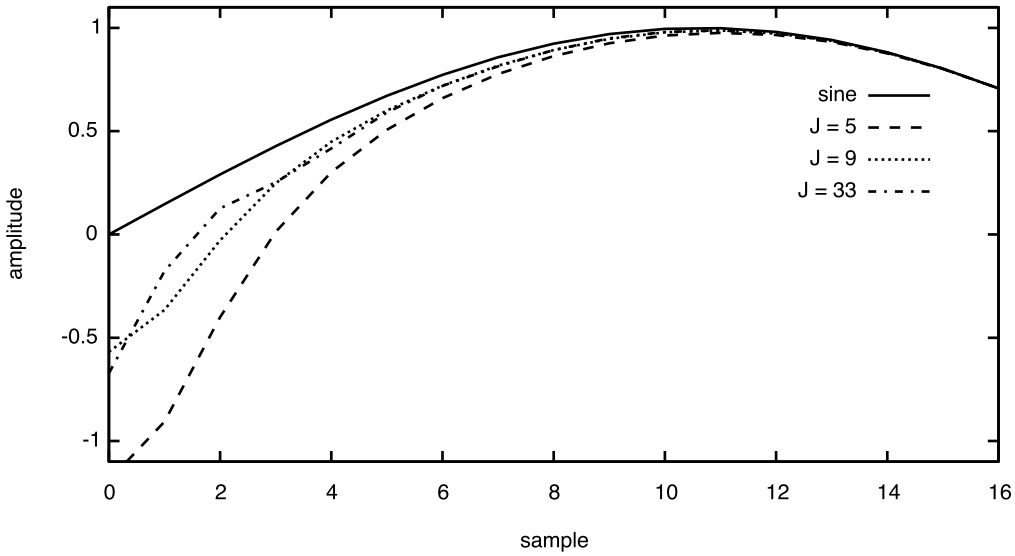


Figure 3.4: Comparison of sinusoids generated via IFFT synthesis with different values of J (the spectral motif length) showing the first 16 samples of the time domain signal where $N = 64$, frequency $= 1.5$ cycles per N samples. Window type is Blackman-Harris.

of the spectral motif), and the effect of D on signal to noise ratio. Jean Laroche's (2000) investigations provide some important preliminary insights in this area.

3.3.2 IFFT Synthesis Implementation

With an overlap add factor of 2, each inverse DFT computes $\frac{N-2D}{2}$ new output samples. In the current implementation, SPEAR performs IFFT synthesis with $N = 512$, $D = \frac{N}{8}$. These parameters allowed for real-time resynthesis of hundreds of partials on relatively modest hardware.[1] Future versions of SPEAR may allow the user to fine tune the resynthesis parameters. In particular, it may be desirable to reduce N. Because frequency and amplitude are constant across each DFT buffer, rapid frequency changes may not be well represented with larger N.

In the current implementation, the spectral motif length, J, is fixed at 9. Because of the problem of distortion at edge of the DFT buffer, the Hamming window, as defined in equation 3.9, was chosen for the synthesis window. Since we divide by the window function, windows that taper to zero will amplify distortions rather dramatically as the

[1] PowerPC G3 class processor at 400 MHz — circa 2003

inverse tends toward infinity. The minimum value of the Hamming window at the buffer edge is $\frac{2}{23}$.

$$h(n) = \begin{cases} \frac{25}{46} - \frac{21}{46}\cos\left(\frac{2\pi n}{N-1}\right) & \text{for } 0 \leq n < N \\ \\ 0 & \text{for } n < 0 \text{ or } n \geq N \end{cases} \tag{3.9}$$

Real time resynthesis proceeds according to a time counter t that indicates the current location in the analysis data. Whenever a new output buffer is required, the synthesis callback checks to value of t to determine the set of sinusoids that need to be synthesized. The center of each DFT synthesis window is represented by t. P_n is the set of partials active at time t_n. The amplitude and frequency for each partial in P_n is determined by linear interpolation. At time t_{n+1}, we have a new set of partials P_{n+1}. All partials in $P_n \bigcap P_{n+1}$ must have phase continuity at the overlap-add midpoint. This is achieved by maintaining a set of virtual oscillators that store the previous phases. Each oscillator structure is associated with an active partial by means of a hash table (Cormen et al. 1990). Thus, it is reasonably efficient to lookup phase as partials turn on and off. Moreover t can proceed at any rate through the analysis data. This facilitates the ability to manually "scrub" forward and backward through the sound. The analysis data can also be freely transformed concurrent with resynthesis. SPEAR does appropriate data locking to make sure the analysis data structures remain consistent between the concurrent resynthesis and user interface threads.

3.3.3 Oscillator Bank Resynthesis

SPEAR can optionally perform a non-realtime oscillator bank resynthesis with linear interpolation of frequency and amplitude as shown in equation 2.24 (page 40). This provides a much higher fidelity result than the IFFT method.

3.3.4 Phase Management

As noted in section 2.4.4, linear interpolation of frequency will, in general, only retain the phase of the first breakpoint of a partial. Although in many situations the phase information may not be perceptually important, there are cases where temporal phase coherence between partials is quite important, for example in the reproduction of transients. We also seem to be particularly attuned to the phase relationships in speech signals.

McAulay and Quatieri (1986) developed a frequency/phase interpolation method that maintains phase and frequency at each breakpoint. Consider two successive breakpoints i and $i+1$ with phases ϕ_i and ϕ_{i+1} and frequencies ω_i and ω_{i+1} (here we are expressing frequency in radians per sample rather than cycles per second). Since frequency is the derivative of phase, we desire a phase interpolation function with endpoints equal to ϕ_i and ϕ_{i+1} and first derivatives at the endpoints equal to ω_i and ω_{i+1}. Independent control of the slope at two points requires a cubic function, so we begin by defining a cubic phase of the form

$$\theta(n) = \zeta + \gamma n + \alpha n^2 + \beta n^3 \quad \text{for } n = 0 \ldots S \tag{3.10}$$

where n is the sample index and S is the number of samples from breakpoint i to breakpoint $i+1$. The first derivative (the frequency function) is

$$\theta'(n) = \gamma + 2\alpha n + 3\beta n^2 \quad \text{for } n = 0 \ldots S \tag{3.11}$$

The endpoints at sample 0 and sample S are

$$\theta(0) = \zeta = \phi_i \tag{3.12}$$

$$\theta'(0) = \gamma = \omega_i \tag{3.13}$$

$$\theta(S) = \zeta + \gamma S + \alpha S^2 + \beta S^3 = \phi_{i+1} \tag{3.14}$$

$$\theta'(S) = \gamma + 2\alpha S + 3\beta S^2 = \omega_{i+1} \tag{3.15}$$

With four equations we can solve for the four unknowns. ζ is equal to the initial phase, and γ is equal to the initial frequency, which leaves only to solve for α and β. One complication

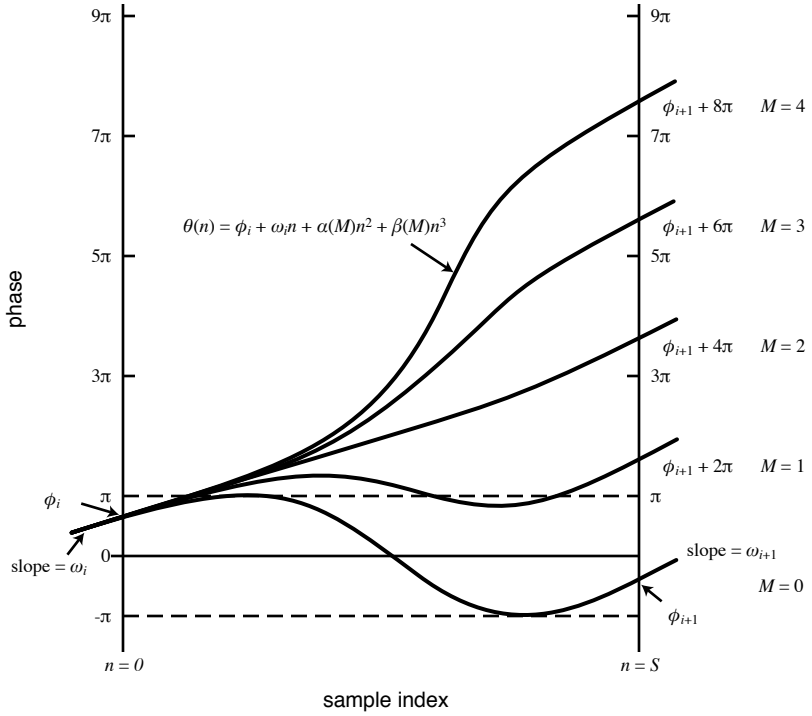


Figure 3.5: Family of cubic phase interpolation functions with different values of M (after McAulay and Quatieri (1986)).

is that the final phase is measured modulo 2π, so in fact $\theta(S)$ should be expressed as

$$\theta(S) = \zeta + \gamma S + \alpha S^2 + \beta S^3 = \phi_{i+1} + 2\pi M \tag{3.16}$$

for some integer M. M defines a family of cubic phase functions with different amounts of phase unwrapping (see figure 3.5).

The solutions for α and β are parameterized in terms of M as follows:

$$\alpha(M) = \frac{3}{S^2}\left(\phi_{i+1} - \phi_i - \omega_i S + 2\pi M\right) - \frac{1}{S}\left(\omega_{i+1} - \omega_i\right) \tag{3.17}$$

$$\beta(M) = \frac{-2}{S^2}\left(\phi_{i+1} - \phi_i - \omega_i S + 2\pi M\right) + \frac{1}{S^2}\left(\omega_{i+1} - \omega_i\right) \tag{3.18}$$

McAulay and Quatieri show that the optional choice of M is that which makes the phase curve maximally smooth (the maximally smooth choice for figure 3.5 is $M = 2$). This can be defined in terms of minimizing the second derivative of $\theta(n)$ with respect to sample

index. The result is that the maximally smooth M is the closest integer to x, where

$$x = \frac{1}{2\pi}\left[(\phi_i + \omega_i S - \phi_{i+1}) + (\omega_{i+1} - \omega_i)\frac{S}{2}\right] \tag{3.19}$$

The complete phase function is

$$\theta(n) = \phi_i + \omega_i n + \alpha(M)n^2 + \beta(M)n^3 \quad \text{for } n = 0 \ldots S \tag{3.20}$$

It is important to note that cubic phase interpolation will generally only work for *unmodified* resynthesis. Partials that have undergone time dilation, transposition, or frequency shifting cannot be synthesized reliably with this method since the resulting phase functions are no longer "smooth" enough to avoid frequency modulation artifacts. Methods for preserving phase under modifications remain an interesting open problem. It may prove effective to resynchronize phase only at certain moments (as suggested by the work in Robel (2003) and Fitz and Haken (2002)), or slightly modify the breakpoint frequencies to smooth out the phase function. Phase interpolation in the unmodified case is important for recovering the residual signal, which is discussed in section 3.6.

3.4 Data Storage

A popular data storage model in analysis-synthesis applications is a sorted list of time frames. Each frame contains a list of peaks (amplitude, frequency, and phase) that are connected to successive time frames via either index numbers or pointers.

Rather than represent the analysis data as a sorted list of time frames, SPEAR uses a list of partials that are represented by breakpoint functions of time versus amplitude, frequency, and phase. With this storage model the implementation of cut, copy, and paste operations, of both entire partials or segments, is straightforward. Additionally, partials can be individually shifted, stretched, or compressed in time without resampling to fixed frame time points. A further advantage is that the storage model can easily support multirate analysis data (Levine 1998) or time reassigned breakpoints (Fitz 1999).

3.4.1 Time-span Frames

A common operation, particularly during synthesis, is determining the list of partials (and breakpoint segments) crossing a particular time point. In the frame based storage model, a binary search of sorted frames quickly determines the partials that are active at any particular time.

With a list of breakpoint partials, this query requires an iteration through all the partials in the data set. For a short sound, this may only require looking at several hundred elements, but for longer sounds there are likely to be many partials of short duration. For example, the total number of partials for a one minute sound could be well over twenty-thousand. During synthesis, continued iteration over thousands of partials is too inefficient. A solution is to maintain a parallel data structure of time-span frames. For each frame, a list is kept of all partials that have breakpoints within the frame's time span — typically on the order of 0.125 seconds. Figure 3.6 shows an example of ten partials segmented into five frames where each frame contains a list of its active partials. The active partial lists maintain a relatively constant size as partials turn on and off. For a typical sound, this reduces active partial lookups to an iteration over only several hundred elements.

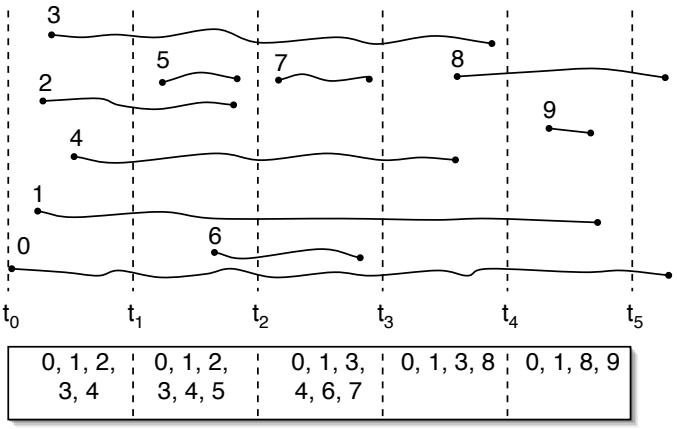


Figure 3.6: Data structure for division of partials into time-span frames. Comma delimited lists for each frame indicate the active partials.

3.5 Transient Sharpening

As observed in section 3.1.1, the choice of window length directly affects both the frequency and temporal resolution of the analysis. For polyphonic or inharmonic sounds in particular, we desire a long analysis window so as to be able to resolve closely spaced partials. However, as the window length increases, the temporal resolution decreases. This is a particular problem for sounds that contain transients. All signal energy is spread across the window and assigned to a breakpoint located at the temporal center of the analysis window rather than at the precise location of the transient. Moreover, since analysis windows overlap, the energy will be distributed across multiple windows and assigned to multiple breakpoints. This results in the familiar pre- and post-echo effects of traditional STFT analysis/synthesis.

Fitz (1999) shows how the *method of reassignment* can be used to improve the temporal resolution *and* reduce pre- and post-echo. The method of reassignment begins by noting that traditional STFT analysis assigns all energy included in the window to the temporal center of the window. When there is significant energy in a particular frequency band that is located near the edge of a window, this energy should instead be assigned to the center of gravity (time-frequency centroid) of the window. This time-frequency coordinate can be computed from the derivatives of the phase spectrum. The time-frequency centroid will be located at the point where phase is changing most slowly (where the partial derivative of phase with respect to time and phase with respect to frequency is zero).

Auger and Flandrin (1995) and Plante et al. (1998) show how to efficiently compute the point of reassignment as a ratio of Fourier transforms. The reassigned time for frequency bin k is given by

$$t(k) = t + \Re e \left\{ \frac{X_{ht}(k)\ \overline{X_h(k)}}{|X_h(k)|^2} \right\} \tag{3.21}$$

where t is the original center of the analysis window (in samples). $X_h(k)$ is the DFT of the signal using the usual analysis window $h(n)$. $X_{ht}(k)$ is the DFT of the signal using a special time-weighted analysis window $ht(n)$. $ht(n)$ is defined as the product of $h(n)$ and

a ramp function from $-\frac{M-1}{2}$ to $\frac{M-1}{2}$, where M is the window length.

$$ht(n) = h(n)\left[n - \frac{M-1}{2}\right] \qquad n = 0, 1, 2, \ldots, M-1 \tag{3.22}$$

The real portion of this ratio represents the time reassignment in samples.

The reassigned frequency for bin k is given by

$$\omega(k) = \omega_k - \Im m\left\{\frac{X_{h\omega}(k)\ \overline{X_h(k)}}{|X_w(k)|^2}\right\} \tag{3.23}$$

where ω_k is the bin frequency, and $X_{h\omega}(k)$ is the DFT of the signal using a frequency-weighted analysis window $h\omega(n)$. $h\omega(n)$ can be computed by taking the DFT of $h(n)$, multiplying by a frequency ramp, and then taking the inverse DFT. Note that the time-frequency center of gravity is computed independently for each frequency bin.[2]

Pre- and post-echo can be dramatically reduced by removing breakpoints with large time corrections (Fitz 1999; Fitz and Haken 2002). Specifically, if the time correction is larger than analysis hop size (and the hop size is less than the window length — which is always the case in this implementation), a breakpoint can be safely removed since it will be detected in a neighboring frame. Figures 3.7 and 3.8 show the analysis of a castanet click that occurs at about 0.868 seconds.

Note that the transient event is well localized, and pre- and post-echo are dramatically reduced. In order to avoid a complete amplitude discontinuity at the onset of a partial, a zero amplitude breakpoint is joined to the head of each partial. Informal observations indicate that although the reassignment method does an excellent job of localizing sharp transients, there is some loss of energy, likely due to the removal of the breakpoints with large time corrections. The addition of a zero amplitude breakpoint creates a short fade into each transient which has the additional benefit of raising the average signal level around the transient event. Further experimentation should be conducted to determine whether transient breakpoints would benefit from an additional amplitude boost.

[2] Note that equations 3.21 and 3.23 differ from those in Auger and Flandrin (1995), Plante et al. (1998), and Fitz (1999) in that we *add* the reassignment term for time reassignment and *subtract* for frequency reassignment. The sign depends on whether or not the STFT is defined with a time reversed window. We use the definition in

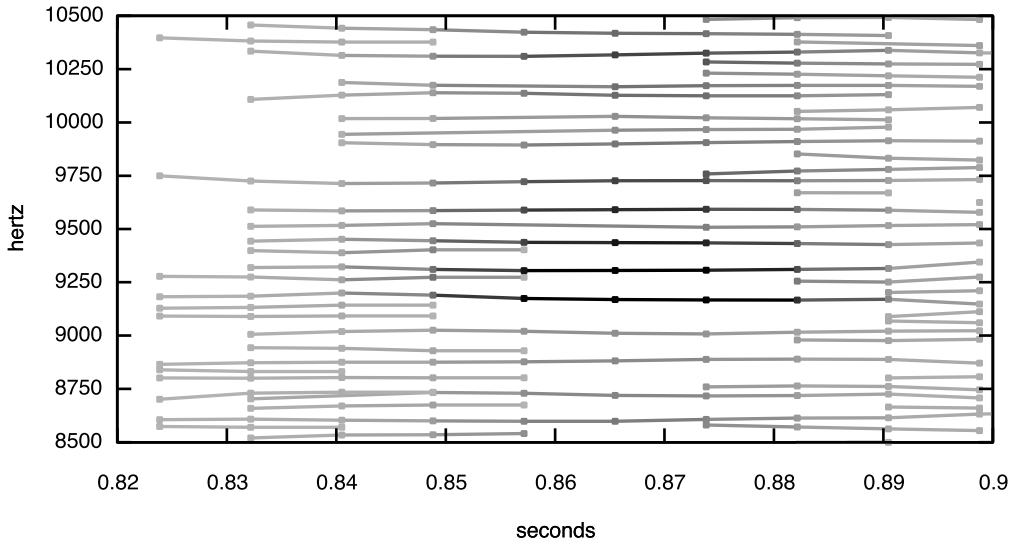


Figure 3.7: Breakpoints from the analysis of a castanet click using the standard sinusoidal model.

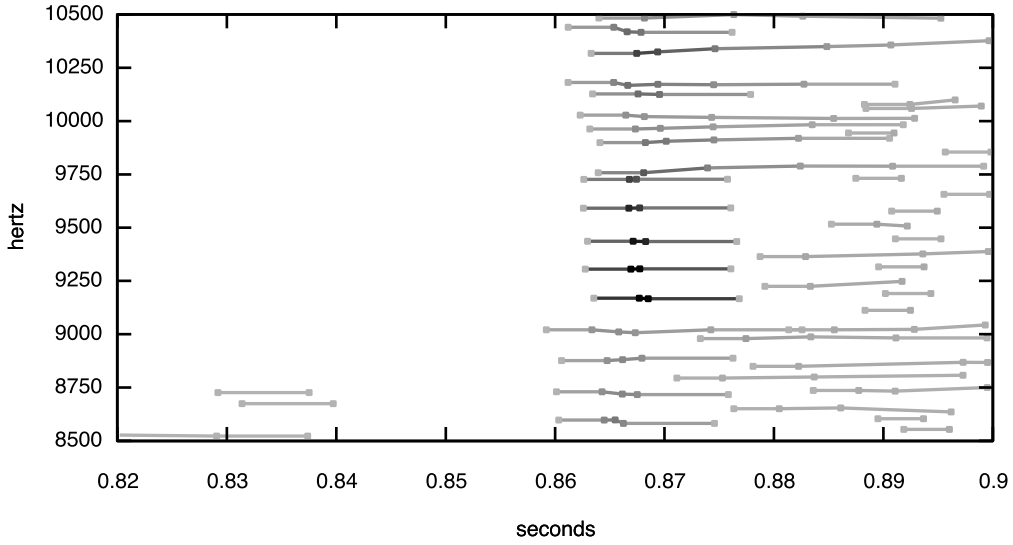


Figure 3.8: Breakpoints from the analysis of a castanet click using the time reassignment method. Breakpoints with large time corrections have been removed.

3.6 Noise Modeling

Serra and Smith were the first to develop extensions to sinusoidal modeling for representing noise. The SMS (Spectral Modeling Synthesis) system offers two different ways to model non-sinusoidal components. In the residual model, the sound is first analyzed using sinusoidal modeling. Next, the signal is resynthesized using the oscillator bank method with cubic phase interpolation as described in section 3.3.4:

$$\hat{x}(n) = \sum_{i=0}^{N-1} \hat{A}_i(n) cos(\hat{\theta}_i(n)) \tag{3.24}$$

The residual signal $e(n)$ is then computed by subtracting the resynthesized signal $\hat{x}(n)$ from the original $x(n)$.

$$e(n) = x(n) - \hat{x}(n) \tag{3.25}$$

Because phase is preserved in the resynthesis, this subtraction can take place in the time domain. The residual $e(n)$ will typically consist of noise, attack transients, and pre- and post-echo. The original signal can of course be perfectly reconstructed by adding the residual to the resynthesis. However, typically we wish to make some sonic transformations prior to resynthesis. In this case a more flexible modeling of the residual is required.

The stochastic model represents the residual in the frequency domain. For each frame l of the analysis, we compute a residual magnitude spectrum from the magnitude spectrum of the original signal and the magnitude spectrum of the phase accurate resynthesis.

$$|E_l(k)| = |X_l(k)| - |\hat{X}_l(k)| \tag{3.26}$$

$X_l(k)$ is the DFT of the original signal at frame l and $\hat{X}_l(k)$ is the DFT of the sinusoidal resynthesis. Assuming that $e(n)$ is a stochastic signal, it can be modeled as a set of magnitude spectra with random phase. Time stretching or transposition of the residual can be accomplished by interpolating new magnitude spectra $|\hat{E}_l(k)|$, setting the phase

equation 2.13 (page 20) in which the window is *not* time reversed. Auger and Flandrin and Plante et al. define their STFT with a time reversed window. Thanks to Kelly Fitz for explaining the reason for this difference.

spectra to random values between $-\pi$ and π, applying the inverse DFT, and then overlap adding.

Optionally $|E_l(k)|$ can be computed directly in the frequency domain by subtracting the frequency domain peaks detected in frame l from $|X_l(k)|$. $|E_l(k)|$ may also be approximated as a spectral envelope. The spectral envelope can be viewed as a filter that shapes a flat noise spectrum.

Several variants of the deterministic plus stochastic model have been proposed. Pampin (2004) developed the ATS system which separates the energy in $E_l(k)$ into 25 bands; each band corresponding to a step on the Bark frequency scale. The noise energy in each band may then be optionally redistributed among the partials.

3.6.1 Bandwidth Enhanced Sinusoids

This approach is similar to the *bandwidth enhanced* additive sound model described in Fitz (1999) and implemented in Loris. Bandwidth enhanced synthesis models the signal as a sum of bandwidth enhanced oscillators—sinusoidal oscillators that are amplitude modulated by a lowpass filtered noise signal $\chi(n)$.

$$y(n) = [A(n) + \beta(n)\chi(n)] \cos\left(n\frac{2\pi F(n)}{f_s}\right) \tag{3.27}$$

Equation 3.27 defines a bandwidth enhanced oscillator with time varying amplitude $A(n)$ and time varying frequency $F(n)$. The amplitude of the lowpass filtered noise is controlled by $\beta(n)$. Multiplication of a sinusoid by noise in the time domain results in the convolution of the noise spectrum with the sinusoid spectrum. The multiplication shifts the noise spectrum so that it is centered at frequency $F(n)$.

In Loris, each breakpoint has an additional noise parameter κ that varies between 0 and 1. If $\hat{A}(n)$ is the local average partial energy, the the bandwidth enhanced oscillator is defined by

$$y(n) = \hat{A}(n)\left[\sqrt{1-\kappa} + \sqrt{2\kappa}\chi(n)\right] \cos\left(n\frac{2\pi F(n)}{f_s}\right) \tag{3.28}$$

When $\kappa = 0$, $y(n)$ is a pure sinusoid. As κ increases, the amplitude of the narrowband noise also increases. This model has the advantage that noise is tightly coupled to the sinusoidal partials. As the sound is edited and manipulated — for example as partials are transposed — the noise will follow. It is sometimes convenient that the noise amplitude is controlled by a single parameter κ. However since the noise parameter is coupled to overall partial amplitude, several square root operations must be invoked, which is somewhat inefficient for realtime resynthesis.

SPEAR currently does not implement noise analysis, but it will resynthesize noise of partials imported from Loris SDIF files (see section 4.6 for more information about the use of SDIF). Loris defines the lowpass filtered noise signal as white noise filtered by a 3rd order Chebychev IIR filter with cutoff of 500 Hz, and SPEAR attempts to match this behavior. Although the Loris noise representation is compact (based on a single parameter κ), it lacks generality and is not an ideal choice for cross program data exchange. A more flexible system would specify noise amplitude and bandwidth independently of sinusoidal amplitude. Such a data model would better support different noise analysis strategies and would allow for experimentation with different noise bandwidths. Expanding the number of noise parameters would require either a new SDIF matrix type or an extension of an existing type. Expanding the noise handling capabilities of SPEAR is an important area for future work.

4. Editing and Transformation

4.1 Graphical User Interface

One of the motivations behind the development of SPEAR was the desire for a graphical interface that would allow the user to make a quick visual assessment of analysis quality. In spite of the best efforts in the choice and implementation of the analysis algorithms, sonic artifacts are common. Even with a reduced set of available analysis parameters, some trial and error is usually necessary to determine the optimal settings for a particular application.

With systems that are command-line oriented (Loris, ATS, SNDAN), the user is left to either resynthesize and listen carefully for artifacts, or use an additional plotting or visualization tool. SPEAR offers an intuitive interface that allows the user to quickly zoom in on analysis artifacts. Problematic areas of the sound can be isolated and auditioned, and the data can be modified as desired. For example, spurious sinusoidal tracks can be deleted, attenuated, or joined with other neighboring tracks.

The analysis data is displayed with time on the abscissa and frequency on the ordinate. As in a grayscale spectrogram, varying levels of shading indicate the relative amplitudes of partials. Figure 4.1 illustrates the overall user interface. Following the model of time domain waveform editors, the amount of detail shown for the breakpoint functions varies according to a user controlled zoom factor. At high zoom levels, every breakpoint is shown and clickable handles allow individual manipulation in time and frequency (figure 4.2). At lower zoom levels, the decrease in detail avoids visual clutter and significantly speeds redraw.

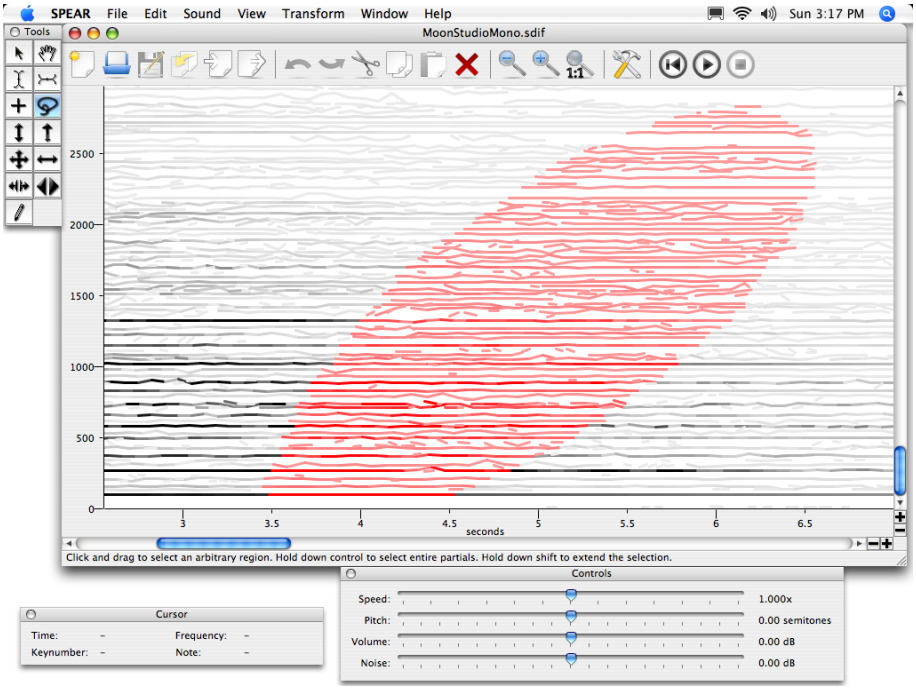


Figure 4.1: Zoomed-out display showing lasso selection and playback control sliders.

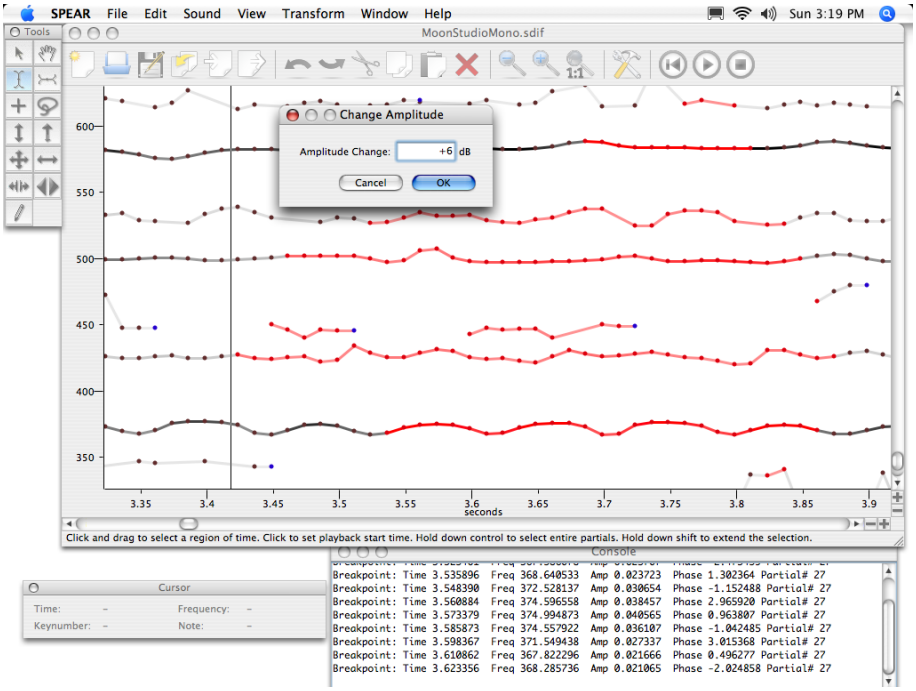


Figure 4.2: Zoomed-in display showing amplitude adjustment dialog.

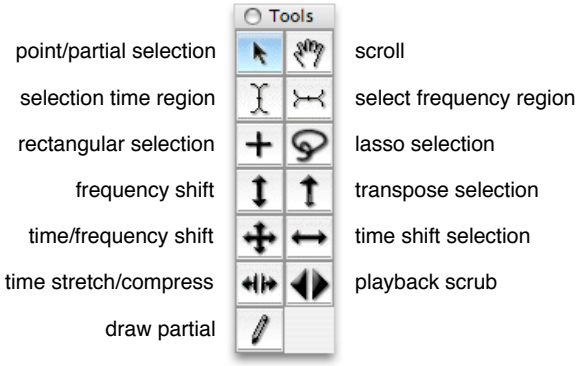


Figure 4.3: The SPEAR tool palette.

A tool palette (figure 4.3) offers different modes for graphical selection and manipulation. Interaction follows a selection and direct manipulation paradigm (Shneiderman 1983). For example, clicking on a partial selects it and shift-clicking adds partials to the current selection. Choosing the transpose tool then allows the selected partials to be transposed by dragging up or down. Additional selection modes allow the user to sweep out areas in time and frequency or to draw arbitrary time-frequency regions with a "lasso." Following the model of graphics editors, arbitrary selections can be made up of a union or difference of individual contiguous selections.

Editing tools and commands that operate on the current selection include transposition, frequency shifting, time shifting, time stretching, and amplitude adjustment. Unlimited undo/redo is supported for all editing operations, and most editing can be performed while the sound is being synthesized. For further real-time control, sliders allow adjustment of the overall transposition level, amplitude, and playback speed (figure 4.4). New partials can be added by drawing with the pencil tool. New breakpoints are added with amplitude proportional to the dragging speed.

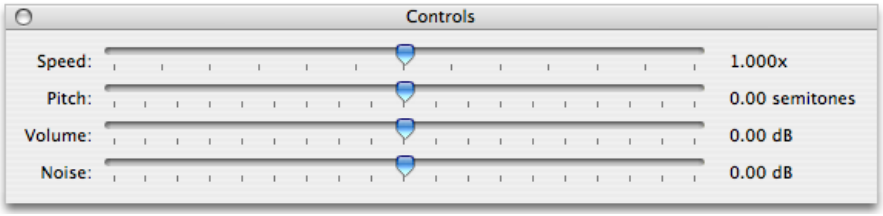


Figure 4.4: The SPEAR playback control sliders.

4.2 Selection Model

During the development of SPEAR, two possible selection models and implementations were considered. The first may be termed the "region selection" method. In this model, a selection is represented as a polygon (concave or convex) in the time-frequency plane. (It would also be possible to represent the selection in the time-amplitude plane if desired.) Selections would be defined by sweeping out regions graphically or by setting numeric parameters. More complex selections could be represented as the union of several regions. The second selection model is "point selection" method. In this case, the selection is represented as a collection of references to the specific breakpoints of interest.

Given the desire to allow both extremely detailed editing and complete flexibility in the temporal positioning of breakpoints, it was decided that the point selection model was best. To reduce implementation and interface complexity, this is the only selection model that has been implemented to date.

There are a few drawbacks to this model. In some cases it may be desirable to define a transformation (such as an amplitude envelope) that occurs across a certain time span. In this case, the region selection model is ideal. When the user requests such an operation on a point selection (for example, an amplitude fade), the bounding region of the point selection must first be computed. It may also be desirable to define a time-frequency region (such as a formant) that remains fixed when partials are shifted or transposed. In this case, the region selection model is required.

Point selections are implemented by associating a selection list with each breakpoint array. The selection is stored efficiently as a range set data structure (Phillips 1999). Rather than maintaining the index of each breakpoint, the range set represents contiguous runs of breakpoints by the start and ending points of the run. More formally, if n breakpoints starting at index i are selected, the range set represents this selection as the pair $(i, i+n)$, rather than the list $[i, i+1, i+2, \ldots, i+n-1]$. Discontiguous selections are represented as an ordered list of pairs: $\left[(i_0, i_0+n_0), (i_1, i_1+n_1), \ldots, (i_j, i_j+n_j)\right]$. Given that contiguous selections are the most common case, the range set is very efficient in both time and

space requirements. The range set also supports operations to efficiently add and coalesce ranges, as well as delete and split ranges.

4.2.1 Rule-based Selection

Selections can also be made programatically. For example, the user can choose to select all partials with a duration less than some specified value. Or, the user can choose to select partials with an average amplitude less than some desired level. Currently the number of choices for rule-based selection is quite limited, however one can imagine any number of selection criteria: minimum or maximum frequency, average frequency, frequency range, frequency periodicity (vibrato rate), phase stability, amplitude range, maximum amplitude, etc. Comparison ($=$, $>$, $<$) and logical operations (NOT, AND, OR, XOR) could be used to develop any number of rule-based selection criteria. The extent of possibilities suggests that a scripting language (rather than a GUI) would be best way to implement flexible rule-based selection. It is hoped that future versions of SPEAR will include this capability.

4.3 Editing Features

The SPEAR document model supports cut, copy, and paste of any selected data. When pasting data from one document to another, care must be taken to make sure data types match. Breakpoints are required to have time, frequency, and amplitude components. They may also have phase and noise components, but these are optional (for a discussion of the noise component, see section 3.6.1, page 62). Analyses performed with SPEAR will have at least time, frequency, amplitude, and phase. Since analysis data may be imported from other software and via different file formats (see section 4.6), the document model must be prepared to support different breakpoint types. When pasting into an existing document, the pasted data is modified to match the destination document's breakpoint type. For example, when pasting data that has only time, frequency, and amplitude into a document that also has phase, the newly pasted data will have phases set to zero.

In future versions of SPEAR it may prove beneficial to allow greater flexibility of point types. Additional data types could specify specify stereo panning, 3D spatial location, noise bandwidth, etc. Partials are currently implemented as multi-dimensional arrays with contiguous allocation. A flexible point model would be better implemented with collections of one-dimensional arrays, each array corresponding to one data type.

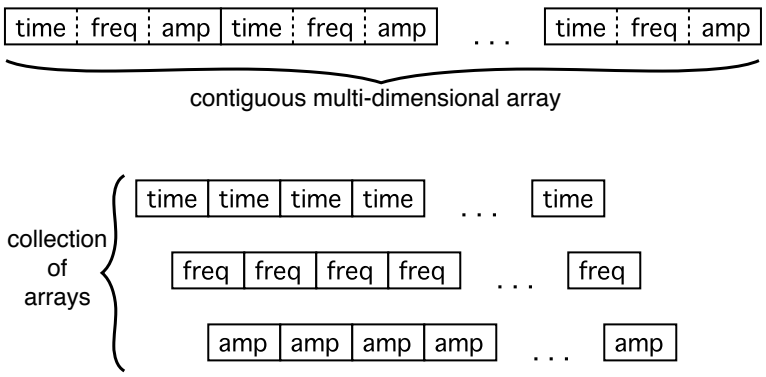


Figure 4.5: Different breakpoint storage models.

SPEAR supports unlimited undo/redo for all editing operations. Early in the design process it was decided that the undo implementation should be simple and robust, yet flexible enough to support arbitrary edits and transformations. Each undo/redo operation is modeled as an ordered collection of sub-operations consisting of any of the following:

- selection
- edit partial
- insert partial
- remove partial

These sub-operations are bundled together into a single undo or redo operation. An undo stack stores the state before the operation (this is the data required to restore the document to its previous state), while the redo stack stores state after the operation. To avoid excessive dynamic memory use, the data for edit, insert, and remove operations is stored on disk. Selections are stored in locally addressable memory. Generally this does not present a storage problem since the range map data structure is quite compact.

A selection operation consists of the partial indices and breakpoint selection ranges. An edit operation is defined as any operation on a partial that does not require the insertion or removal of any other partials (and therefore does not change any partial index numbers). Insert or remove operations are those that will alter the partial indices in the data model. Operations that can split partials into multiple segments (such as cut or delete) are modeled as a sequence of edit, insert, and remove operations. It is crucial that the atomic sub-operations are ordered properly so that the precise order of partial indices is always restored.

4.4 Time Expansion and Contraction

SPEAR supports two different modes for time expansion and contraction. The first is the "independent" mode. In this mode, each partial is expanded or contracted relative to the starting time of the partial (or selection), which remains fixed in time. If $t_k(n)$ represents the time of nth breakpoint for partial k, then the independent expansion for breakpoint n given by the following $t_k(0) + \alpha[t_k(n) - t_k(0)]$ where α is the expansion factor. In "proportional" expansion/contraction mode, the new time is relative to two fixed time boundaries T_{min} and T_{max}. Breakpoints that fall before T_{min} are left unchanged. Breakpoints falling between T_{min} and T_{max} are scaled. Breakpoints the occur after T_{max} are shifted earlier (for contraction) or later (for expansion). The following equation summarizes:

$$t'_k(n) = \begin{cases} t_k(n) & \text{for } t_k(n) \leq T_{min} \\ T_{min} + \alpha[t_k(n) - T_{min}] & \text{for } t_k(n) > T_{min} \wedge t_k(n) \leq T_{max} \\ t_k(n) + (\alpha - 1)\,[T_{max} - T_{min}] & \text{for } t_k(n) > T_{max} \end{cases}$$

Proportional expansion/contraction scales the time base uniformly and accomplishes the same thing as traditional time warping. Independent expansion/contraction can produce interesting effects that "smear" and overlap the evolution of each individual partial (in the case of expansion), or emphasize short sinusoidal segments (in the case of contraction). Figures 4.6–4.8 on page 71 illustrate the different time expansion modes.

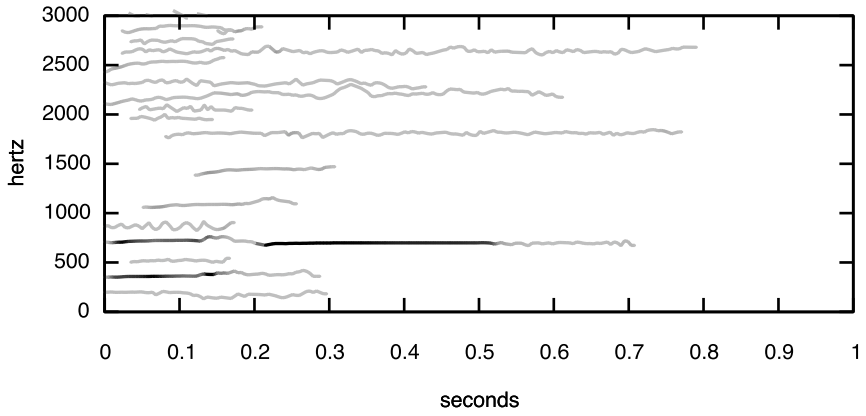


Figure 4.6: Original partials prior to time expansion.

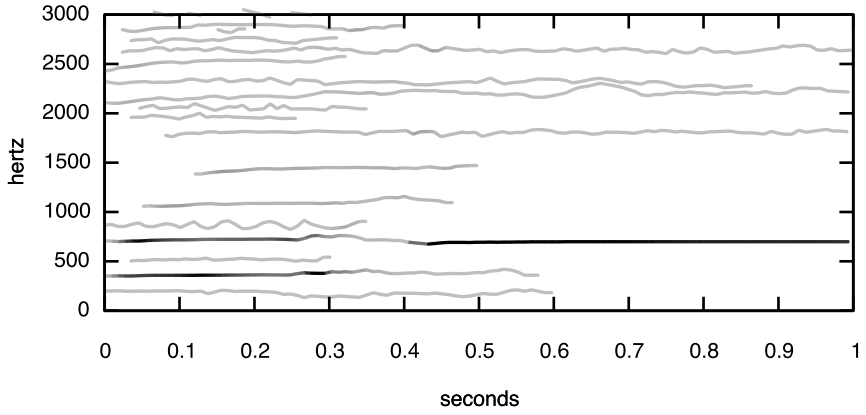


Figure 4.7: Partials time expanded in independent mode. Note that the start time of each partial is the same as in the unexpanded version.

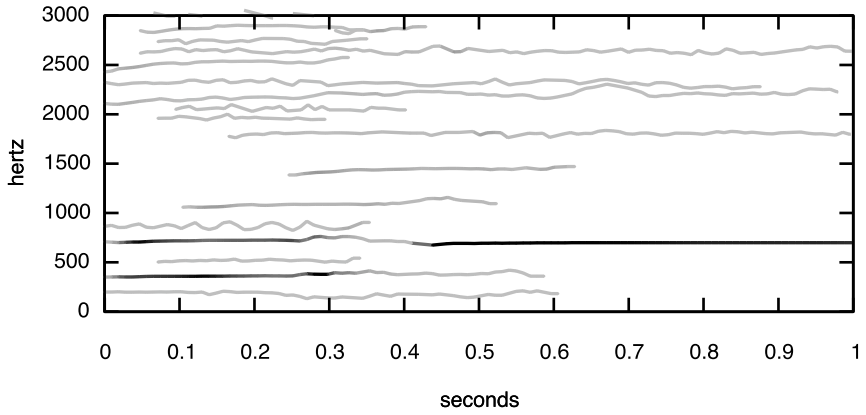


Figure 4.8: Partials time expanded in proportional mode. Note that the start time of each partial is scaled relative to time zero.

4.5 Frequency Modifications

Currently the SPEAR GUI only supports a few basic frequency-based transformations. Frequency shifting and transposition can be performed either by dragging the selection vertically with the appropriate tool or by entering a specific frequency shift (in hertz) or transposition (in floating point semitones). The interface is shown in figure 4.9

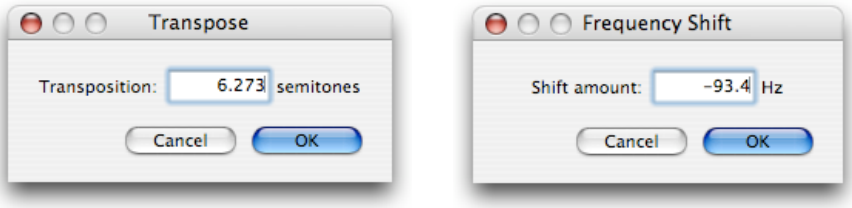


Figure 4.9: Dialog boxes for transposition and frequency shifting.

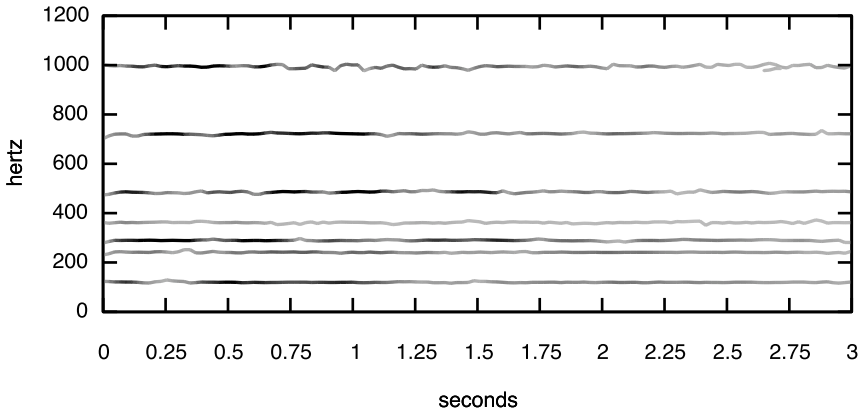
Frequency "flipping" inverts all frequencies in the range F_{min} to F_{max} around the axis $\frac{1}{2}(F_{min} + F_{max})$. For frequency $f_k(n)$ of partial k and breakpoint n, the new flipped frequency $f'_k(n)$ is given by

$$f'_k(n) = F_{max} - (f_k(n) - F_{min})$$

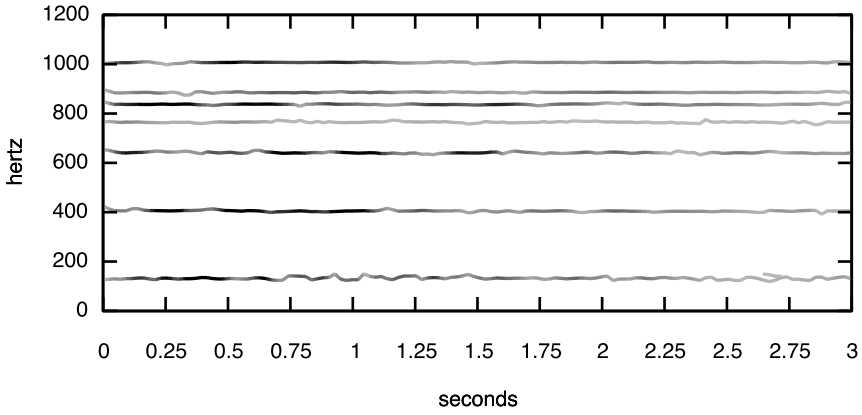
Frequency flipping can be effective when applied to specific frequency bands. For example, one might wish to try flipping the 3rd through 11th partials of an inharmonic timbre. Figure 4.10 illustrates the effect. Frequency flipping tends to be less interesting when applied to an entire instrumental sound, as the strong lower partials invert into high frequency regions (often above 10 kHz). Strong energy in this area is often perceived simply as a loud noise band.

4.6 Data Exchange

A central feature of SPEAR is support for data exchange with existing analysis synthesis packages. For example, one might wish to use SPEAR's visualization to compare the analysis results from different software. The Sound Description Interchange Format (SDIF) (Wright et al. 1999a) is supported both for import and export.



(a) Selected partials of a bell sound prior to frequency flipping.



(b) Selected partials of a bell sound after frequency flipping.

Figure 4.10: Frequency flipping.

SDIF was developed out of a need for a standard file format that could represent sounds in ways other than as a sequence of time domain samples. SDIF is particularly well-suited to storing spectral models. An SDIF file consists of a sequence of frames that are tagged with a 64-bit floating point time value (measured in seconds). Each frame is also tagged with a 32-bit integer stream ID which allows multiple logical channels per file. Each frame consists of one or more *matrices* that contain the data. Various standard matrix types, which are tagged with a 4-byte code, have been defined for different types of analysis data and are shown in table 4.1.

For SDIF import, SPEAR supports 1TRC (sinusoidal tracks), 1HRM (harmonic sinusoidal tracks), RBEP (reassigned bandwidth enhanced partials — a matrix type used by

Matrix Type	Description
1FQ0	Fundamental frequency estimates
1STF	Short time Fourier transform data. The time tag in a 1STF frame is the time of the center of the window, not the beginning.
1PIC	Picked spectral peaks with amplitude, frequency, phase, and confidence values
1TRC	Sinusoidal tracks with track index, amplitude, frequency, and phase values
1HRM	Sinusoidal tracks with track harmonic number, amplitude, frequency, and phase values
1RES	Exponentially decaying sinusoids/resonances with amplitude, frequency, decay rate, and initial phase values
1TDS	Time domain samples
RBEP	Reassigned, bandwidth enhanced sinusoidal tracks with track index, amplitude, frequency, phase, time-offset, and noise values
1ENV	Sampled spectral envelope
1NVT	Name-value table storing text or numerical header information (value) for arbitrary keys (name)
EASM	Matrix storing all breakpoints of single partial. Supported by SDIF-Edit (Bresson and Agon 2004).

Table 4.1: Some of the standard SDIF matrix types.

Loris), and 1STF (short-time Fourier transform frames). Data can be exported either as 1TRC frames or RBEP frames. In the case of 1TRC frames, resampling of the breakpoint functions must be performed to conform to the fixed frames of the 1TRC file type. For RBEP frames, each point has a time offset value. By collating all breakpoints into frame-sized chunks and properly setting the time offset, any distribution of breakpoints can be losslessly represented with an RBEP SDIF file.

Additional formats are supported for importing — the .mq and .an formats from SNDAN (Beauchamp 1993) and the .ats format from the ATS package (Pampin 2004). The data exchange options are summarized in table 4.2.

Two custom text file formats have been implemented for both import and export. `par-text-frame-format` represents resampled frames, where each line of the file con-

Format	Import	Export
SDIF 1TRC	×	×
SDIF RBEP	×	×
SDIF 1HRM	×	
SDIF 1STF	×	
SNDAN .mq	×	
SNDAN .pv	×	
ATS .ats	×	
Text frames	×	×
Text partials	×	×

Table 4.2: Supported file formats.

tains the data of a single time frame. Index numbers are used to link sinusoidal tracks. `par-text-partials-format` represents exact breakpoint functions, where each line of the file represents a complete partial. Each line is separated by a newline character. Both formats begin with a preamble. The first line of the preamble is either `par-text-frame-format` or `par-text-partials-format` depending on the type of file. The second line specifies the data types for each breakpoint. Subsequent lines give data about the number of partials and/or frames. The final line of the preamble is either `frame-data` or `partials-data` depending on the type. Figures 4.11–4.12 show the frame based format and figures 4.13–4.14 show the partials based format.

Although the SPEAR graphical interface is intuitive and fairly feature complete, it may not offer enough flexibility for more advanced uses. For example, batch analysis and transformation of many sound files would be quite tedious. The ability to import and export data gives motivated users the option to implement batch transformations with other applications (see section 5.3 for details). The motivation for creating a GUI based analysis/synthesis tool arose from the dearth of options. Existing options such as SNDAN, ATS, Loris, and IRCAM's Additive were all primarily command-line based applications. The ideal environment would offer strong graphical and scripting capabilities, and it is hoped that future versions of SPEAR will incorporate a scripting engine to facilitate a wider variety of use cases.

```
par-text-frame-format
point-type index frequency amplitude
partials-count <J>
frame-count <K>
frame-data
<frame-time0> <N> <index0> <freq0> <amp0> ... <indexN> <freqN> <ampN>
<frame-time1> <N> <index0> <freq0> <amp0> ... <indexN> <freqN> <ampN>
...
<frame-timeK> <N> <index0> <freq0> <amp0> ... <indexN> <freqN> <ampN>
```

Figure 4.11: The `par-text-frame-format` specification. Following the `frame-data` line, each line contains the breakpoints for one frame. `N` indicates the number of peaks in each frame. The `index` values connect peaks from frame to frame. Each line is separated by a newline character.

```
par-text-frame-format
point-type index frequency amplitude
partials-count 5
frame-count 7
frame-data
0.000000 5 4 79.209549 0.000271 3 199.714951 0.000551 2 350.347504 0.002547 1 540.650085 0.000262 0 708.789856 0.000477
0.010000 5 4 77.832207 0.000453 3 199.756546 0.000949 2 351.608307 0.013954 1 529.902954 0.000625 0 704.903259 0.004945
0.020000 5 4 75.407600 0.000525 3 201.211563 0.001151 2 352.697296 0.034572 1 533.938293 0.000891 0 701.399414 0.020578
0.030000 5 4 74.918587 0.000409 3 199.946182 0.001182 2 352.384399 0.051492 1 552.429321 0.001080 0 702.354309 0.042789
0.040000 5 4 75.172478 0.000174 3 198.636566 0.001109 2 352.013580 0.059633 1 564.025574 0.000642 0 704.860168 0.063039
0.050000 3 3 199.029831 0.001001 2 353.066437 0.061233 0 706.239319 0.076010
0.060000 3 3 199.642029 0.000907 2 355.612793 0.061363 0 706.990723 0.080715
```

Figure 4.12: Sample data as `par-text-frame-format`. There are a total of 5 partials and 7 frames.

```
par-text-partials-format
point-type time frequency amplitude
partials-count <J>
partials-data
<index0> <P> <start-time> <end-time>
<time0> <freq0> <amp0> ... <timeP> <freqP> <ampP>
<index1> <P> <start-time> <end-time>
<time0> <freq0> <amp0> ... <timeP> <freqP> <ampP>
...
<indexJ> <P> <start-time> <end-time>
<time0> <freq0> <amp0> ... <timeP> <freqP> <ampP>
```

Figure 4.13: The `par-text-partials-format` specification. Following the `partials-data` line, each pair of lines contains the data for one partial. The first line of each pair gives overall data for each partial: index number, length (`P`), start time, and end time. The second line gives the breakpoints. Each line is separated by a newline character.

```
par-text-partials-format
point-type time frequency amplitude
partials-count 5
partials-data
0 5 0.000000 0.046440
0.000000 540.650085 0.000262 0.011610 528.172668 0.000683 0.023220 536.151062 0.000970 0.034830 564.025574 0.001158 ...
1 7 0.000000 0.069660
0.000000 350.347504 0.002547 0.011610 351.811279 0.015791 0.023220 353.037323 0.041780 0.034830 351.919281 0.058412 ...
2 7 0.000000 0.069660
0.000000 199.714951 0.000551 0.011610 199.763245 0.001013 0.023220 201.767395 0.001204 0.034830 198.648788 0.001166 ...
3 5 0.000000 0.046440
0.000000 79.209549 0.000271 0.011610 77.610458 0.000483 0.023220 74.562180 0.000542 0.034830 75.172478 0.000314 ...
4 6 0.011610 0.069660
0.011610 704.277527 0.005664 0.023220 700.294800 0.026302 0.034830 703.821472 0.054533 0.046440 706.153931 0.073634 ...
```

Figure 4.14: Sample data as `par-text-partials-format`. There are a total of 5 partials.

5. Compositional Applications

The introductory chapter outlined a number of challenges composers face when working with the computer: the time/cost problem, the synthesis problem, the control problem, and the compositional problem. Thus far we have seen that SPEAR most directly addresses the time/cost problem (by providing fast synthesis) and the control problem (by providing an agile and highly intuitive user interface). We now turn in earnest to the composition problem. Musical tools, such as SPEAR, are never neutral agents in the compositional process. Whether deliberate or not, musical intentions are always latent in the design of computer music software systems. SPEAR was conceived with particular attention to the needs of spectral composition.

5.1 Spectral Composition

Spectral composition might be seen as subset of the more general notion of timbral composition. Many of the notable practitioners of spectral composition are quick to eschew the idea of a specific "spectral style." Nevertheless, developments over the past thirty years do point to consistencies which allow us to formulate a coherent conception of spectral music. The wariness of labels stems in part from a mistrust of compositional orthodoxies, serialism in particular, from which many spectral composers were trying to escape. A central tenet of spectral composition in fact centers around the rejection of discrete categories in favor of continuums. This distinguishes the spectral approach from other notions of timbral composition which are more oriented toward categorical and hierarchical notions: Fred Lerdahl's Timbral Hierarchies (1987), Pierre Schaefer's *Solfège de l'objet sonore*, or James Dashow's Dyad System (1999).

Spectral composition can be understood from several perspectives. From an aesthetic point of view, the notion of continuity and process is central. Tristan Murail speaks in particular about the organization of continuous frequency space:

> "Frequency space is continuous and acoustical reality only has to define its own temperaments. If we push this reasoning to an extreme, the combination of pure frequencies could be used to explain all past categories of musical discourse and all future ones. Harmony, melody, counterpoint, orchestration, etc., become outdated and are included in larger concepts." (Murail 1984)

Gérard Grisey holds a complementary view that locates notions of continuity and process in the temporal domain.

> "For me, spectral music has a temporal origin. It was necessary at a particular moment in our history to give form to the exploration of an extremely dilated time and to allow the finest degree of control for the transition from one sound to the next. ... From its beginnings, this music has been characterized by the hypnotic power of slowness and by a virtual obsession with continuity, thresholds, transience and dynamic forms." (Grisey 2000)

Joanthan Harvey's view of spectralism stresses the potentially spiritual aspect of the approach: "History seems grand, for once; spectralism is a moment of fundamental shift after which thinking about music can never be quite the same again. ... spectralism in its simplest form as color-thinking, is a spiritual breakthrough." Later he comments specifically:

> "In several works, to take a simple example, violins provide upper harmonics to a louder, lower fundamental and at a given moment they cease to fuse, begin to vibrate, begin to move with independent intervals and then again return to their previous state. The images of union and individuation are powerful ones which have both psychological and mystical implications. 'The Many and the One.'" (Harvey 1999)

It is notable that all three composers make a particular point of spectralism's place in history. Composers of the postwar generation seemed particularly keen to define spectralism as a coherent alternative to what they perceived as the cognitively opaque results of serialism. Spectral composition nevertheless gravitates toward a formalist approach that privileges individual compositional control. Technology has tended to

function as a means of gaining precision and power. It remains an interesting open area for exploration to see how spectral approaches might be applied in improvisational or indeterminate compositional situations.

From a technical point of view, spectral music engages a number of specific concerns. Pitch is organized in a continuous fashion and pitch structures are often reckoned directly in cycles per second. In instrumental writing, approximations to the nearest quarter-tone (or in some cases eighth tone) are used. This is in marked contrast to a precise just intonation pitch specification.[1] The harmonic series serves as an important point of departure, and often functions as a stable pole (if not *the* stable pole) in the harmonic discourse. Harmony often proceeds by a differential approach which unfolds processually.

The notion of "instrumental additive synthesis" often features prominently in spectral composition. A combination of instrumental sounds with the appropriate frequencies and intensities can fuse into a single timbral entity. In contrast to some approaches to timbral composition, instrumental color tends to be exploited not for its differentiating abilities but for its potential to participate in spectral fusion. Thus individual instrumental timbres are subsumed into larger musical structures.[2] In some spectral works, harmonic models are derived from characteristic spectra of instrumental sounds. The goal of instrumental additive synthesis is never to recreate, but rather to reveal latent musical potential is pre-exisiting sonic material. The archetypal gesture of instrumental additive synthesis is the arpeggiated spectra which gradually transforms from a series of individual tones to a fused complex.

As with pitch, the approach to rhythm derives from a temporal sensibility that is continuous rather than discrete. Rhythmic processes often involve controlled long range accelerations and decelerations. Additive and subtractive rhythms or rhythmic interpolations (as in the music of Philippe Hurel) are another common strategy. Overlaid rhythmic

[1] The approximation approach tends to be found in the music of the French spectralists and their followers. American music that might be identified as spectral or proto-spectral has tended to follow a just intonation approach (as in the compositions of James Tenney or Ben Johnston).

[2] This technique is a hallmark of spectral music. While instrumental timbre played an important role in some proto-spectral music (that of Varèse in particular) timbre tended to be more about making elements more distinct (Varèse's planes of sound) rather than fusing them together.

pulsations with various rational relationships are also encountered (particularly in the music of Grisey). In many cases rhythms are approximated, either via quantization or proportional notation.

A complete summary of all technical aspects found in spectral music is beyond the scope of this document (see Fineberg (2000) for more details). For our discussion, the following list of some of the central concerns of spectral music will suffice:

- continuous frequency space
- continuous temporal space
- timbre-harmony continuum
- the use of approximations
- global approach to form
- smooth transitions contrasted with ruptures
- process preferred over development
- appeal to perceptual and psychoacoustic aspects of sound

The last item points to a particularly anti-formalist aspect of spectral music. For all its technical machinations, spectral music is very much "for the listener." However, this tendency to appeal to cognitive "universals" raises interesting concerns. For example, to what degree might the expressive or communicative ability of the spectral approach be limited by this focus? In particular, does the pre-occupation with the purely acoustic preclude the possibility of addressing culture-specific aspects of the musical enterprise? These open questions point to interesting possible directions for future compositional work.

Technology has always played an important role in the development of spectral music. Models derived from electronic processes such as frequency shifting, ring modulation, frequency modulation, or tape delay can be found in any number of spectral pieces. The desire for mastery of the sonic continuum led naturally to the use of the digital computer.[3] Digital sound synthesis is complementary to instrumental synthesis and

[3]Ironically, one gains reproducible control of the continuum in the discrete world of the computer.

spectral works that combine electronic and instrumental sounds can achieve remarkable coherence. Computer-assisted composition environments, which will be discussed in section 5.3, are indispensable for realizing sonic processes. SPEAR is a particularly useful tool for spectral composition. The high performance GUI makes it possible to visualize and manipulate complex sounds. Its detailed analyses can be used in the derivation of harmonic and/or rhythmic models, or the data can be harnessed directly to drive various forms of digital synthesis.

5.2 Sonic Transformations

A sinusoidal analysis may be viewed as a reservoir of sonic data with almost limitless musical potential. Possible temporal transformations include replication, time delay, time dilatation, and time reversal. Amplitude envelopes may be applied (for example, to fade partials in and out, or to emphasize the attack portion of a sound). Partial frequencies may be altered by any desired transposition or frequency shifting envelope. The analysis data can be used to drive other compositional or synthesis processes (for example, granular synthesis). The following subsections detail a number of possible sonic transformations.

5.2.1 Timbre Hybridization

Properties of different spectral analyses can be combined to create new hybrids. This process of timbre hybridization is sometimes referred to as *cross synthesis*. There are many possible cross synthesis implementations, some of which are found in software such as AudioSculpt, SoundHack, Shapee, and in commercial plugins such as Prosoniq Morph. Many of these techniques focus on different ways of combining STFT frames or applying properties of one STFT frame to another. Sinusoidal modeling offers the possibility of very fine-grained control over the cross synthesis process.

Dynamic morphing between two collections of partials A and B can be achieved by matching common partials and interpolating their frequencies and amplitudes. Such a procedure has been implemented as in Common LISP as follows:

1. Temporally align the two sounds by means of timebase envelopes that map output times to time points in the original sound.
2. Define an interpolation envelope that maps output time to relative fraction of A or B that should be present in the output — a value of 0 indicates the sound should consist of all A, 1 indicates all B, intermediate values should be a blend of A and B.
3. Compute a succession of blended spectral frames to be synthesized. This proceeds as follows:
    - Advance an output time variable t by a predetermined hop.
    - For each value of t, determine the active partials in A and B and compute two frames A_t and B_t with the interpolated frequencies and amplitudes of the active partials at time t.
    - For each value of t, determine the current blend factor x from the interpolation envelope.
    - Determine a set of matching frequency/amplitude points in A_t and B_t. Matching is controlled by a frequency tolerance. Interpolate the amplitude and frequency of each of the matching points according to the blend factor x. Unmatched points can be discarded or have their amplitude adjusted according to x (thereby creating a cross fade of the unmatched partials).
4. Synthesize the blended frame and window with a tapered window function. Overlap add into the output.

This morphing algorithm is somewhat crude, but is reasonably effective for certain kinds of sounds. It could be improved by computing a new set of blended partials, rather than overlap adding phase-incoherent frames in a granular fashion. As it is, the algorithm only works well if sounds contain similar frequency content. Harmonic sounds need to share the same fundamental frequency. A more sophisticated algorithm, such as that used in Loris, would tag the harmonics of A and B relative to a fundamental frequency track. Like harmonics can then be interpolated and smoothly glide to new frequencies. For the specifics of the Loris morphing algorithms see Fitz et al. (2003).

A more sophisticated algorithm should also be able to interpolate between more than two sources. It would be musically useful to create long sequences of seamless morphs between a series of sounds as with IRCAM's Diphone Studio. Despite its limitations, this basic algorithm shows promise and has been used to create some effective morphs between harmonic and inharmonic timbres.

Another approach to cross synthesis is to keep the frequency content fixed and manipulate amplitudes to affect a change in the spectral envelope (see section 2.3 for a brief discussion of spectral envelopes). A hybrid can be created by imposing the spectral envelope of one sound on another. Although SPEAR does not currently include spectral envelope capabilities, a number of Common LISP functions have been developed for manipulating spectral envelopes and applying them to SPEAR analyses (see table 5.3). The functions are designed to work with spectral envelope data generated by SuperVP, a powerful command line analysis/synthesis tool developed at IRCAM. Successful experiments have been conducted with the application of vocal spectral envelopes to tam-tam sounds (and visa-versa).

5.2.2 Spectral Tuning

Spectral tuning adjusts the frequency content of a sound to match the frequencies of a predetermined spectrum or harmonic field. The target frequencies might be derived from any number of sources: octave repeating scales or chords, synthetic spectra, frequencies derived from other spectral analyses, distortion and warping of existing spectra, etc. Spectral tuning using the phase vocoder has been explored extensively by a number of composers and researchers including Eric Lyon and Christopher Penrose with FFTease (Lyon 2004), Paul Koonce with PVC, and Trevor Wishart with Composers Desktop Project (CDP) (Wishart 2000).

Spectral tuning with a sinusoidal model allows very precise control of the evolution of each partial. In a typical application, the tuning is static: the local amplitude and pitch fluctuations of each partial are maintained and the partial is transposed so that its *average*

frequency matches the target. Local pitch fluctuations can be attenuated or emphasized as desired. Dynamic tuning transposes a partial according to a variable function. If the rate of frequency change is not too rapid, the retuning can still maintain local frequency perturbations which are crucial to the character of most natural sounds. A simple example of dynamic tuning would be the application of vibrato or glissando.

Parameters can control the strength of retuning. For example if a partial is separated by a distance of 3 semitones from its target, a retuning strength of 70% would transpose the target by 2.153 semitones rather than the full 3 semitones. Or a parameter can specify that partials are retuned only if they are in close enough proximity to the target. Particularly interesting results have been achieved with progressively retuning inharmonic timbres so that a succession of sounds eventually converges on a harmonic spectrum. Spectral tuning can also be used for cross synthesis where the target frequencies are derived from analysis of a different sound.

5.3 Software Composition Environments

Although a great deal of useful sonic manipulation and transformation can be accomplished directly with the SPEAR GUI, much of the software's compositional utility is revealed when it is used in conjunction with other tools. Computer-assisted composition and performance environments such as OpenMusic, Common Music, Max/MSP, RTCmix, and SuperCollider (to name a few), can be used to import, manipulate, export, and/or synthesize SPEAR analysis data.

SPEAR's support for text based file formats (as well as standard SDIF formats) makes cross program data exchange relatively straightforward. The text formats are particularly useful because they are human readable and easy to use with interactive interpreted programming languages such as Java, Python, Scheme, and Common LISP. Import and export of the SDIF formats is more difficult and time consuming to implement for casual programmers. The author's own experience has shown that even though SDIF is robust and well-defined, it is not necessarily widely understood among computer music practitioners.

Even with supporting SDIF libraries such as those provided by CNMAT or IRCAM, it may prove more expedient (particularly in one-off applications) to use the text data formats.

5.3.1 OpenMusic

OpenMusic is a computer-assisted composition environment developed at IRCAM using the Common LISP language. It is a visual programming environment based on patching of pre-defined or user-defined objects. The patch paradigm provides full access to Common LISP programming, including iteration and recursion. Nodes in the object graph are evaluated for their results and possible side-effects. A user request for evaluation will trigger evaluation of all parent nodes (figure 5.1). OpenMusic includes graphical editors for MIDI data, breakpoint functions, and common practice notation (figure 5.2). OpenMusic was developed as a successor to PatchWork a similar LISP based environment for visual programming.

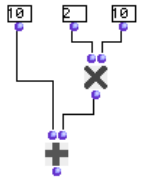


Figure 5.1: OpenMusic patch that evaluates the arithmetic expression $10 + (2 \times 10)$.

The environment may be extended with user defined libraries. New OpenMusic function are defined as CLOS (Common LISP Object System) methods using the `om:defmethod!` macro. SPEAR import and export has been integrated with *OMTristan*, a custom OM library created by Tristan Murail. *OMTristan*, which has evolved over decades, includes a wide variety of compositional algorithms for the generation of chords and spectra, temporal computations, MIDI communication, and general data manipulation. The `spdata` object in *OMTristan* is specifically designed for efficient storage and manipulation of frame based spectral data. The methods `spear-read` and `spear-write` convert between the SPEAR `par-text-frame-format` file type and OM `spdata` objects.

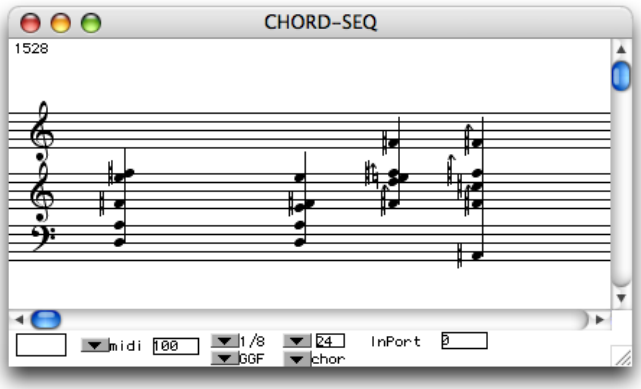


Figure 5.2: OpenMusic notation editor displaying a chord sequence.

Figure 5.3 shows an OM patch that reads SPEAR data, transforms the data, displays the transformed data in common practice notation, and writes out the transformed data to a new text file. The patch functions as follows: The `spear-read` object (located at the top) takes the name of a text file as input and outputs an `spdata` object which is stored as an embedded object (this avoids having to re-read the data from disk on subsequent evaluations). The `spdata` output is split into three streams: the frequencies, the amplitudes, and the partial index numbers. Each stream is a LISP list of two elements. The first element is a list of times for each frame and the second element is a list of lists where each sublist is the frame data (either frequencies, amplitudes, or indices). The time base is stretched (the "dilation" parameter) and the frequencies distorted in the `dist-frq` sub-patch. The data is reassembled into frames and converted to a chord sequence in the `visu-chseq` sub-patch. The times, indices, frequencies, and amplitudes are also routed to the `spear-write` object which outputs a new text file with the transformed data. The data can then be opened in SPEAR for resynthesis or further transformation.

5.3.2 Max/MSP

Max/MSP is a widely used graphical patching environment for realtime media processing. SPEAR SDIF data can be imported into Max/MSP using the CNMAT SDIF objects (Wright et al. 1999b). The `SDIF-buffer` stores the SDIF data in a named buffer which can be accessed using the `SDIF-tuples` object. SDIF-tuples outputs matrix data at the specified

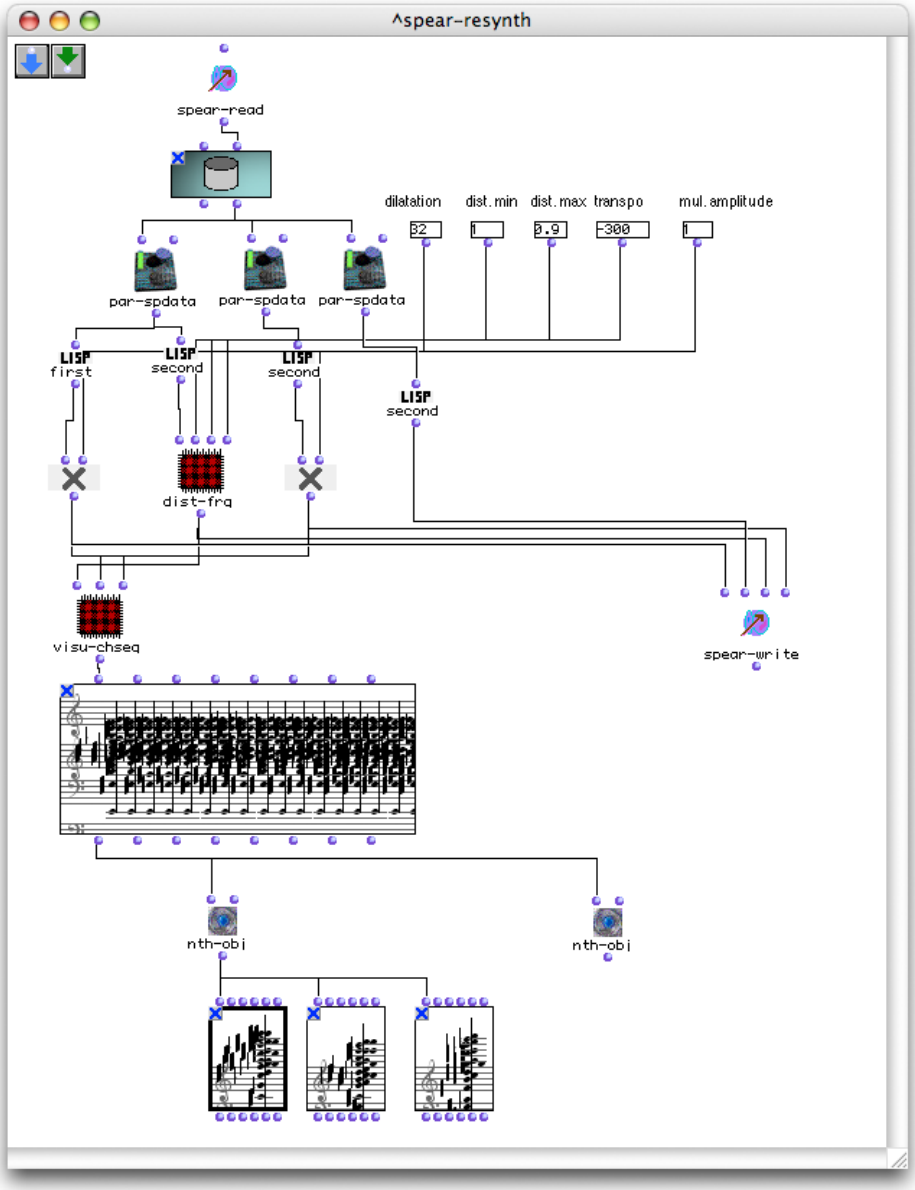


Figure 5.3: OpenMusic patch that reads and writes SPEAR data.

time (with optional interpolation) as a Max list. CNMAT's `sinusoids~` object implements an oscillator bank (with optional bandwidth enhancement as described in section 3.6.1) that is driven by a list of frequency amplitude pairs. The `threefates` object can be used to manage the sinusoidal track indices output from the SDIF buffer and format the data into frequency amplitude pairs suitable for the `sinusoids~` object.

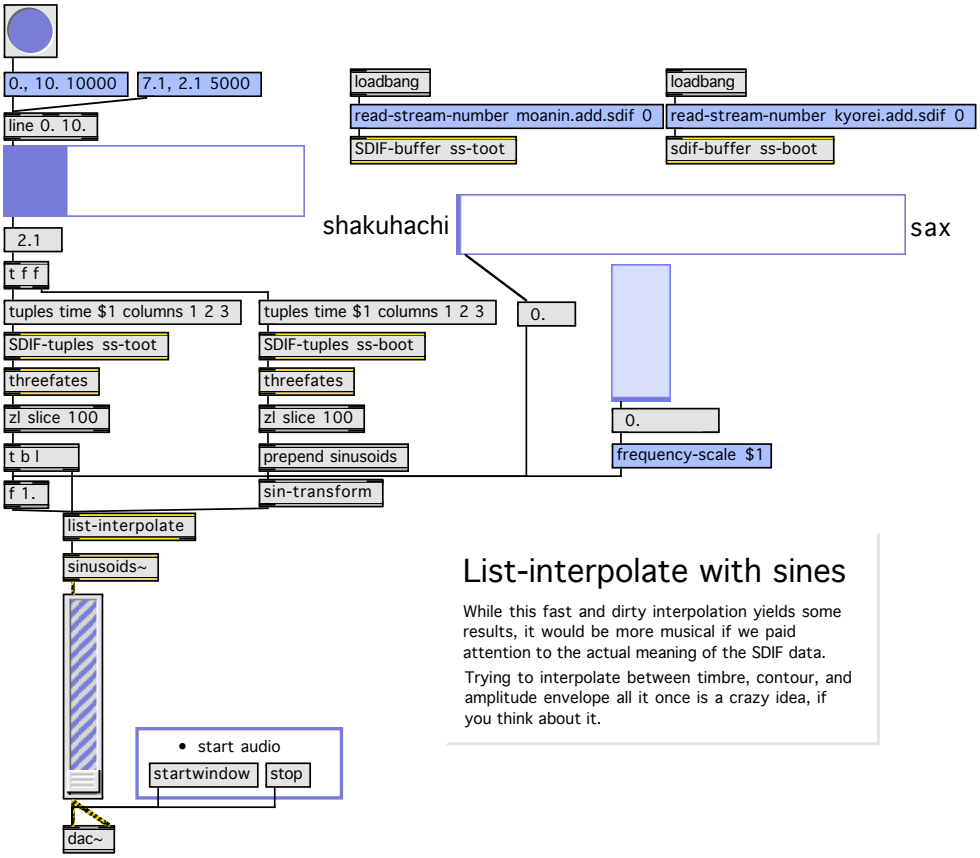


Figure 5.4: Max/MSP patch from the CNMAT Spectral Synthesis Tutorials that interpolates between two sinusoidal models to achieve a dynamically controllable cross synthesis.

The limitation of 256 elements with the standard Max list objects does put some constraints on the complexity of the spectral data. Michael Zbyszynski's "Spectral Synthesis Tutorials" suggest some solutions (Zbyszynski et al. 2007). Possibilities include the use of custom C or Java objects, the use of Javascript, the use of Jitter matrix processing objects, or the use of IRCAM's FTM extensions.

The use of sinusoidal models in the Max/MSP/Jitter environment offers numerous options for interactive transformation and resynthesis. The ability to manipulate spectral data directly in Max using Java and Javascript eases the burden of managing large spectral data sets. Future work could lead to the development of "partial based" storage models rather than the frame based approach implicit in the SDIF 1TRC format. As we shall see in the next section, the partials based format (collections of breakpoint functions) is particularly well suited for certain types of musical operations.

5.3.3 Common Music

Common Music (CM) is an algorithmic composition environment developed by Heinrich Taube. Like OpenMusic, CM is implemented in Common LISP. It is built around the notion of generalized input/output streams which can include MIDI files, MIDI ports, Open Sound Control streams, CSound scores, CLM notelists, and CMN scores. Compositional facilities include the scheduled processes, a full featured pattern iteration system, and a flexible graphical interface called *Plotter*. *Plotter* facilitates display and editing of breakpoint functions, histograms, scatter plots, and piano roll style MIDI data (Taube 2005). CM can be tightly integrated with CLM (Common Lisp Music), a Music-N style acoustic compiler.

Current versions of CM include a spectral composition toolkit than can read SPEAR data from `par-text-frame-format` and `par-text-partials-format` files. The data is stored as nested LISP lists. For manipulating large sounds, it can be more efficient to store spectral data in LISP vectors (contiguous arrays). A LISP toolkit has been developed for reading, writing, and manipulating `par-text-partials-format` data. A complete sound is stored as a LISP list of `sp-partial` data structures. Each `sp-partial` represents a single breakpoint envelope with time, frequency, and amplitude arrays. Functions exist to transpose, time warp, and amplitude scale partials. The adjustments can be controlled dynamically with envelopes. The function for transforming single partials are summarized in table 5.1, page 91. Many of the transformation functions take an optional *`copy`* parameter which, when

(**copy-partial** *partial*) Return a newly allocated copy of *partial*.
(**offset-partial** *partial offset* [*copy*]) Shift the time of *partial* by *offset* seconds.
(**transpose-partial** *partial interval-ratio* [*copy*]) Transpose the frequencies of *partial* by *interval-ratio*.
(**transpose-partial-env** *partial interval-ratio strength-env* [*copy*]) Transpose the frequencies of *partial* by a ratio ranging from 1 to *interval-ratio*. *strength-env* is an envelope controlling the percentage of *interval-ratio* applied.
(**time-scale-partial** *partial timescale* [*copy*]) Apply a fixed time scaling ratio *timescale* to *partial*.
(**amplitude-scale-partial** partial ampscale [*copy*]) Apply a fixed amplitude scaling ratio *ampscale* to *partial*.
(**reverse-partial** partial [*copy*]) Reverse the temporal evolution of *partial*.
(**amp-env-partial** partial env [*copy*]) Apply an amplitude scaling envelope *env* to *partial*.
(**sp-partial-avg-freq** *partial*) Compute the average frequency of *partial*.
(**sp-partial-avg-amp** partial [*points*]) Compute the average amplitude of *partial*. If *points* is non-nil, the average is computed based on the first *points* breakpoints. If *points* is nil (the default) all breakpoints are used to compute the average.
(**sp-partial-interp** *partial time*) Interpolates the frequency and amplitude of *partial* at *time* seconds. The frequency and amplitude are returned as multiple values (in that order). Uses a binary search for efficiency.

Table 5.1: Transformation functions applied to individual partials.

non-nil, applies the transformation to a new copy of partial, leaving the original unchanged. The default value of *copy* is nil, meaning the transformation is applied destructively.

A `retune-partials` function can transpose a collection of partials to a new set of target frequencies. The `retune-partials` function assumes partials with relatively constant frequency (the retuning is based on the average frequency of the original partial). Functions

(**retune-partials** *partials target* {*keyword value*}∗)

Transpose each of the partials of *partials* so that its average frequency matches a frequency contained in the list *target*. Because the target frequency set may cover a vastly different frequency range from the original partials, the average partial frequencies are first rescaled to lie within the minimum and maximum frequencies of *target*. Each rescaled frequency is then matched to the closest frequency in *target*. The matching is controlled by the following keyword arguments:

`:low-limit`	Only tune partials above this frequncy. Default is 0 Hz.
`:high-limit`	Only tune partials below this frequcny. Default is 3000 Hz.
`:delay-curve`	Envelope to apply a frequency dependent delay to each partial. Default is nil.
`:amp-curve`	Envelope to apply frequency dependent amplitude scaling to each partial. Default is nil.
`:flatten`	Scale the ambitus of each partial. Values < 1.0 reduce the amount of frequency variation of each partial. When 0.0, all frequencies are set to the average frequency.
`:strength`	Adjust the amount of frequency remapping. When 1.0, the partials are retuned to exactly match the target frequencies. When 0.0, the partials remain at their original frequencies.
`:strength-curve`	Adjust the amount of frequency remapping over time. Partials can be made to glide between original and target frequencies.
`:index-scramble`	Scramble the targets such that the original partials will remap to less proximate target frequencies. When 0, there is no scrambling (the default). When 1, remap to the neighbor of the closest target frequency. When *n*, remap to the *n*th neighbor (either above or below).

(**sp-partials-apply-vibrato** *partials* {*keyword value*}∗)

Apply slow frequency modulation (vibrato) to *partials* (a list of partials). This function requires CLM. The following keyword arguments are supported:

`:vibfreq`	The frequency of the vibrato in hertz. Default is 6.5.
`:vibinterval`	The width of the vibrato in semitones. Default is 0.5.
`:start-time`	Starting time for the onset of vibrato (in seconds). Default is 0.
`:end-time`	Ending time for the vibrato (in seconds) or nil if vibrato continues to the end. Default is nil.
`:copy`	If non-nil, return new copies of the partials leaving the originals unchanged. Default is true.

Table 5.2: Frequency adjustment functions applied to collections of partials.

for frequency transformation are summarized in table 5.2, page 92. Spectral envelope functions are show in table 5.3, page 93.

5.4 Compositional Examples

A straightforward application of SPEAR is the derivation of harmonic models for instrumental composition. Pitches can be transcribed manually from the GUI or by importing

(**sp-partials-flatten-spectral-env** *partials spectral-env* [*amp-scaler*])

Flatten the spectral envelope of the sound represented by *partials* by dividing the partial amplitudes by the spectral envelope *spectral-env*. The spectral envelope can be generated by reading an SVP text formatted spectral envelope analysis. Apply an optional amplitude scaler *amp-scaler* (default value 0.025) following the division.

(**sp-partials-apply-spectral-env** *partials old-spectral-env new-spectral-env* {*keyword value*}∗)

Apply a new spectral envelope *new-spectral-env* to the sound represented by *partials*. *old-spectral-envelope* is the current spectral envelope of the sound. The following keyword arguments are supported:

:transpose	Transposition ratio.
:time-warp	Time remapping envelope.
:freq-warp	Frequency remapping envelope.
:freq-func	Generalized frequency transformation. *freq-func* is a function of three arguments: *frequency*, *partial-index*, and *time*. The function should return the new desired frequency value.
:filter-func	Generalized amplitude transformation. *filter-func* is a function of four arguments: *amplitude*, *frequency*, *partial-index*, and *time*. The function should return the new desired amplitude value.
:copy	If non-nil, return new copies of the partials leaving the originals unchanged. Default is true.

Table 5.3: Spectral envelope functions applied to collections of partials.

into other environments. Rather than relying on individual spectral snapshots, the evolution of a collection of partials can be incorporated into the composition model.

In the author's work *Subterrain* (2008) for B♭ clarinet, string trio, and electronics, clarinet multiphonics were analyzed. The SPEAR GUI was indispensable for extracting only the most stable and/or musically interesting partials from these complex sounds. The data was imported into Common Music and amplitudes were averaged across entire partials, which gave a very good sense of the overall spectral color of the multiphonic. Further manipulation of the multiphonic spectra, including frequency shifting and ring modulation, were used to create harmonies and pitch reservoirs.

Spectral retuning using SPEAR and OpenMusic was used to create a number of sounds in Tristan Murail's *Pour adoucir le cours du temps* (2005) for 18 instruments and synthesized sound. In this piece, the partials of tam-tam sounds were retuned to the frequencies of artificially generated spectra. A cow-bell analysis also undergoes similar retuning. Of the work, Murail writes,

> "The 'noises' (breath, grainy sounds, metallic resonances) of the piece have been domesticated, tuned to instrumental harmonies through the use of a specific technique for altering the internal components of the sounds. This is how the sound of a gong can become a harmony, or the virtual cow-bells can be constantly varied and altered by their musical context. The synthesized sounds are triggered from an onstage MIDI keyboard — they are mixed and spatialized in real-time by the program Holophon developed by the GMEM." (Murail 2005)

Spectral tuning was used to generate several of the sounds used in the author's work *Tear of the Clouds* (2007) (see appendix A for the complete score). The low vocal sounds (at m. 14, reh. A and m. 203) were created by retuning and resynthesizing samples of a choral ensemble. Although these particular source samples exhibited a high standard of performance and excellent recording quality, they were not ideally suited for resynthesis. Because they consisted of a sum of many ensemble voices (all in unison pitch), and because they were recorded in a reverberant environment, partial tracking resulted in a large number of short low amplitude partials in addition to the fundamental and harmonics. For retuning purposes, a cleaner representation was desired. IRCAM's Additive program was used to perform fundamental frequency tracking and to analyze a fixed set of harmonic partials (up to 22 kHz). The SDIF 1TRC file from Additive was imported into SPEAR and then exported as a text file. This was then imported into LISP where the retuning was performed.

At m. 14, the retuning target consists of selected partials of a compressed harmonic spectrum built on G1 (49 Hz). The tuning was only applied to the original partials ranging from 49 Hz to 1850 Hz. Because the original sound had a fundamental of 82 Hz, retuning and compressing the entire spectrum would have resulted in a considerable loss of high frequency presence. Since most of the harmonic "color" of a sound is found below 2000 Hz, the harmonics above 1850 Hz were retained in their original tuning (another option would be to replicate partials to fill in the missing upper harmonics). The retuned data was then imported into SPEAR and synthesized. Further post processing, including granular time stretching, was then applied. The granulation produced a sound of the required

duration and helped re-impart some of the reverberant quality of the original sound that was lost in the analysis.

Further applications of spectral tuning in *Tear of the Clouds* include the retuning of tubular bell, piano, and tam-tam sounds. The penultimate electronic sound of the piece is built from a retuning of a piano sound. The target spectrum consists of a distorted harmonic series built on D♯1 (39 Hz), in which the even partials are stretched and the odd partials are unchanged. For this particular sound, the retuning is dynamic. The attack maintains the original tuning and the partials glide to the target spectrum over a period of 2.75 seconds. Rather than sliding to the nearest target frequency, the retuning was configured so that the partials make wider frequency excursions (either up or down), crossing each other as they proceed.

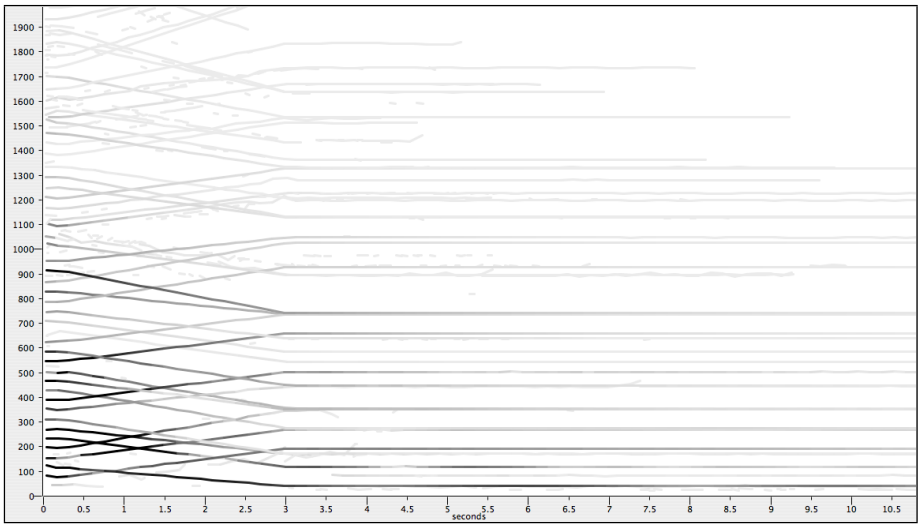


Figure 5.5: Dynamic retuning with crossing partials.

5.5 Performance Models

Although not extensively explored in the current work, some words are in order about the potential applications in performance and improvisation. The SPEAR interface itself could be viewed as a crude performance interface where one can scrub through the sound using the mouse (Garton 2007). One can easily imagine a richer instrument that would give

control of elements such as playback speed, transposition, frequency shifting, harmonic parity (balance of even and odd partials), etc. These potentials would ideally be realized in a flexible environment such as Max/MSP, rather than in the more confining space of the SPEAR user interface. A variety of gestural and tactile inputs could be used to drive resynthesis parameters.

6. Conclusions

The preceding chapters have given a thorough account of the technical aspects of the design and development of SPEAR while also touching on some possible compositional applications. I do not wish to claim that SPEAR represents a revolutionary new computer music tool. It has been preceded by more than fifteen years of research and development in the field of spectral analysis and modeling. However, I hope it is clear that SPEAR is an important evolutionary convergence of a number of mature technologies.

Although SPEAR has experienced reasonable popularity among a wide variety of technologically oriented music practitioners, the software is still very much a work-in-progress. Although the interface works well, it is still limited in many ways. A multi-demensional display should be developed that allows one to view and edit all aspects of a partial's evolution: amplitude, noise bandwidth, phase, stereo (or multi-channel) panning, etc. The user should be able to draw breakpoint envelopes and curves to control the transformation of any parameter. Rule based selection (see section 4.2.1) should be implemented. Moreover, an interface that allows one to specify *persistent selections* (like audio regions in a waveform editor) would be incredibly useful. A user interface for sound morphing and hybridization would also be welcome. Since it is impossible to envision all of the possible desired editing and transformation possibilities, SPEAR needs a built-in scripting language and command-line tools that allow users to develop their own processes.

SPEAR must also keep pace with developments in spectral analysis and synthesis. To build more compact and malleable spectral models, SPEAR needs robust fundamental frequency tracking. The option should be available to perform analyses with fixed numbers

of harmonics (as with IRCAM's Additive and Diphone programs). Models with fixed numbers of partials are much easier to manipulate in real-time environments such as Max/MSP or SuperCollider.

Some form of noise modeling, as discussed in section 3.6, should be implemented. The transient sharpening method described in section 3.5 is successful to a point, but to handle wide-band polyphonic audio and percussive attacks, a separate transient detection and representation method is needed. The multi-resolution sinusoidal modeling techniques described in Levine (1998) should also be implemented. Although multi-resolution modeling significantly increases the implementation complexity of partial tracking (particularly at the band boundaries), it has the potential to considerably reduce pre- and post-echo artifacts. The combination of time-reassignment, multi-resolution analysis, noise modeling, and improved partial tracking would represent a formidable analysis engine.

In order to tie together these more sophisticated analysis techniques, a more advanced user interface is required. Each component of the analysis would be assigned to its own layer:

- fundamental frequency layer
- sinusoids layer
- noise layer
- transient layer
- spectral envelope layer

Each of these layers could be (optionally) coupled to the others. For example, when layers are coupled together, adjustments to the fundamental frequency track would correspondingly adjust the frequencies in the sinusoids layer. As the partials are transposed, the amplitudes could be adjusted to match the spectral envelope layer. Peak detection and tracking could be applied to the spectral envelope data so that formant trajectories and widths could be adjusted in a manner analogous to individual partials. Given sufficient time and effort, all of these ideas are feasible.

Returning to some of the issues raised in the opening chapter, we must consider what a tool like SPEAR tells us about timbre and musical organization. For all its abilities, SPEAR perhaps raises more questions than it answers. Although obvious, it's worth stating the overall conclusion: additive synthesis is not a sufficient model for timbre. Although additive models are malleable, they tend to be fragile — minor adjustments to a few partials can destroy timbral fusion and coherence. Higher level models are needed. Spectral envelopes are a good first step. We should also look for models of the micro-level evolution of partials. The amplitude and frequency fluctuations can be correlated between partials and with parameters of the overall evolution of the sound. Finally, we must recognize that the time varying aspects of spectra are often far more important than the instantaneous relationships. We need timbral models than can capture micro-level rhythm, grainy textures, and impulses. The TAPESTREA software, which uses a sinusoids + stochastic background + transients model, suggests a particularly promising approach to high level timbre modeling (Misra et al. 2006).

From a compositional perspective there is clearly more to learn about timbre and tools such as SPEAR can help with these compositional approaches. But it is not enough for compositional work to merely reflect or reinforce habitual modes of understanding timbre. Experimental composition can reveal new ways to listen to timbre, previously unheard subtleties, surprising relationships, and unanticipated modes of organization. With the acknowledgment that speculations on the future are often more about setting an agenda than trying to make a neutral prediction, we might ask what role computer music might play in this process and in turn what will be the nature of tools to come? Or more precisely, what do we want from our musical tools of the future? Successful software tends to embody at least some of the following attributes:

1. ease of use and configuration
2. high performance
3. large feature set
4. reliability

5. backward compatibility
6. initial novelty
7. openness and extensibility

At present, commercial software tends to excel at items 1–3, while failing miserably in terms of openness and user extensibility. Extensibility has typically been the strength of software with an experimental and open source pedigree. Commercial software is frustrating because it tends to be expensive and creatively limiting. Free software is frustrating because it tends to be difficult to configure and maintain. Of what benefit is extensibility if one is left with no time to exploit it? Future software efforts must bridge this gap. We must avoid the "sealed black boxes" of the commercial world and the arcane configuration, instability, and continual tabula rasa upheavals of the open source world.

Negative proscriptions aside, what potentials are there for music software of the future? An environment that melds a powerful acoustic compiler and general purpose programming language with a flexible and high-performance GUI seems ideal. Such an environment should include flexible time-based displays that can handle common practice notation, breakpoint functions, audio data, and spectral models. It must not be a neutral environment — it must make assumptions. It must assume a user who wishes to work with common practice diatonic music and quantized meter and rhythm; it must assume a user who wishes to work with millions of sonic granules on a proportional time scale; it must assume a user who wishes to precisely notate irrational rhythms and microtonal inflections; it must assume a user who wishes to capture and manipulate performance data in real time. Technologically, this is all within the realm of possibility today. It remains to be seen when, if, and/or how this may come about. Tools of this sort are not envisioned to make composition facile. They should exist to engage and inspire the human imagination and creative spirit.

References

Abe, Mototsugu, and Julius O. Smith. "Design Criteria for Simple Sinusoidal Parameter Estimation based on Quadratic Interpolation of FFT Magnitude Peaks." In *Proceedings of the 117th Audio Engineering Society Convention*. AES, 2004.

Amatriain, Xavier, Maarten de Boer, Enrique Robledo, and David Garcia. "CLAM: An OO Framework for Developing Audio and Music Applications." In *Proceedings of 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications*. Seattle, WA, 2002. http://mtg.upf.edu/publicacions.php.

Auger, F., and P. Flandrin. "Improving the Readability of Time-Frequency and Time-Scale Representations by the Reassignment Method." *IEEE Transactions on Signal Processing* 43: (1995) 1068–1089.

Beauchamp, James W. "Unix Workstation Software for Analysis, Graphics, Modification, and Synthesis of Musical Sounds." In *Proceedings of the 94th Audio Engineering Society Convention*. AES, 1993. Preprint no. 3479.

Bonada, Jordi, and Xavier Serra. "Synthesis of the Singing Voice by Performance Sampling and Spectral Models." *IEEE Signal Processing Magazine* 24, 2: (2007) 67–79.

Bresson, Jean, and Carlos Agon. "SDIF Sound Description Data Representation and Manipulation in Computer Assisted Composition." In *Proceedings of the International Computer Music Conference*. Miami, FL: ICMC, 2004, 520–527.

Cadoz, Claude, Annie Luciani, and Jean Loup Florens. "CORDIS-ANIMA: A Modeling and Simulation System for Sound and Image Synthesis: The General Formalism." *Computer Music Journal* 17, 1: (1993) 19–29.

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

Dashow, James. "The Dyad System (Parts Two and Three)." *Perspectives of New Music* 37, 2: (1999) 189–230. http://www.jamesdashow.net/download/DyadSystemPt2-3.pdf.

Depalle, Philippe, Guillermo García, and Xavier Rodet. "Tracking of Partials for Additive Sound Synthesis Using Hidden Markov Models." In *IEEE International Conference on Acoustics, Speech and Signal Processing*. Minneapolis, MN, 1993, 225–228.

Fineberg, Joshua. "Guide to the basic concepts and techniques of spectral music." *Contemporary Music Review* 19, 2: (2000) 81–113.

Fitz, Kelly. *The Reassigned Bandwidth-Enhanced Model of Additive Synthesis*. Ph.D. thesis, University of Illinois Urbana-Champaign, Urbana, IL, 1999.

Fitz, Kelly, Lippold Haken, and Brian Holloway. "Lemur—A Tool for Timbre Manipulation." In *Proceedings of the International Computer Music Conference*. Banff, Canada, 1995, 158–161.

Fitz, Kelly, Lippold Haken, Susanne Lefvert, Corbin Champion, and Mike O'Donnell. "Cell-utes and Flutter-Tongued Cats: Sound Morphing Using Loris and the Reassigned Bandwidth-Enhanced Model." *The Computer Music Journal* 27, 4: (2003) 44–65.

Fitz, Kelly, and Lippopld Haken. "On the Use of Time-Frequency Reassignment in Additive Sound Modeling." *Journal of the Audio Engineering Society* 50, 11: (2002) 879–892.

Garton, Brad. "Multi-language Max/MSP.", 2007, Accessed August 27, 2008. http://www.cycling74.com/story/2007/3/26/145938/572.

Grey, John M. "Multidimensional perceptual scaling of musical timbres." *Journal of the Acoustical Society of America* 61, 5: (1977) 1270–1277.

Grisey, Gérard. "Did You Say Spectral?" *Contemporary Music Review* 19, 3: (2000) 1–39.

Harris, Fredric J. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform." *Proceedings of the IEEE* 66, 1: (1978) 51–83.

Harvey, Jonathan. *In Quest of Spirit: Thoughts on Music*. Berkeley, CA: University of California Press, 1999.

Lagrange, Mathieu, Sylvain Marchand, Martin Raspaud, and Jean-Bernard Rault. "Enhanced Partial Tracking Using Linear Prediction." In *Proceedings of the 6th International Conference on Digital Audio Effects*. London, UK: DAFx-03, 2003, 402–405.

Lagrange, Mathieu, Sylvain Marchand, and Jean-Bernard Rault. "Partial Tracking based on Future Trajectories Exploration." In *Proceedings of the 116th Audio Engineering Society Convention*. AES, 2004.

———. "Enhancing the Tracking of Partials for the Modeling of Polyphonic Sounds." *IEEE Transactions on Audio, Speech, and Signal Processing* 15, 5: (2007) 1625–1634.

Laroche, Jean. "Synthesis of Sinusoids via Non-Overlapping Inverse Fourier Transform." *IEEE Transactions on Speech and Audio Processing* 8, 4: (2000) 471–477.

Laroche, Jean, and Mark Dolson. "Phase-vocoder: About this phasiness business." In *IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 1997, 19–22.

———. "New Phase-Vocoder Techniques for Pitch-Shifting, Harmonizing and Other Exotic Effects." In *IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 1999, 91–94.

Lerdahl, Fred. "Timbral Hierarchies." *Contemporary Music Review* 1, 2: (1987) 135–160.

Levine, Scott N. *Audio Representations for Data Compression and Compressed Domain Processing*. Ph.D. thesis, Stanford University, Stanford, CA, 1998.

Lindemann, Eric. "Music Synthesis with Reconstructive Phrase Modeling." *IEEE Signal Processing Magazine* 24, 2: (2007) 80–91.

Loy, Gareth. "Composing with Computers—a Survery of Some Compositional Formalisms and Music Programming Languages." In *Current Directions in Computer Music Research*, edited by Max V. Matthews, and John R. Pierce, Cambridge, MA: The MIT Press, 1989, chapter 21, 291–396.

Lyon, Eric. "Spectral Tuning." In *Proceedings of the International Computer Music Conference*. Miami, FL: ICMC, 2004, 375–377.

Maher, Robert C. *An Approach for the Separation of Voices in Composite Musical Signals*. Ph.D. thesis, University of Illinois, Urbana, IL, 1989.

Matthews, Max V. "The Digital Computer as a Musical Instrument." *Science* 142, 3592: (1963) 553–557.

McAulay, Robert J., and Thomas F. Quatieri. "Speech Analysis/Synthesis Based on A Sinusoidal Representation." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 34, 4: (1986) 744–754.

Misra, Ananya, Perry R. Cook, and Ge Wang. "Musical Tapestry: Re-composing Natural Sounds." In *Proceedings of the International Computer Music Conference*. ICMC, 2006.

Murail, Tristan. "Spectra and Pixies." *Contemporary Music Review* 1, 1: (1984) 157–170.

———. *Pour adoucir le cours du temps*. Editions Henry Lemoine, 2005.

Nuttall, Albert H. "Some Windows with Very Good Sidelobe Behavior." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29, 1: (1981) 84–91.

Pampin, Juan. "ATS: A System for Sound Analysis Transformation and Synthesis Based on a Sinusoidal plus Crtitical-Band Noise Model and Psychoacoustics." In *Proceedings of the International Computer Music Conference*. Miami, FL: ICMC, 2004, 402–405.

Phillips, Andrew T. "A Containter for a Set of Ranges." *Dr. Dobb's Journal* 17, 6: (1999) 75–80. http://www.ddj.com/cpp/184403660?pgno=1.

Plante, F., G. Meyer, and W. A. Ainsworth. "Improvement of Speech Spectrogram Accuracy by the Method of Reassignment." *IEEE Transactions on Speech and Audio Processing* 6, 3: (1998) 282–286.

Puckette, Miller. "Phase-locked vocoder." In *IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 1995, 222–225.

———. *Theory and Techniques of Electronic Music*. World Scientific Press, 2007. http://crca.ucsd.edu/~msp/techniques/v0.11/book.pdf. Online edition draft, December 30, 2006.

Roads, Curtis. *Microsound*. Cambridge, MA: The MIT Press, 2001.

Robel, Axel. "A New Approach to Transient Preservation in the Phase Vocoder." In *Proceedings of the 6th International Conference on Digital Audio Effects*. London, UK: DAFx-03, 2003.

Robel, Axel, and Xavier Rodet. "Efficient Spectral Envelope Estimation and its Application to Pitch Shifting and Envelope Preservation." In *Proceedings of the 8th International Conference on Digital Audio Effects*. DAFx-05, 2005.

Rodet, Xavier, and Philippe Depalle. "Spectral Envelopes and Inverse FFT Synthesis." In *Proceedings of the 93rd Audio Engineering Society Convention*. AES, 1992. Preprint no. 3393.

Rodet, Xavier, and Adrien Lefèvre. "The Diphone program: New features, new synthesis methods, and experience of musical use." In *Proceedings of the International Computer Music Conference*. Thessaloniki, Grece: ICMC, 1997, 418–421.

Sandowsky, George. "My 2nd Computer was a UNIVAC I: Some Reflections on a Career in Progress." *Connect: Information Technology at NYU* http://www.nyu.edu/its/pubs/connect/archives/index01.html.

Schoenberg, Arnold. *Theory of Harmony (Harmonielehre)*. Translated by Roy E. Carter. Berkeley, CA: University of California Press, 1983. First published 1922.

Schwarz, Diemo. *Spectral Envelopes in Sound Analysis and Synthesis*. Master's thesis, Institut für Informatik, Stuttgart / IRCAM, 1998.

———. "Current Research in Concatenative Sound Synthesis." In *Proceedings of the International Computer Music Conference*. Barcelona, Spain: ICMC, 2005.

Serra, Xavier. *A System for Sound Analysis/Transformation/Synthesis based on a Deterministic plus Stochastic Decomposition*. Ph.D. thesis, Stanford University, Stanford, CA, 1989.

Shneiderman, Ben. "Direct Manipulation: A Step Beyond Programming Languages." *IEEE Computer* 16, 8: (1983) 57–69.

Smith, Julius O. "Viewpoints on the History of Digital Synthesis.", 1992, Accessed July 5 2007. http://ccrma.stanford.edu/~jos/kna/kna.pdf.

Smith, Julius O. "Virtual Acoustic Musical Instruments: Review and Update." *Journal of New Music Research* 33, 3: (2004) 283–304.

Smith, Julius O., and Xavier Serra. "PARSHL: An Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation." In *Proceedings of the International Computer Music Conference*. Champaign-Urbana, IL: ICMC, 1987, 290–297. http://ccrma.stanford.edu/~jos/parshl/parshl.pdf.

Stilson, Timothy. *Efficiently-Variable Non-Oversampled Algorithms in Virtual-Analog Music Synthesis*. Ph.D. thesis, Stanford University, Stanford, CA, 2006. http://ccrma.stanford.edu/~stilti/papers/TimStilsonPhDThesis2006.pdf.

Taube, Heinrich. "λGTK: A Portable Graphics Layer for Common Music." In *Proceedings of the International Computer Music Conference*. Barcelona, Spain: ICMC, 2005.

Terhardt, Ernst, Gerhard Stoll, and Manfred Seewann. "Pitch of complex signals according to virtual-pitch theory: Tests, examples, and predictions." *Journal of the Acoustical Society of America* 71, 3: (1982) 671–678.

Välimäki, Vesa, and Antti Huovilainen. "Oscillator and Filter Algorithms for Virtual Analog Synthesis." *Computer Music Journal* 20, 2: (2006) 19–31.

Varèse, Edgard, and Chou Wen-chung. "The Liberation of Sound." *Perspectives of New Music* 5, 1: (1966) 11–19.

Wishart, Trevor. "Computer Sound Transformation: A personal perspective from the U.K.", 2000, Accessed August 26, 2008. http://www.composersdesktop.com/trnsform.htm.

Wright, Matthew, Amar Chaudhary, Adrian Freed, Sami Khoury, and David Wessel. "Audio Applications of the Sound Description Interchange Format Standard." In *Proceedings of the 107th Audio Engineering Society Convention*. AES, 1999a. Preprint no. 5032.

Wright, Matthew, Richard Dudas, Sami Khoury, Raymond Wang, and David Zicarelli. "Supporting the Sound Description Interchange Format in the Max/MSP Environment." In *Proceedings of the International Computer Music Conference*. ICMC, 1999b.

Zbyszynski, Michael, Matthew Wright, and Edmund Campion. "Design and Implementation of CNMAT's Pedagogical Software." In *Proceedings of the International Computer Music Conference*. ICMC, 2007.

Zeitlin, Vadim. "The wxWindows Cross-Platform Framework." *Dr. Dobb's Journal* 24, 3: (2001) 106–112.

Appendix A. *Tear of the Clouds* score

Tear of the Clouds is a composition for 13 instruments and electronic sounds. It was first performed May 6, 2008 by the Manhattan Sinfonietta, Jeffrey Milarsky, conductor and music director. In performance, the ensemble includes a MIDI keyboard that functions in a dual role: as a traditional keyboard that controls a variety of sampled and quasi-synthetic pitch sounds, and as a trigger for pre-synthesized sonic complexes. The tuning of the keyboard is dynamic and adjusts to the prevailing non-tempered harmony. Pre-synthesized sonic events were realized using a combination of SPEAR, Common Music, and CLM. Realtime synthesis and sample playback was done using SuperCollider.

A.1 Program Note

As the highest lake in the Adirondacks, Lake Tear of the Clouds is the Hudson River's source. The piece was inspired in large part by my experiences and observations living along this waterway — the vast wide expanse at Tappan Zee, the churning gray whitecaps of a stormy day, the vistas from the New Jersey Palisades, and an imaginary slow motion dive from the top of the cliffs into the murky depths. Some of the sonic materials used in both the ensemble and electronics are derived from recordings of water sounds made at various points along the river. Other sound spectra are derived from sources such as bells, tam-tams, and voices. In many instances these inner structure of these sounds are retuned via a process of analysis and resynthesis.

Tear of the Clouds

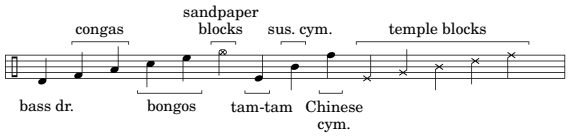
Duration — 10'54"

Instrumentation

Flute doubling Piccolo
Oboe
B♭ Clarinet doubling B♭ Bass Clarinet
Bassoon

Horn
Trumpet
Trombone (with F trigger)

Percussion
- crotales (upper octave, written C5–C6)
- vibraphone (with motor)
- marimba (4 1/3 octave, A3–C7)
- temple blocks (set of 5)
- sandpaper blocks
  (one block mounted, the other free and playable with one hand)
- Chinese cymbal
- medium suspended cymbal
- 2 bongos
- 2 conga drums, high and low (or toms)
- large tam-tam
- bass drum



Computer / electronic sounds (see explanation below)
Keyboard
- 88-key MIDI keyboard with sustain pedal, connected to computer

Violin
Viola
Cello
Contrbass

All instruments sound the written pitch with the following standard exceptions:

- Piccolo — 1 octave higher
- Crotales — 2 octaves higher
- Contrabass — 1 octave lower

Accidentals

- quarter-tone flat
- three-quarter-tones flat
- quarter-tone sharp
- three-quarter-tones sharp

Whenever possible, woodwinds should execute quarter tones using alternate fingerings. Trumpet and horn should use alternate fingerings based on the 7th and 11th partials.

Strings

- exaggerated bow pressure creating a low, noisy, "crushed" sound
- resume normal bow pressure

Note: strings should play with ordinary vibrato unless otherwise indicated

Electronics

There are two types of electronic sounds indicated in the score, those than are triggered by the MIDI keyboard and those that are triggered by the computer.

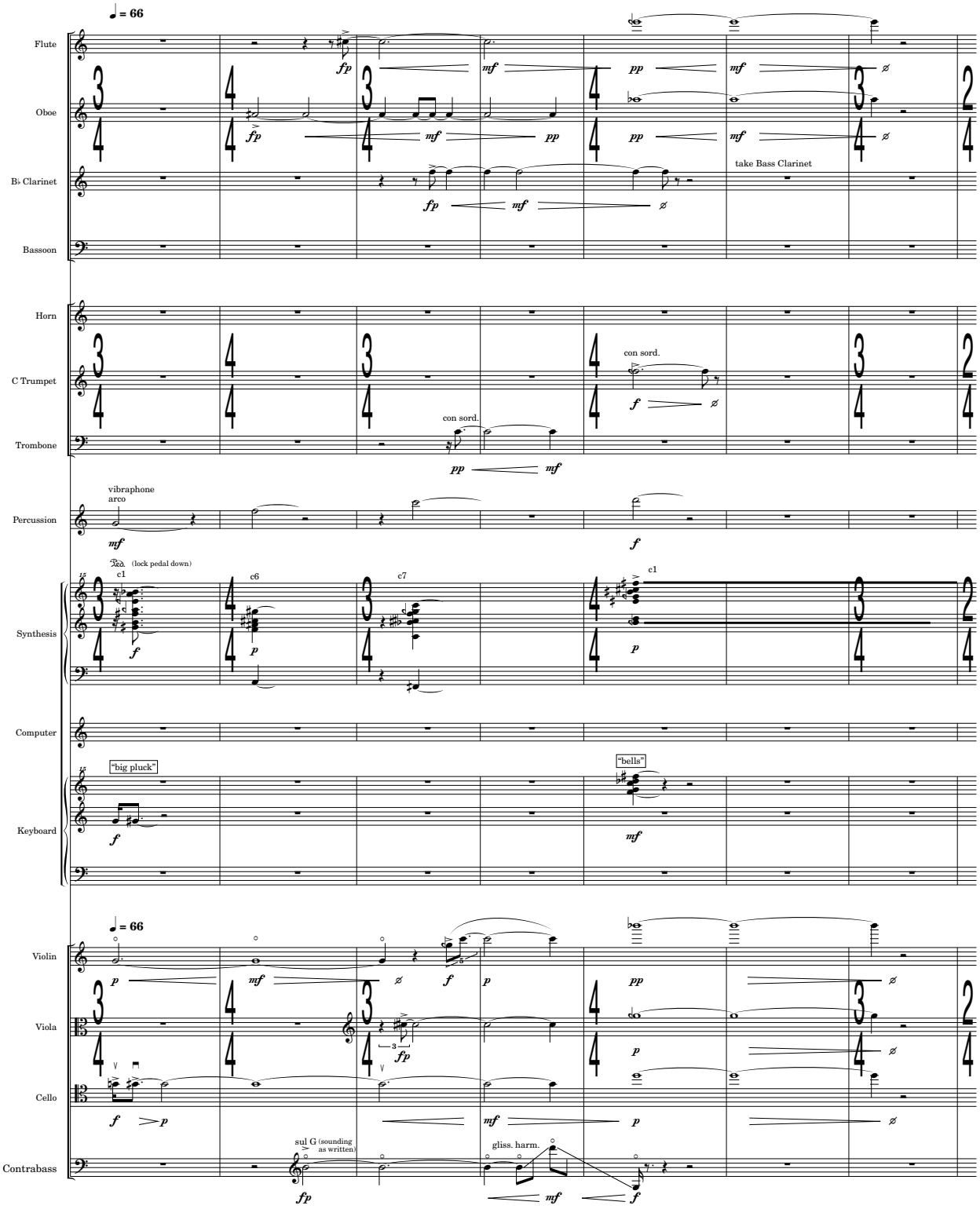
The MIDI keyboard functions much like a traditional keyboard playing a variety of sampled and quasi-synthetic pitched sounds. A timbre description is indicated in quotation marks (e.g. "bowed pluck"). The computer operator takes care of switching timbres at the appropriate time. The actual pitches produced, which are often microtonally inflected, are indicated in the staves labeled "Synthesis."

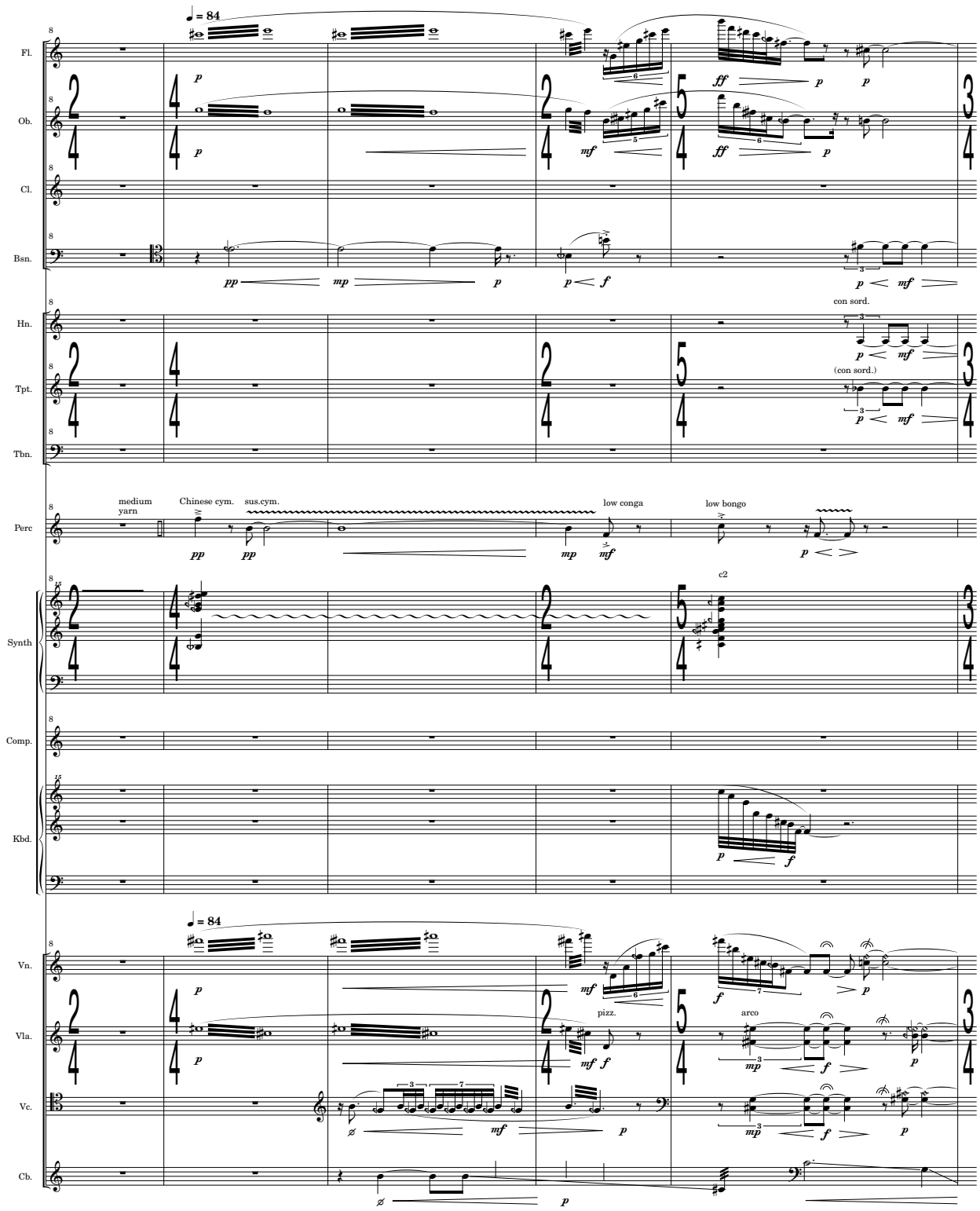
The computer operator also triggers various sonic events — chords, sustained tones, noise complexes, etc. These are also generally indicated in the "Synthesis" staves.

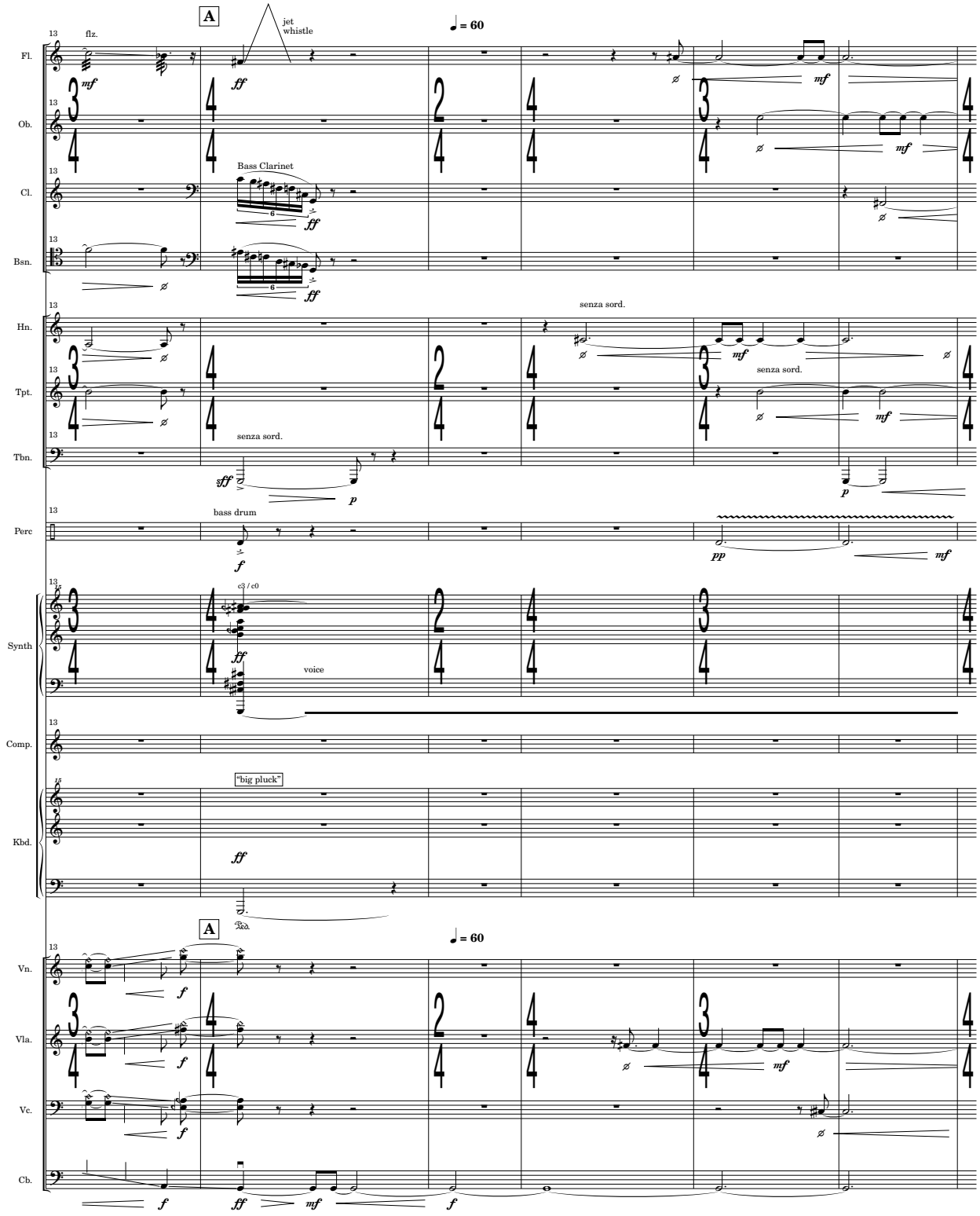
C SCORE

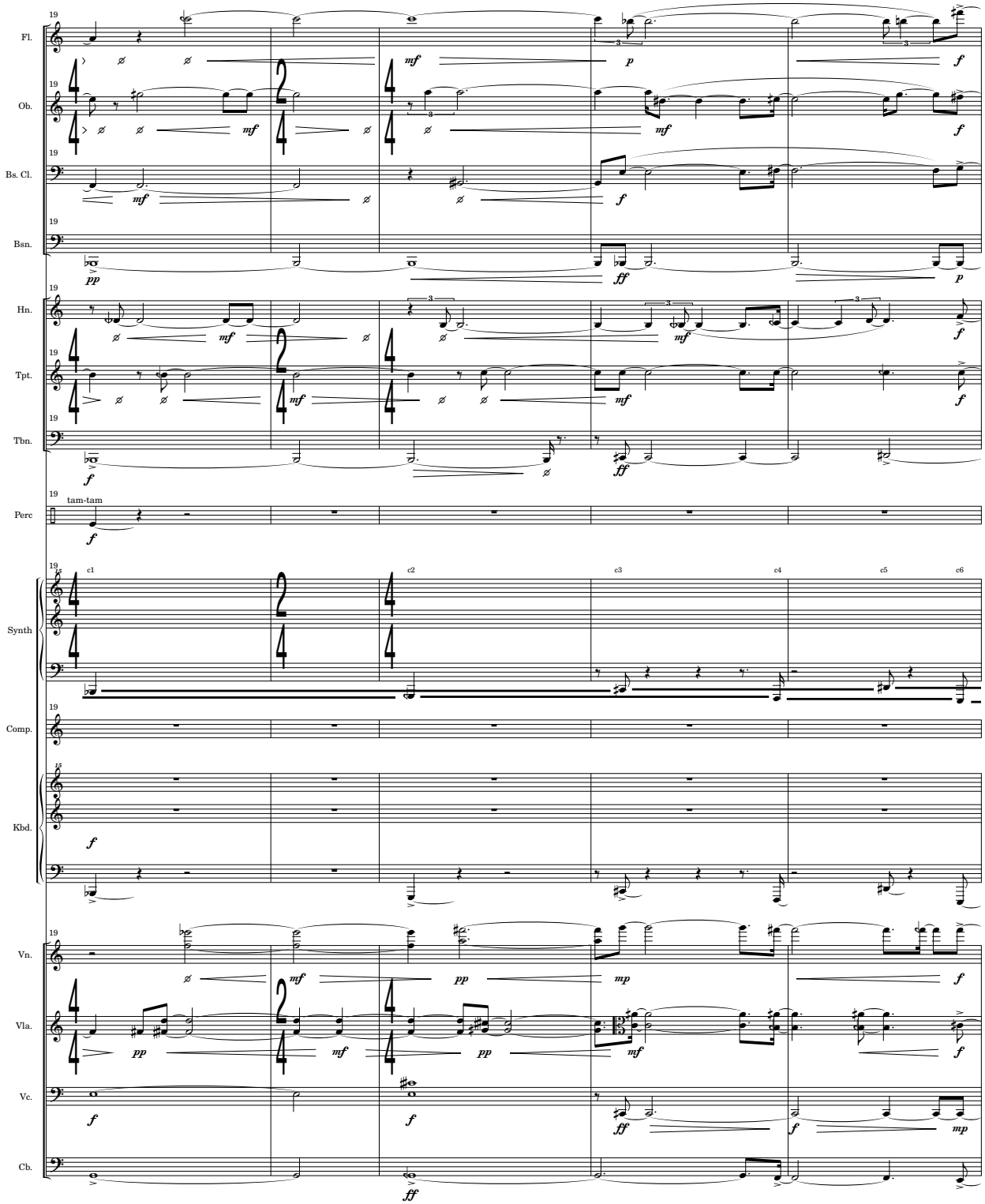
Tear of the Clouds

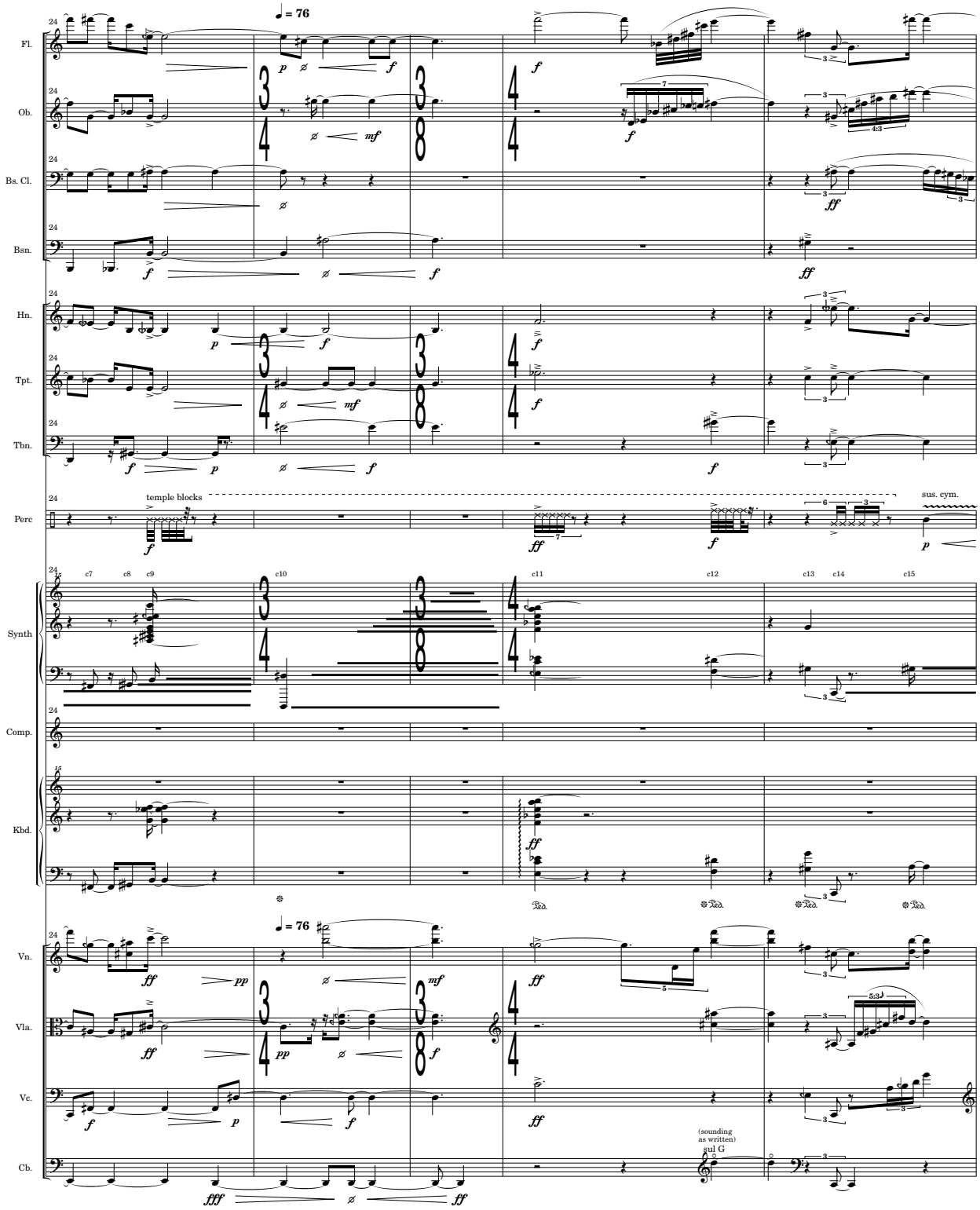
Michael KLINGBEIL

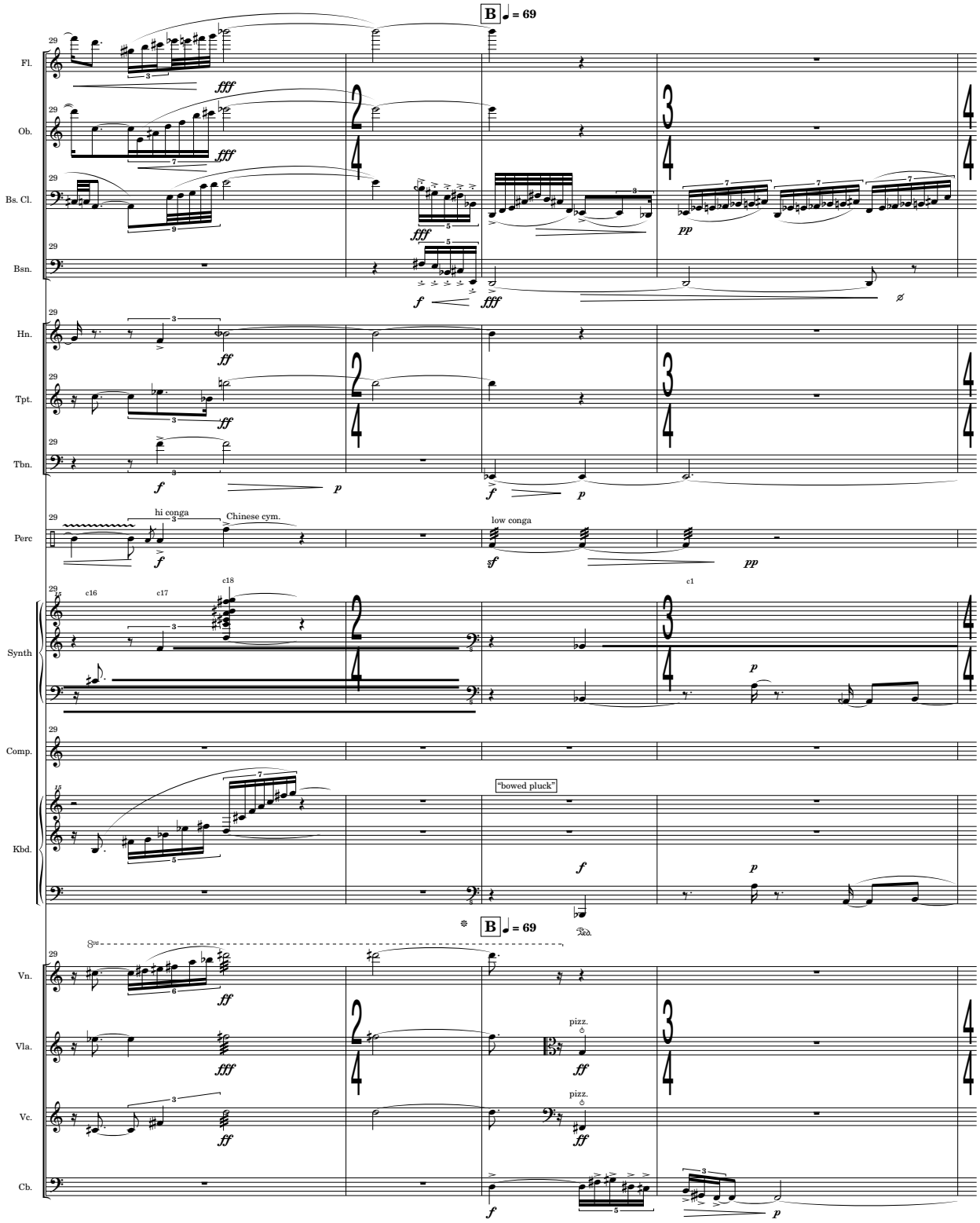


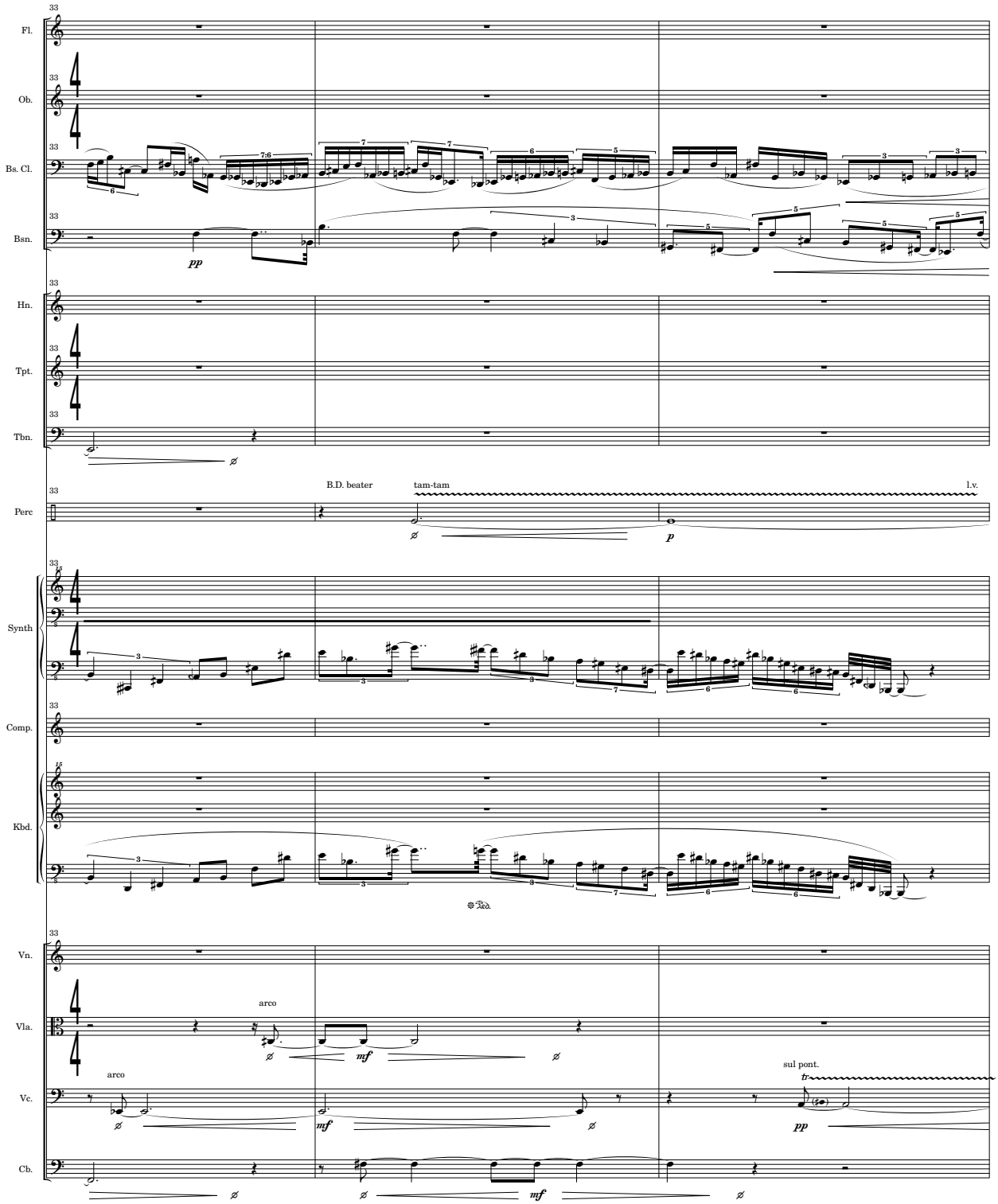

♩ = 84
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
con sord.
(con sord.)
medium yarn
Chinese cym.
sus.cym.
low conga
low bongo
c2
pizz.
arco

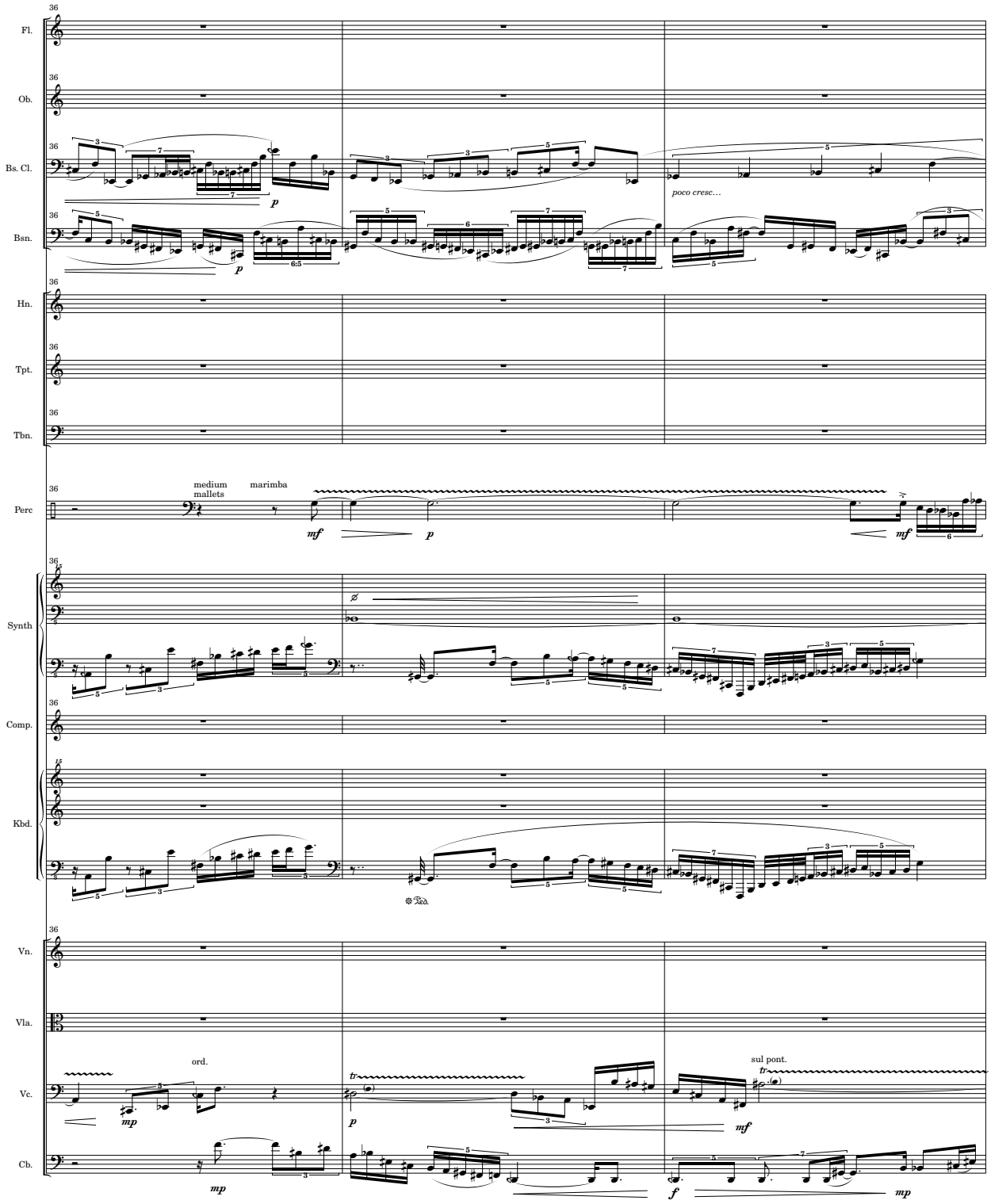

A
jet whistle
♩ = 60
flz.
Fl.
Ob.
Cl.
Bass Clarinet
Bsn.
Hn.
senza sord.
Tpt.
Tbn.
Perc
bass drum
Synth
c3 / c0
voice
Comp.
"big pluck"
Kbd.
Vn.
Vla.
Vc.
Cb.

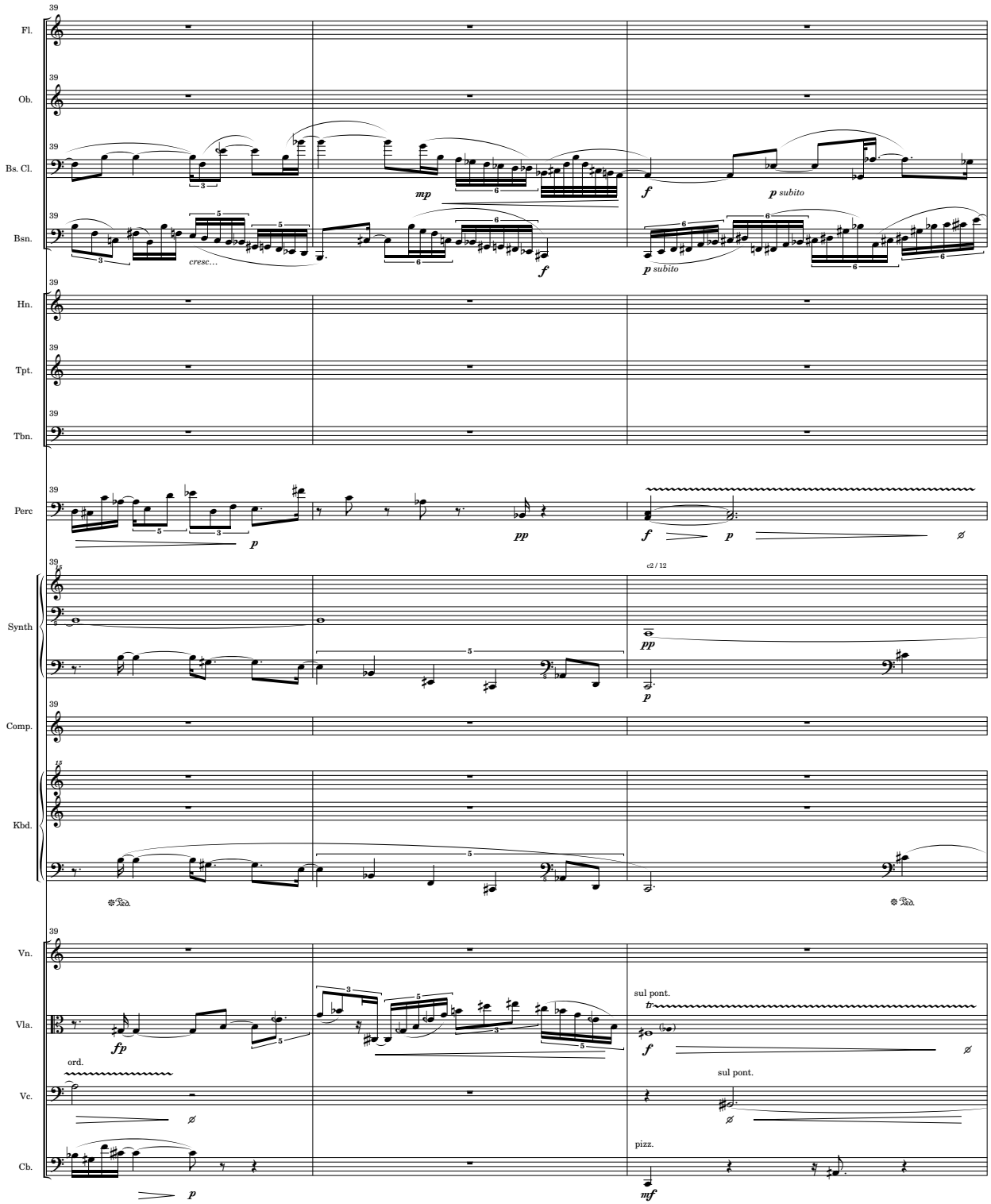

Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
tam-tam
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.

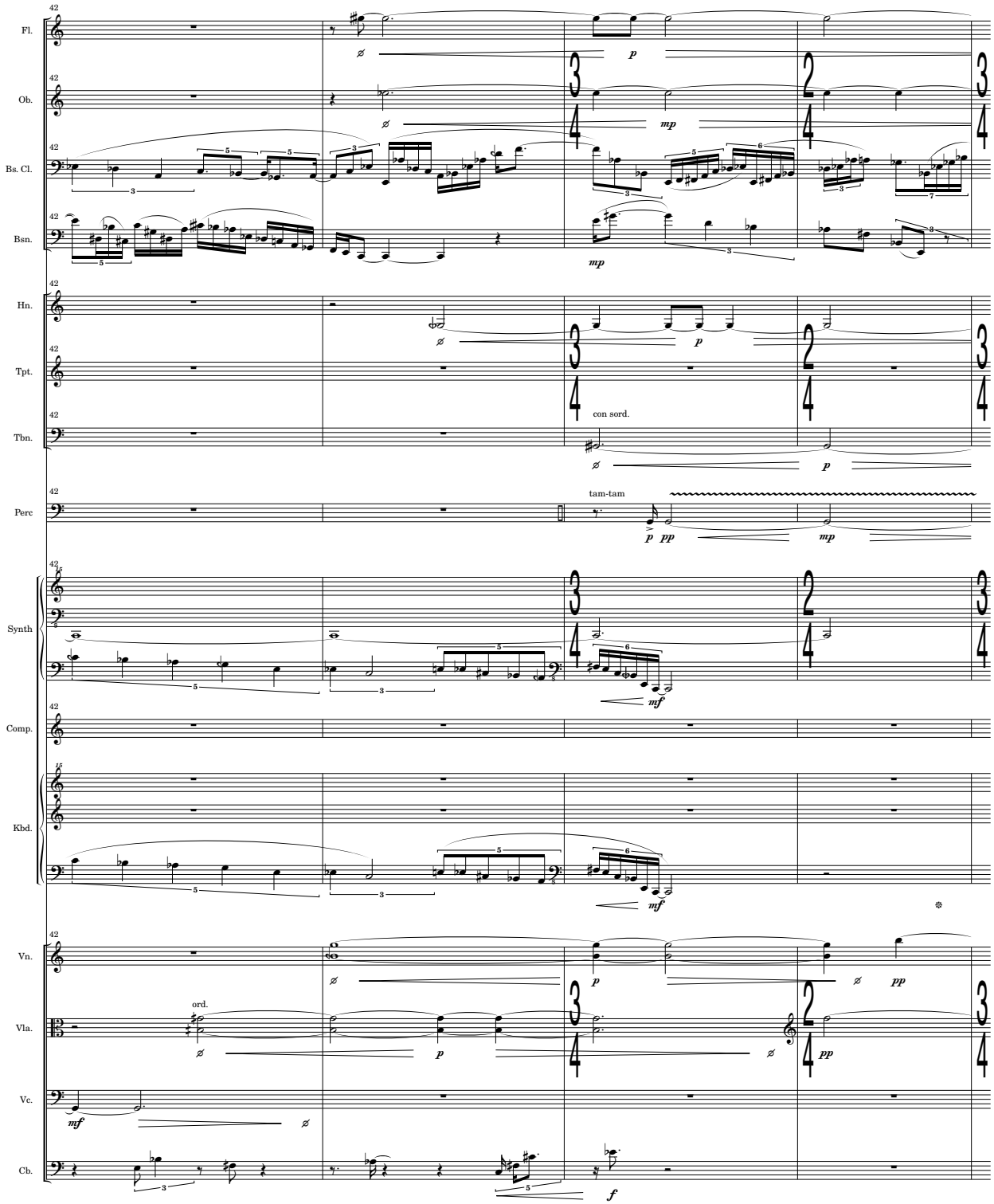

♩ = 76
Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
temple blocks
sus. cym.
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
(sounding as written)
sul G

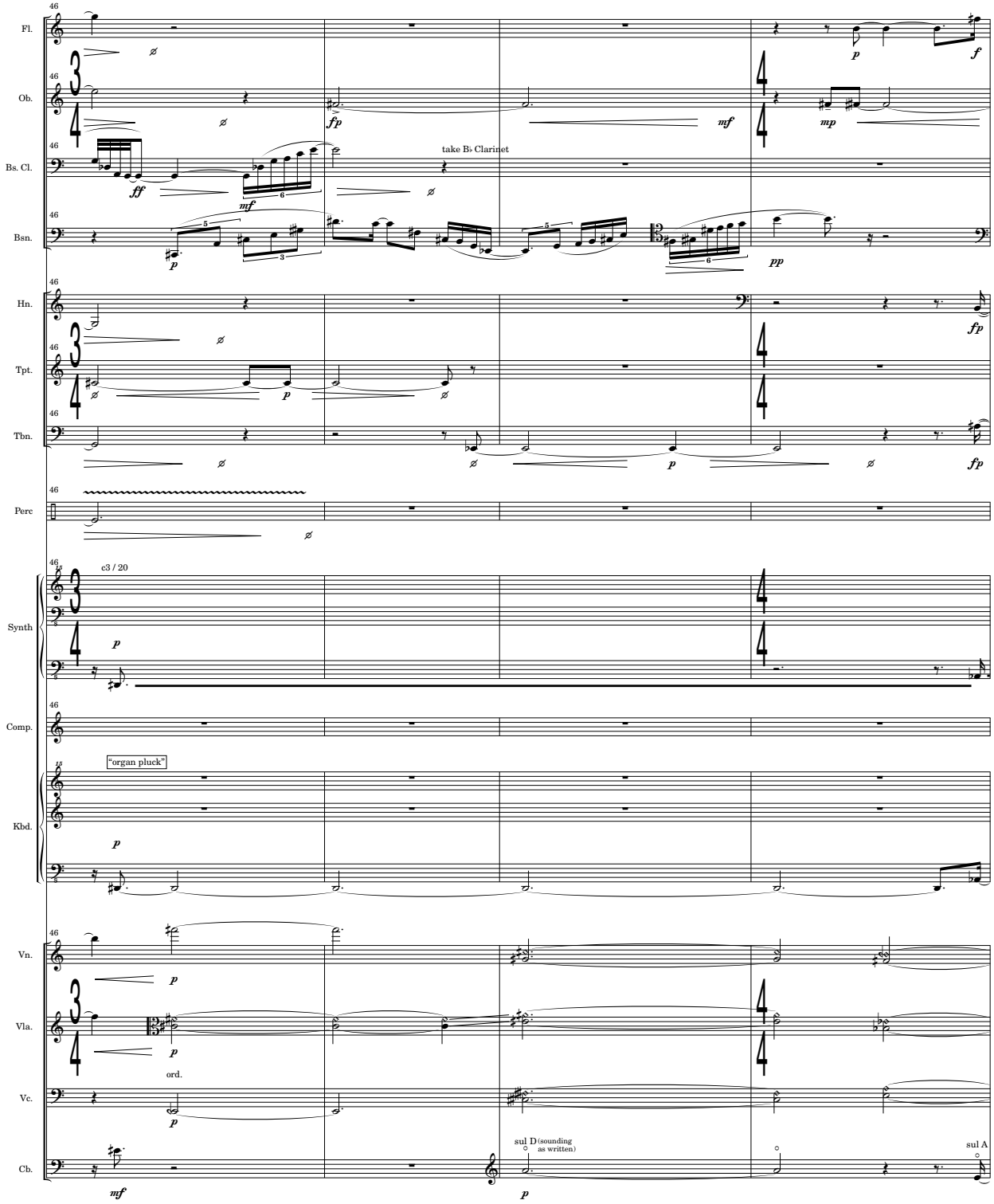

B = 69
Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
hi conga
Chinese cym.
low conga
Synth
c16
c17
c18
c1
Comp.
Kbd.
"bowed pluck"
B = 69
Vn.
Vla.
Vc.
Cb.
pizz.
pizz.

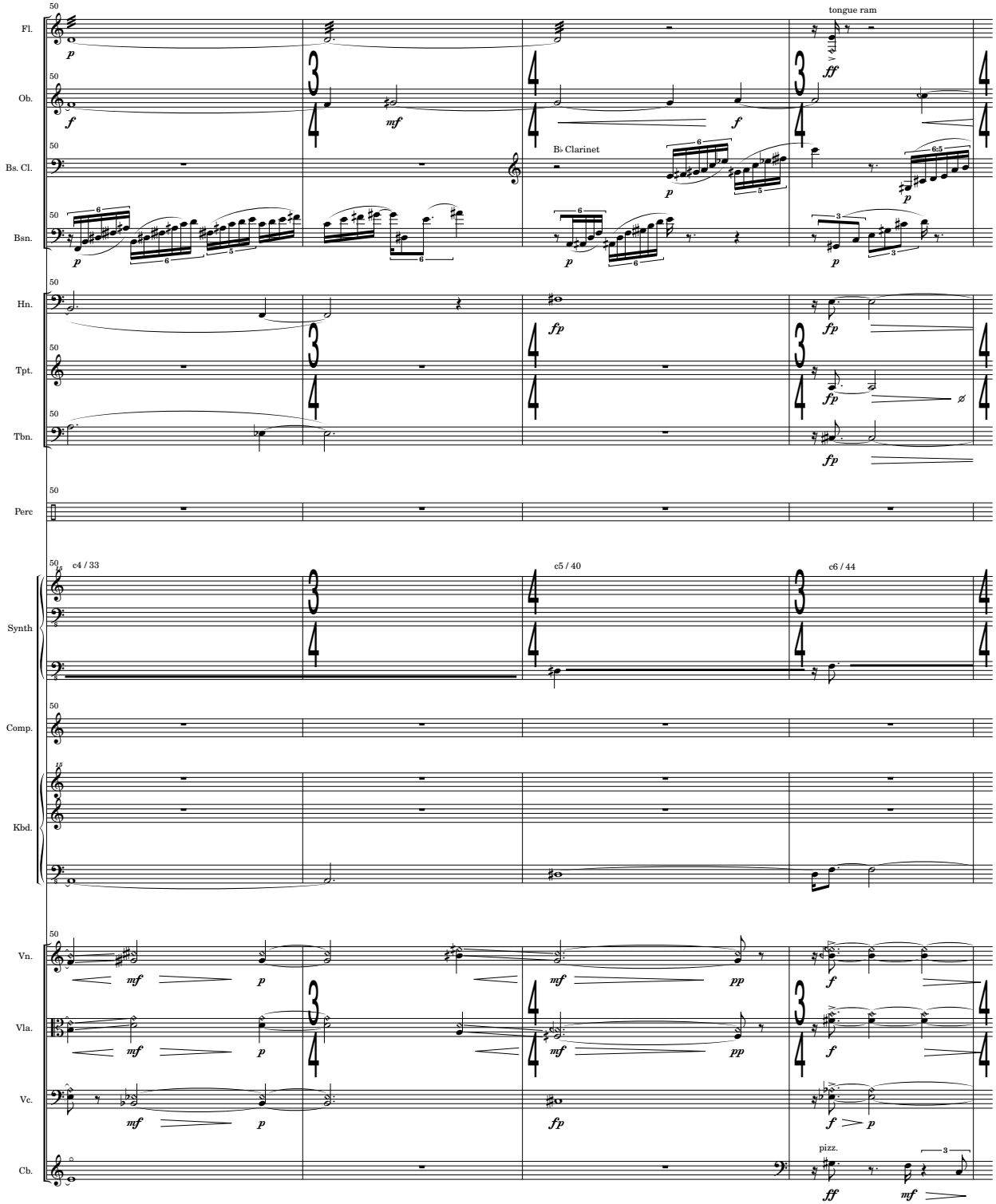

Fl.
Ob.
Bs. Cl.
Bsn.
pp
Hn.
Tpt.
Tbn.
Perc
B.D. beater
tam-tam
l.v.
p
Synth
Comp.
Kbd.
Vn.
Vla.
arco
mf
Vc.
arco
mf
sul pont.
tr
pp
Cb.
mf

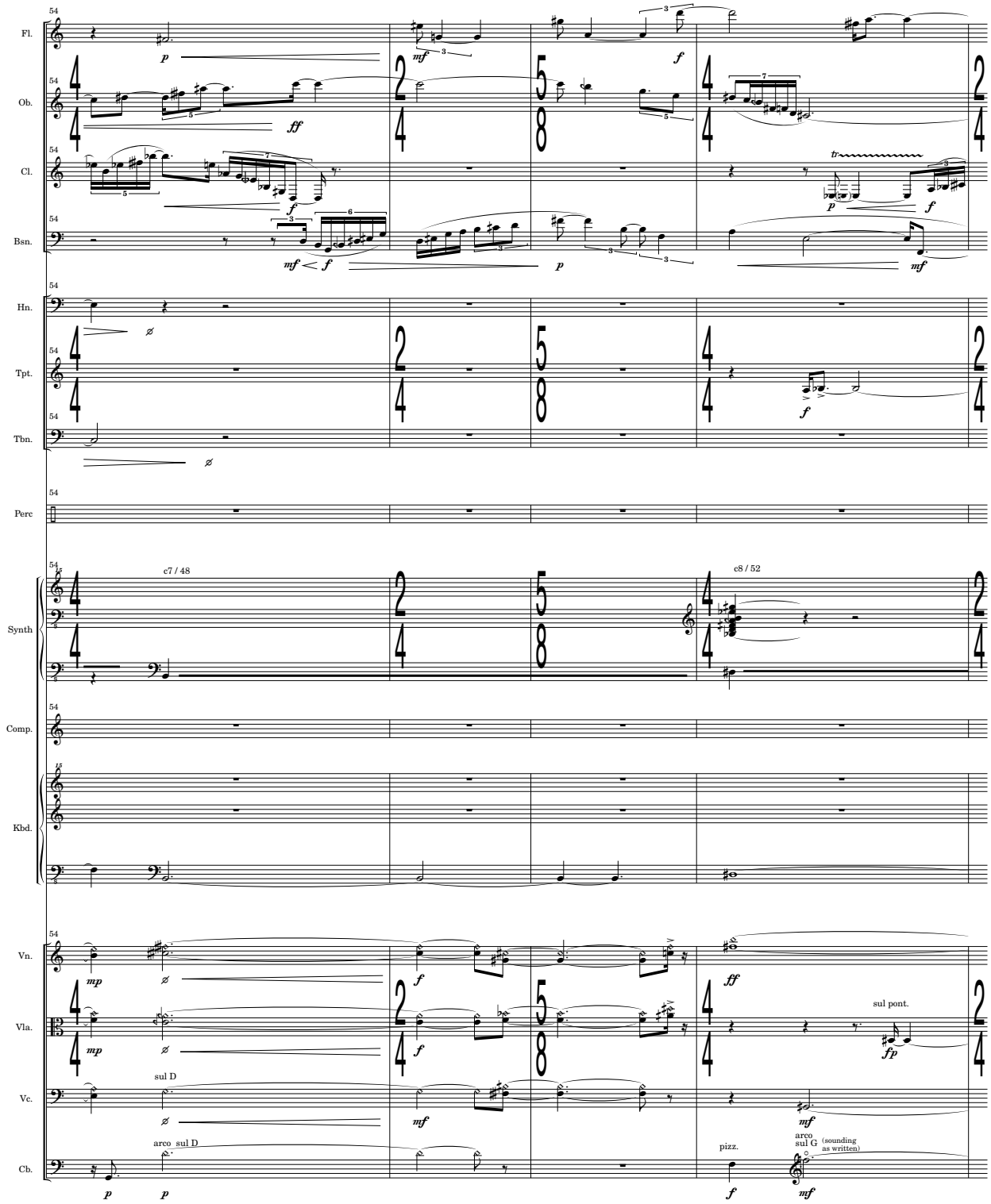

Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
poco cresc…
medium mallets
marimba
ord.
sul pont.

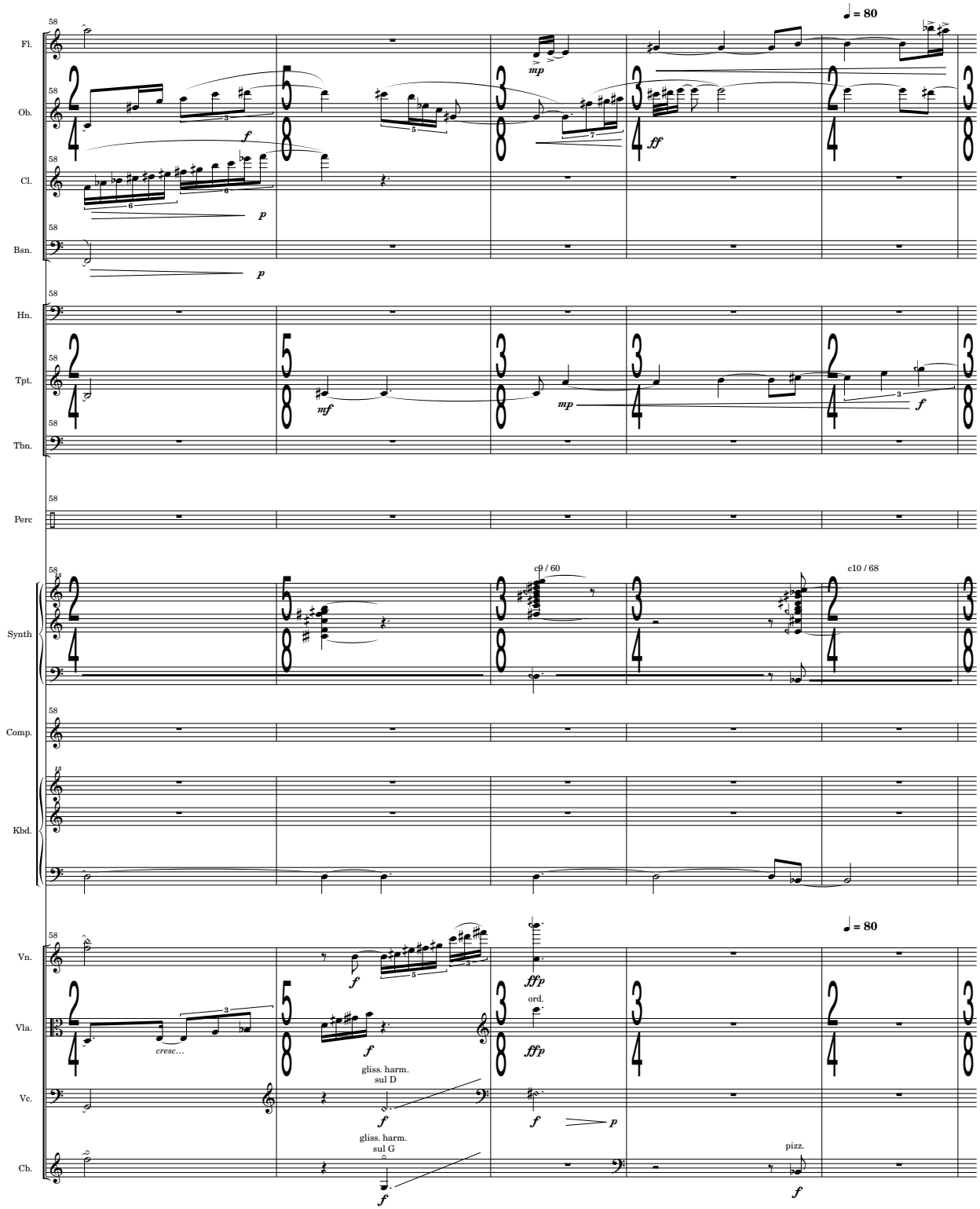

39
Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
mp
f
p subito
cresc…
pp
fp
c2 / 12
sul pont.
ord.
pizz.
mf

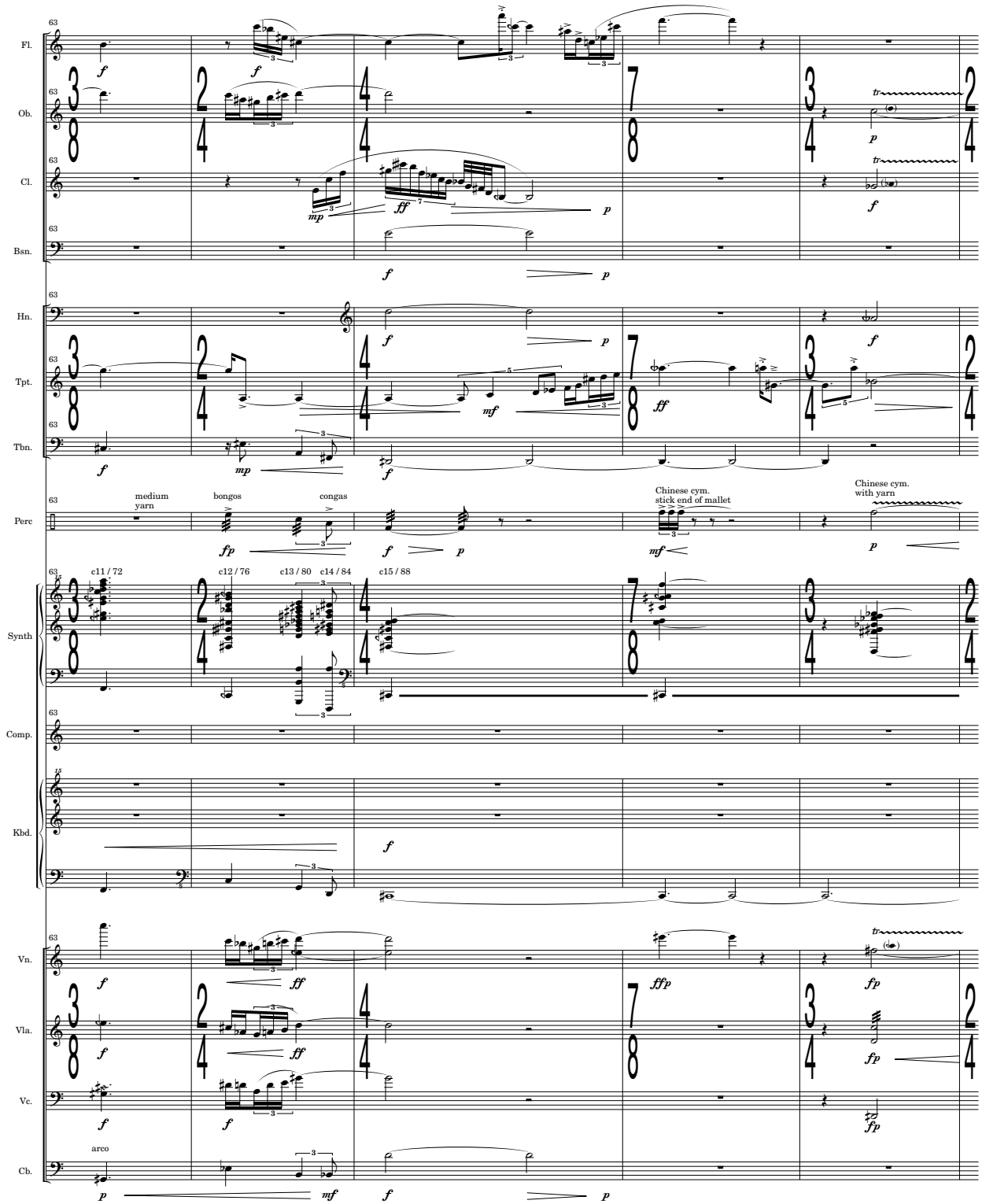

Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
con sord.
Tbn.
Perc
tam-tam
Synth
Comp.
Kbd.
Vn.
ord.
Vla.
Vc.
Cb.

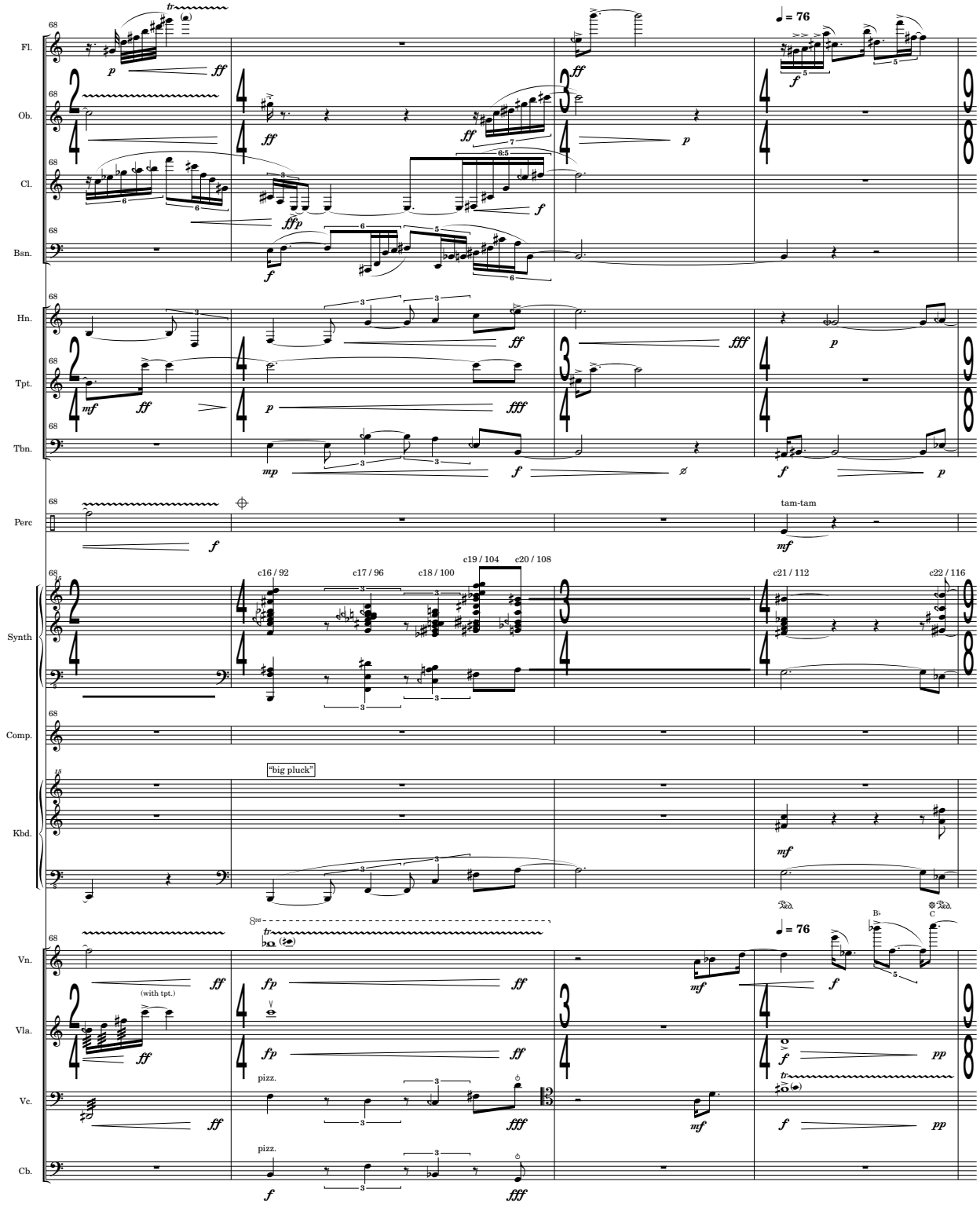

Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
take B♭ Clarinet
c3 / 20
"organ pluck"
ord.
sul D (sounding as written)
sul A

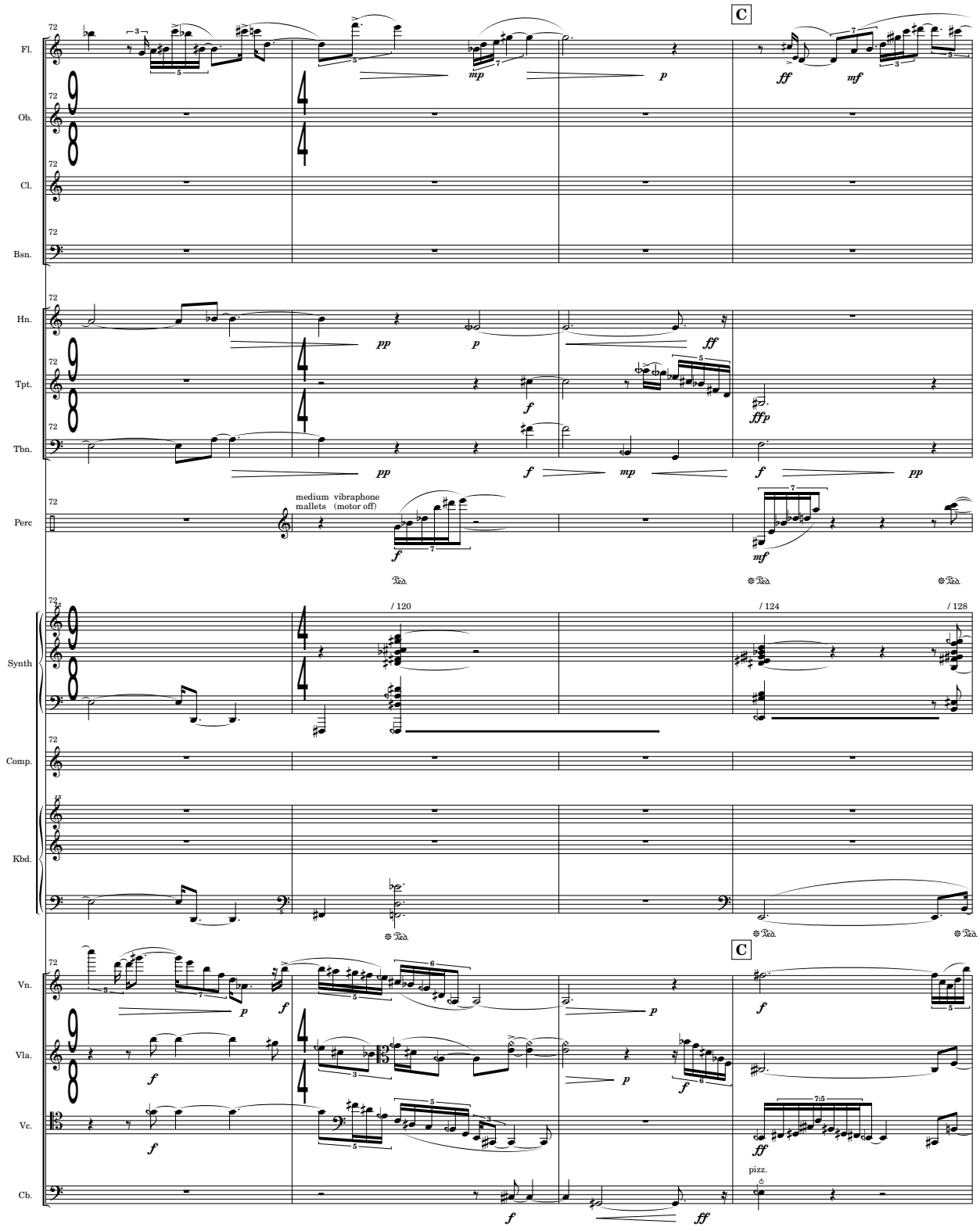

Fl.
Ob.
Bs. Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
tongue ram
B♭ Clarinet
c4 / 33
c5 / 40
c6 / 44
pizz.

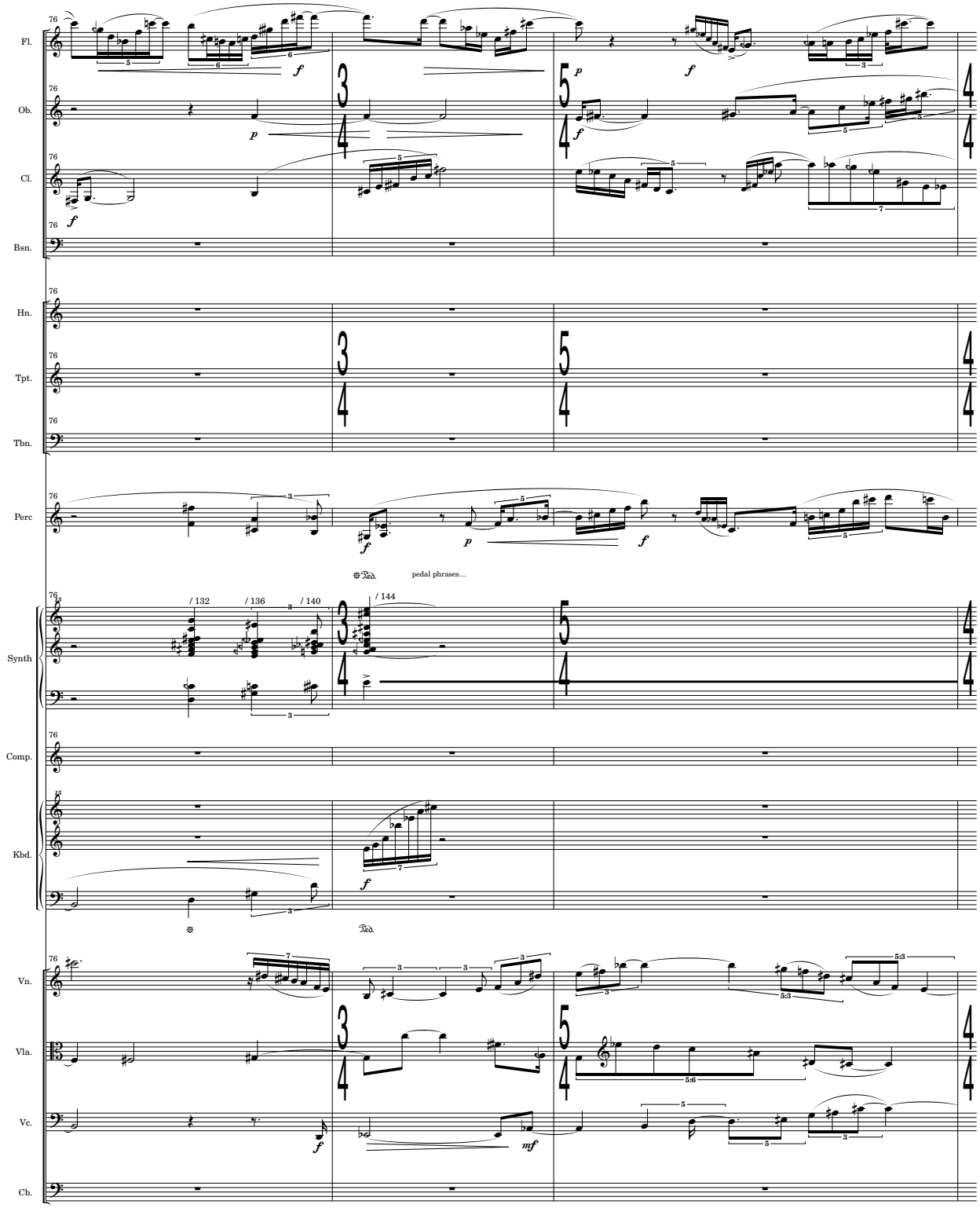

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
c7 / 48
c8 / 52
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
sul D
arco sul D
pizz.
arco
sul G
(sounding as written)
sul pont.

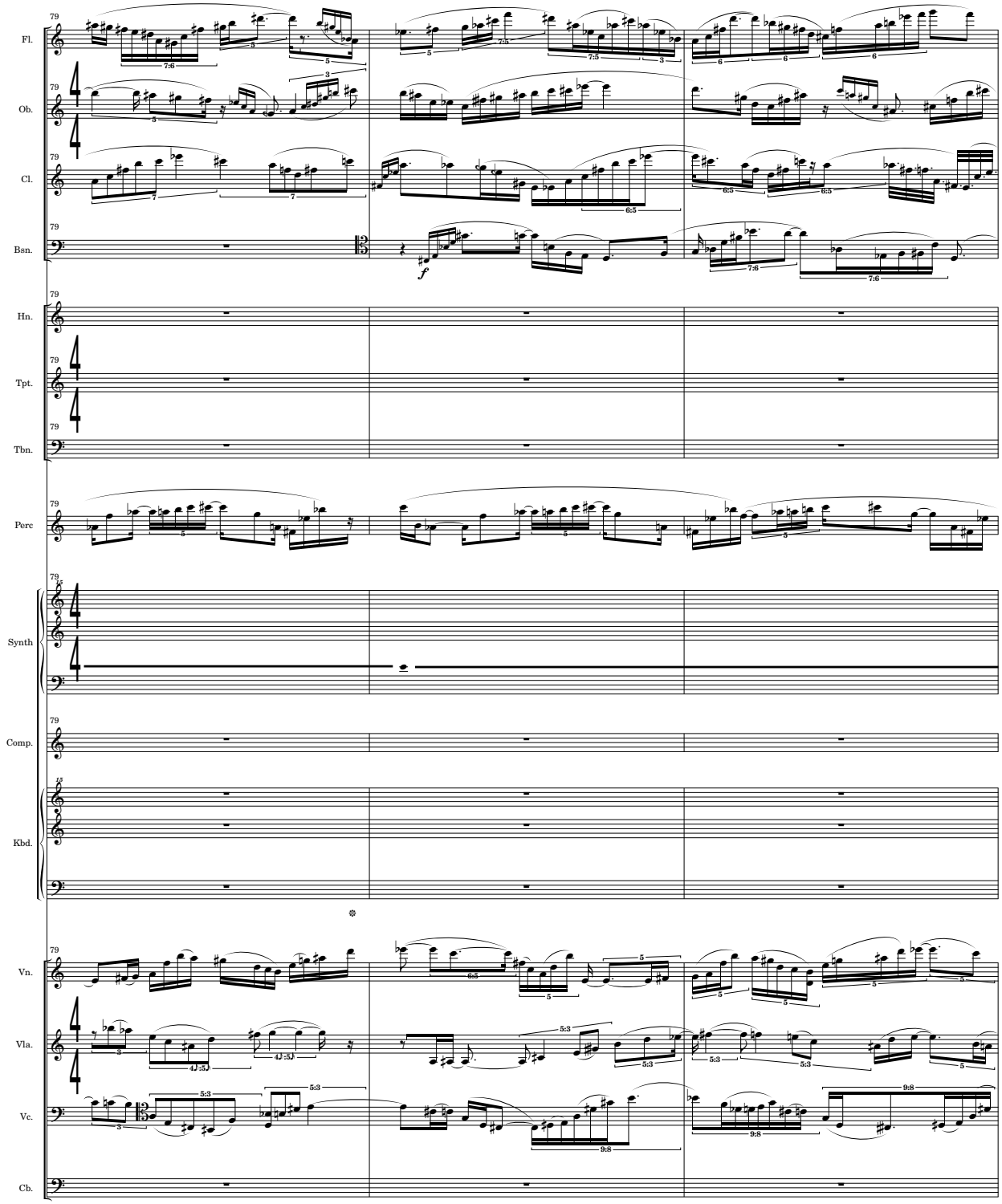

♩ = 80
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
c9 / 60
c10 / 68
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
cresc...
gliss. harm.
sul D
gliss. harm.
sul G
ord.
pizz.

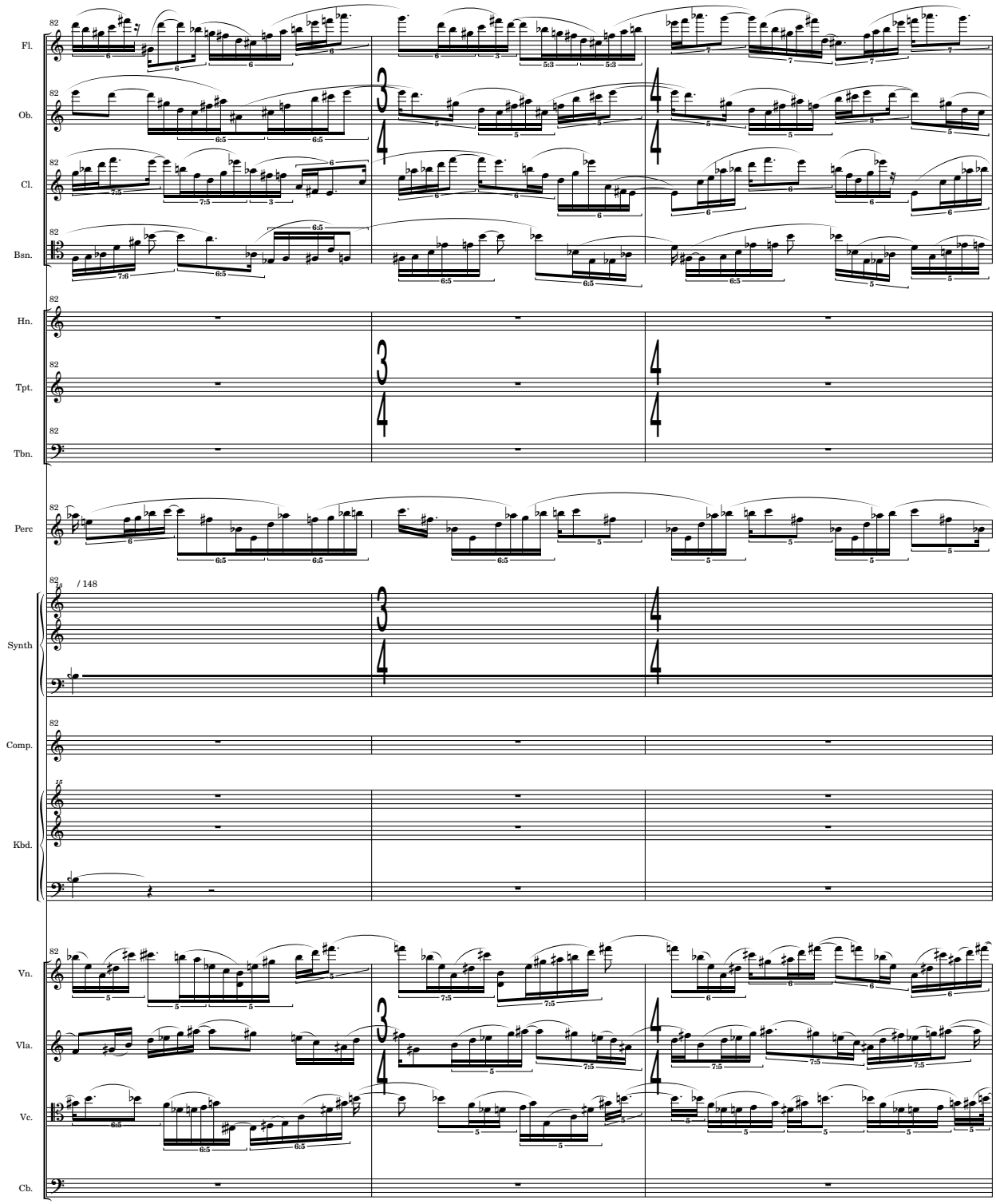

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
medium yarn
bongos
congas
Chinese cym. stick end of mallet
Chinese cym. with yarn
Synth
c11 / 72
c12 / 76
c13 / 80
c14 / 84
c15 / 88
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
arco

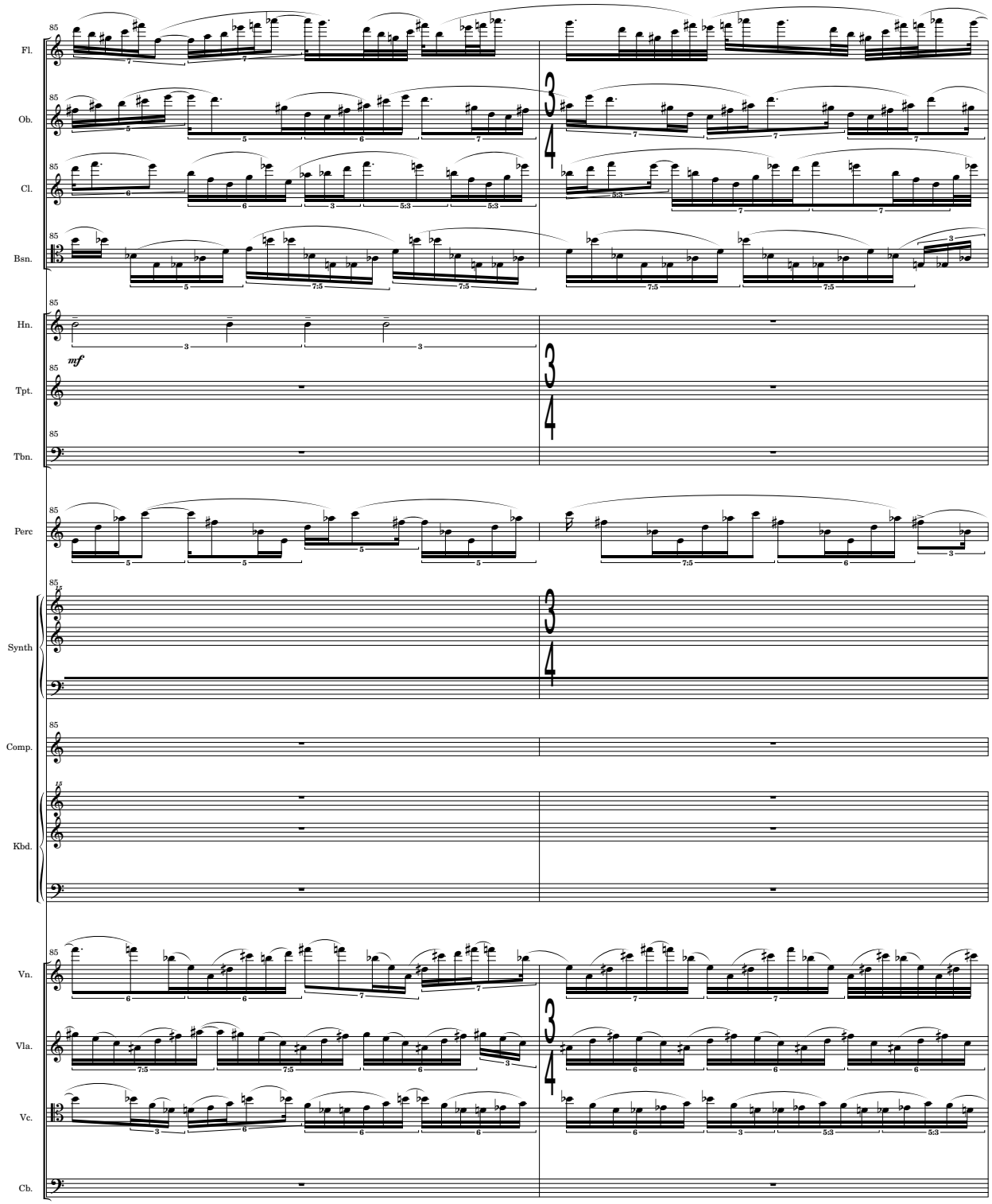

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
68
♩ = 76
tam-tam
c16 / 92
c17 / 96
c18 / 100
c19 / 104
c20 / 108
c21 / 112
c22 / 116
"big pluck"
(with tpt.)
pizz.
pizz.
Ped.
Ped.

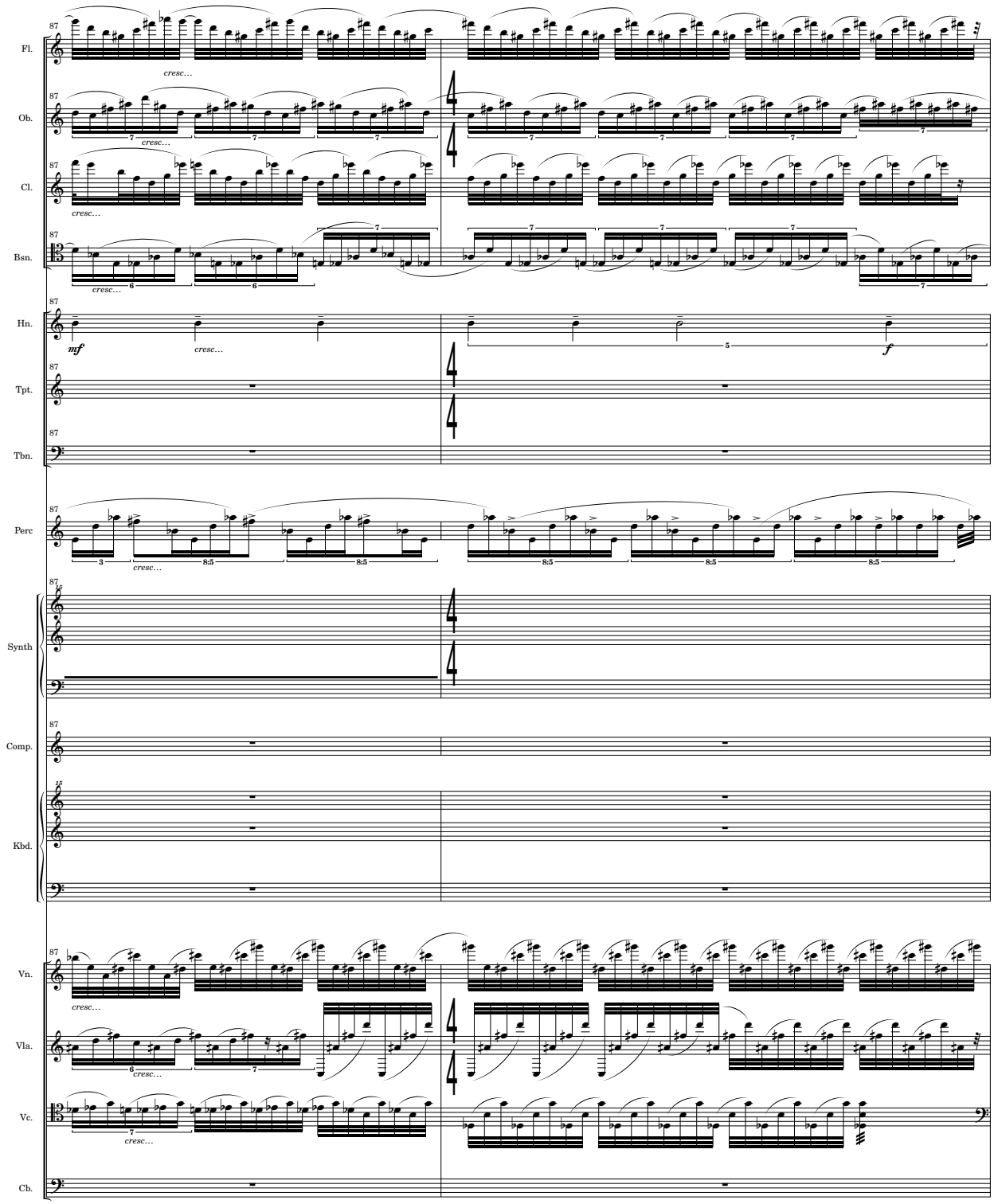

C
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
medium vibraphone mallets (motor off)
Synth
/ 120
/ 124
/ 128
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
pizz.

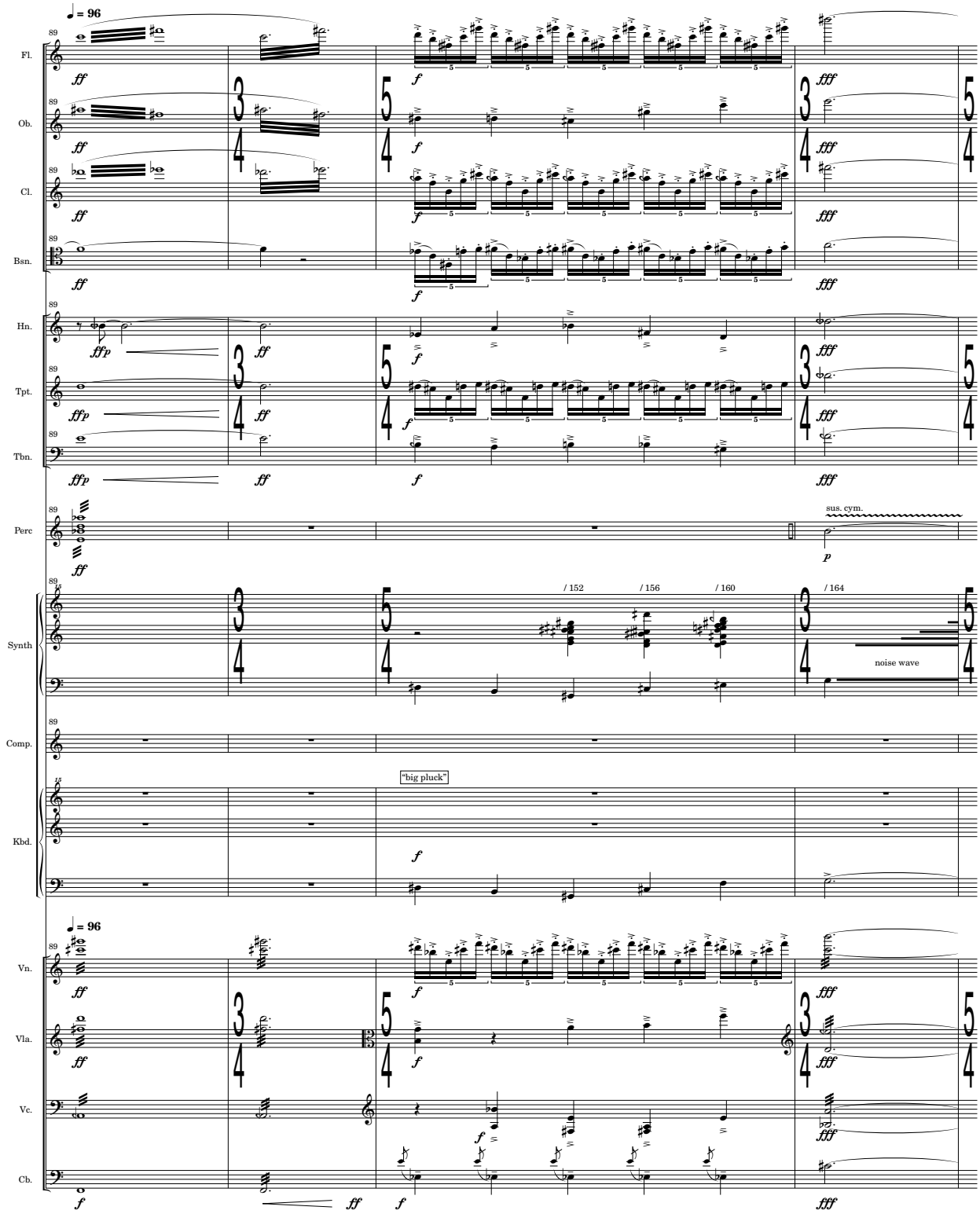

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
pedal phrases...
/ 132
/ 136
/ 140
/ 144

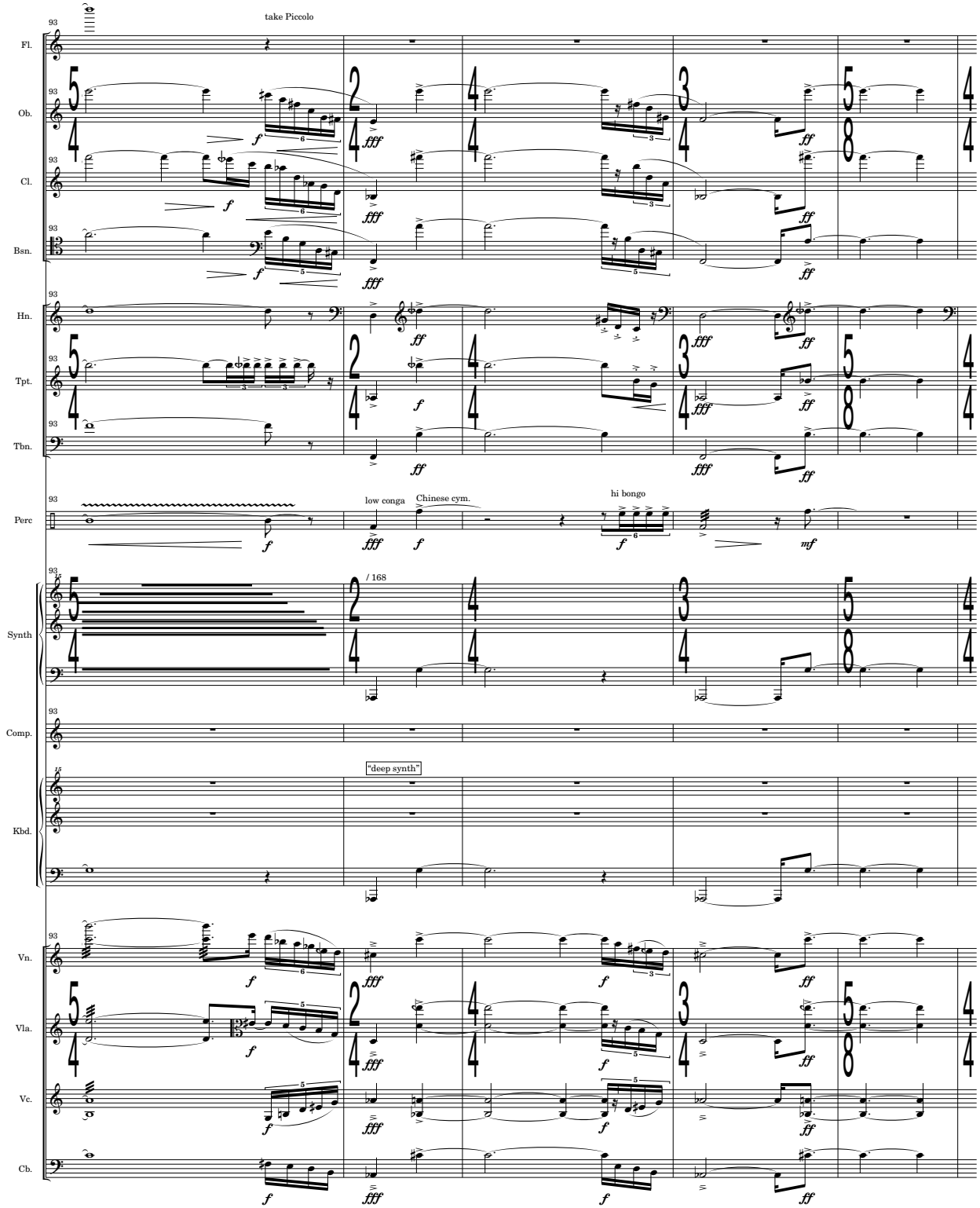

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
79
f

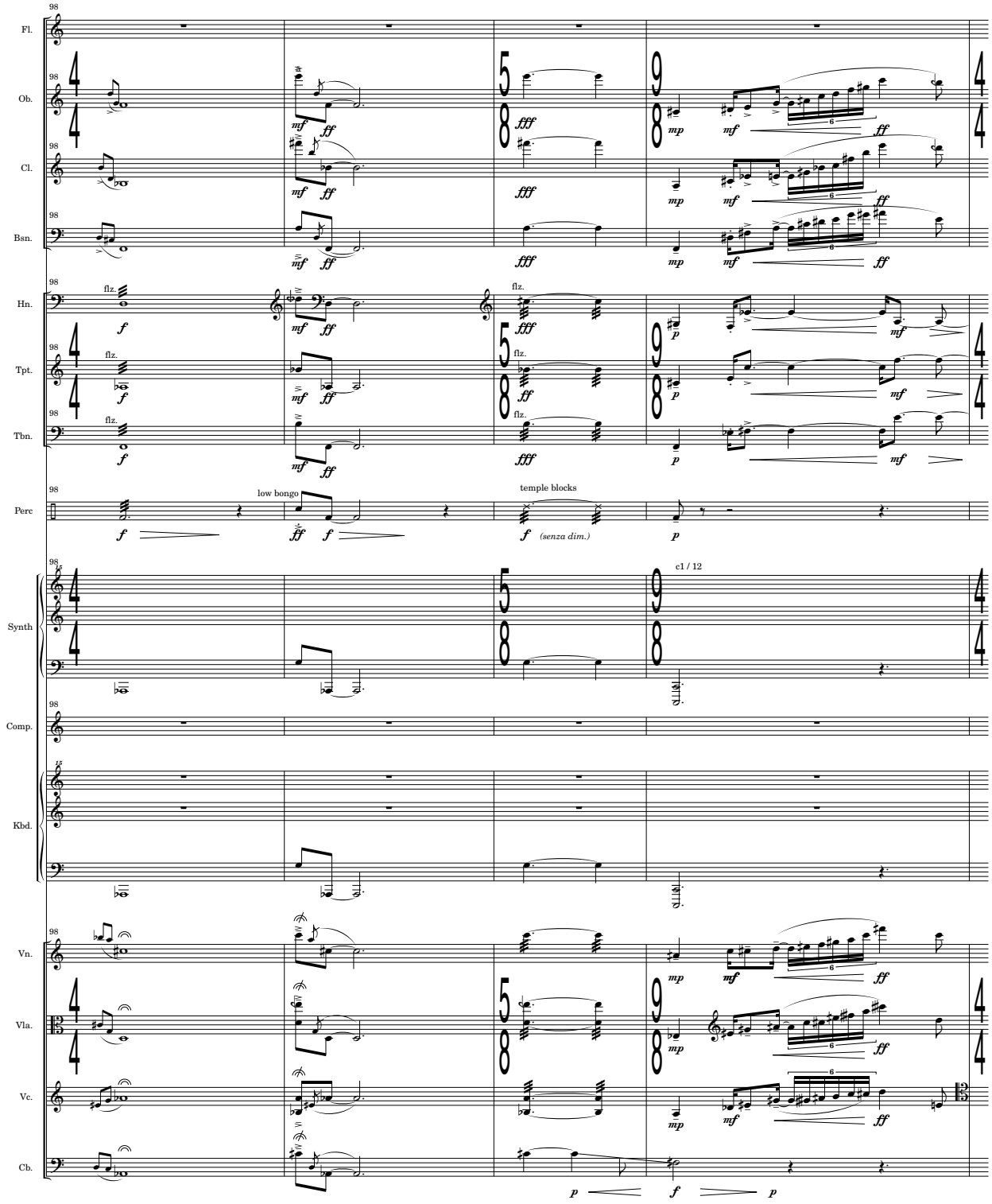

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
/ 148
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.

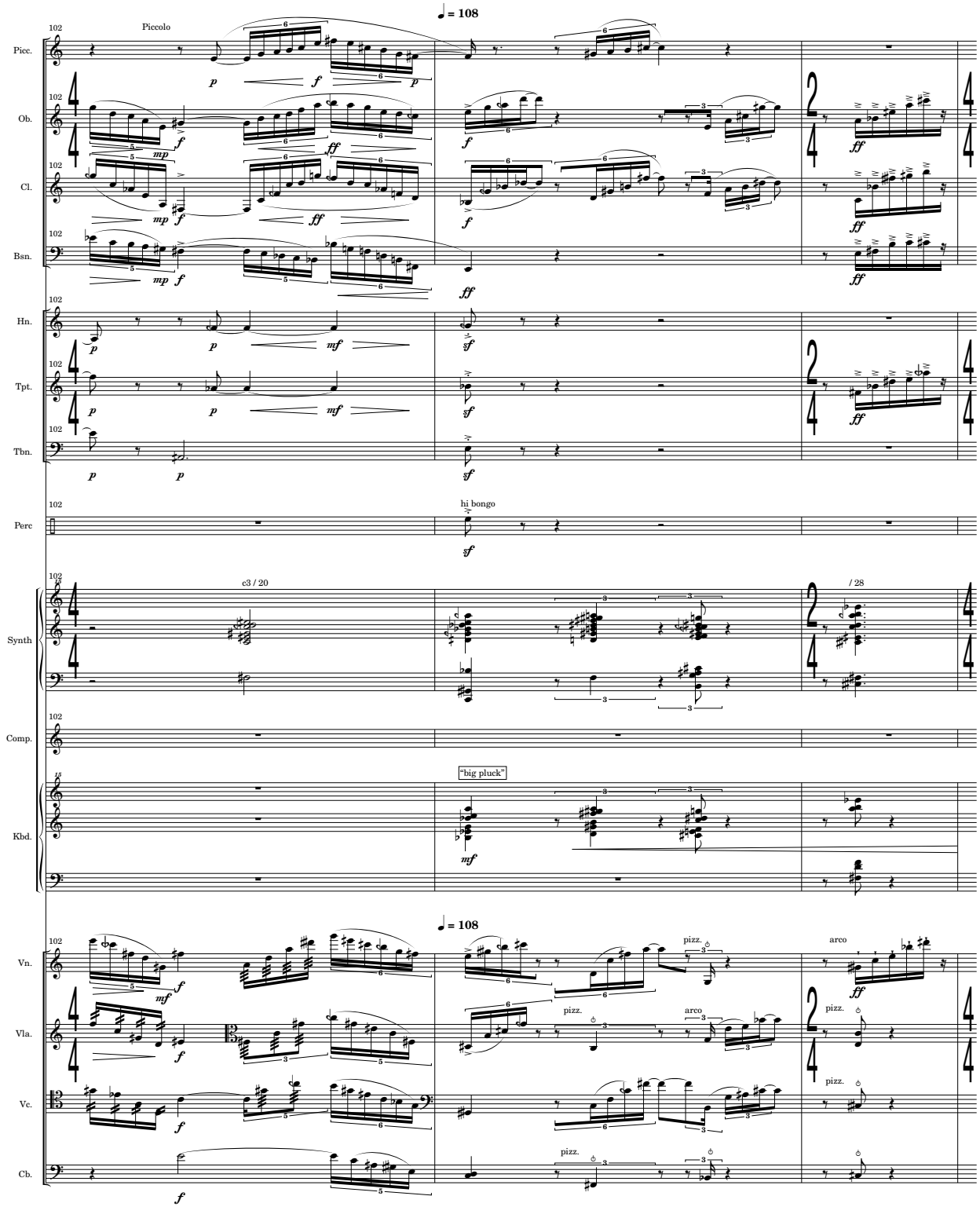

Fl.
Ob.
Cl.
Bsn.
Hn.
mf
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.

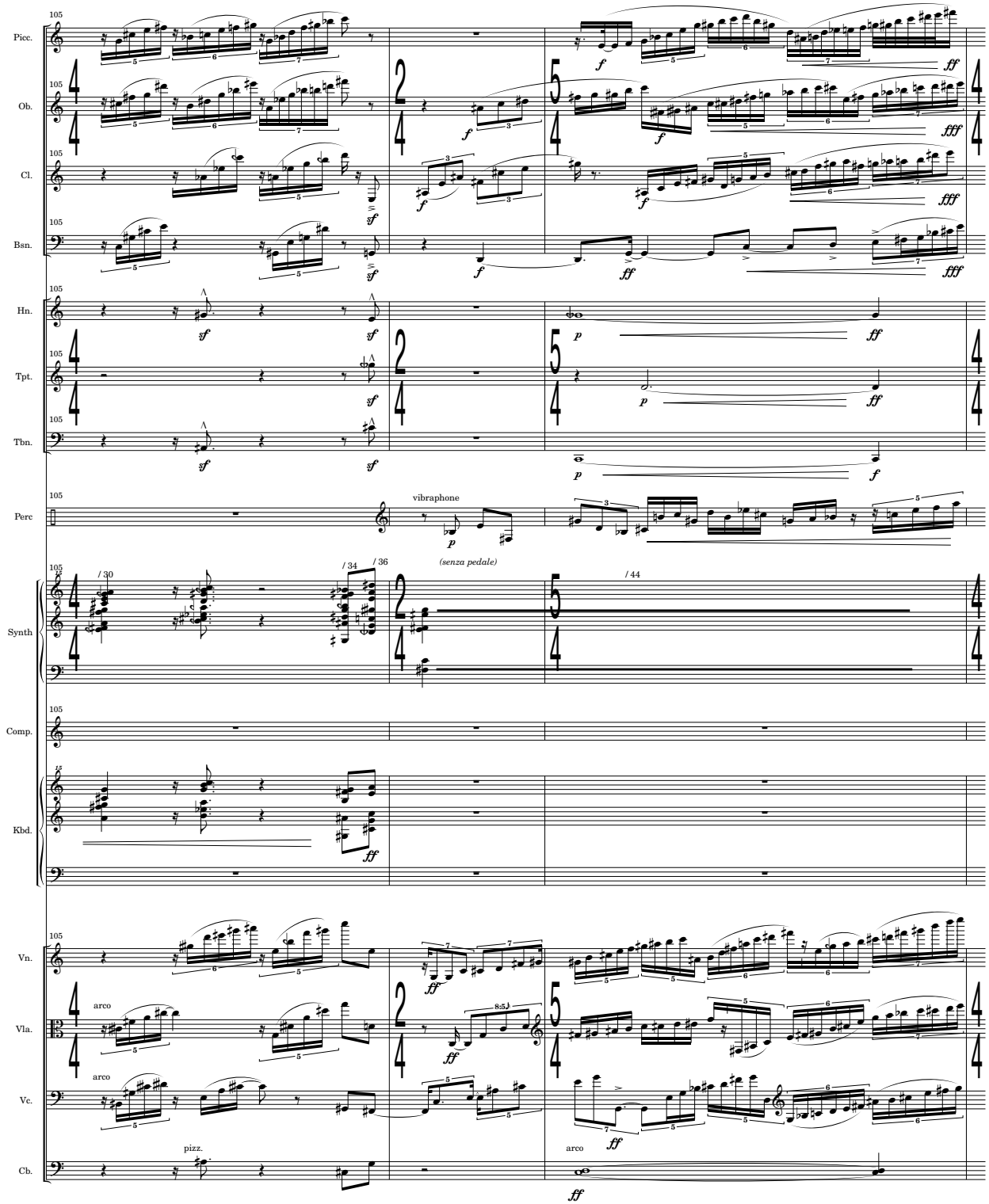

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
87
cresc…
mf
f

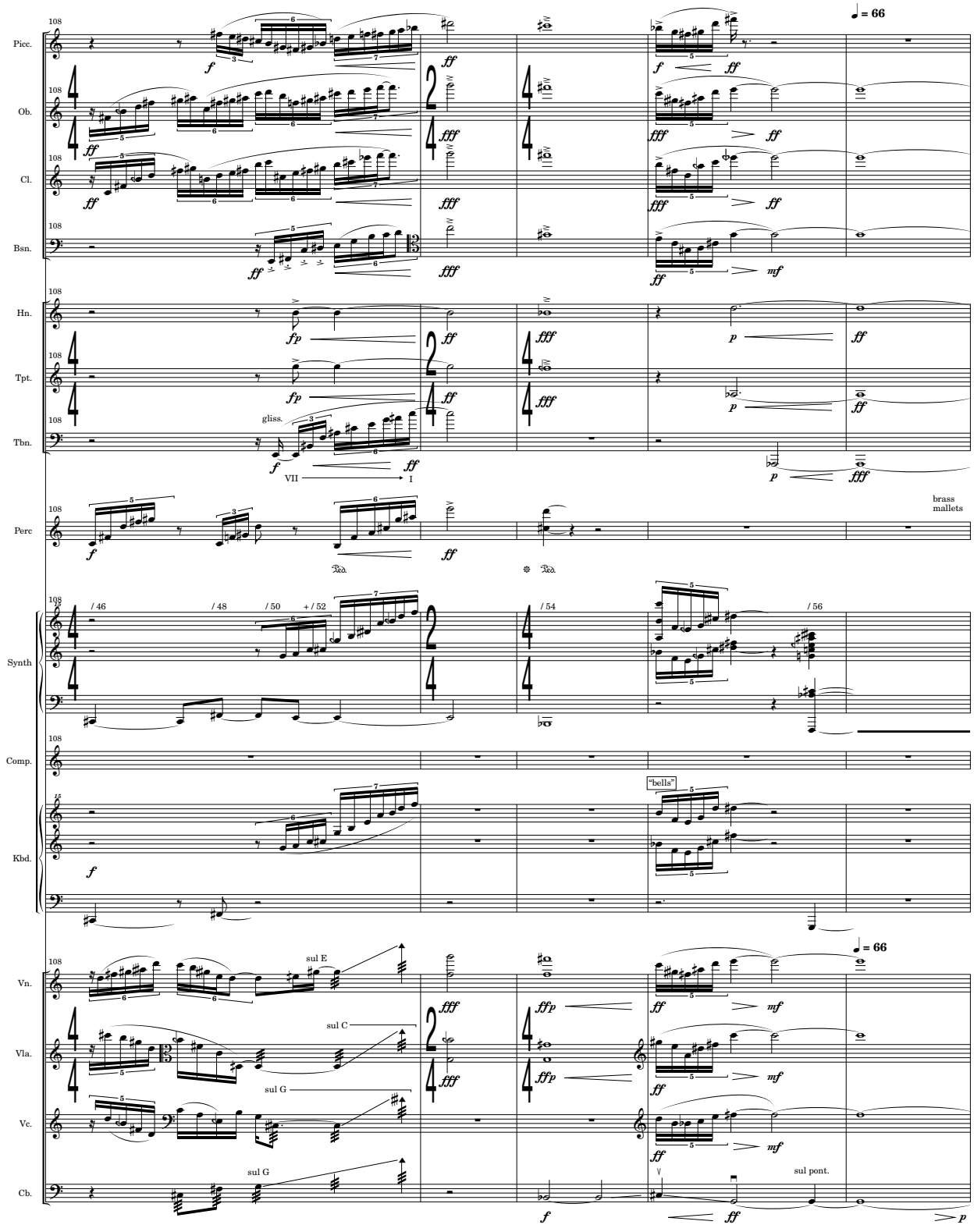

♩ = 96
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
sus. cym.
Synth
/ 152
/ 156
/ 160
/ 164
noise wave
Comp.
Kbd.
"big pluck"
♩ = 96
Vn.
Vla.
Vc.
Cb.

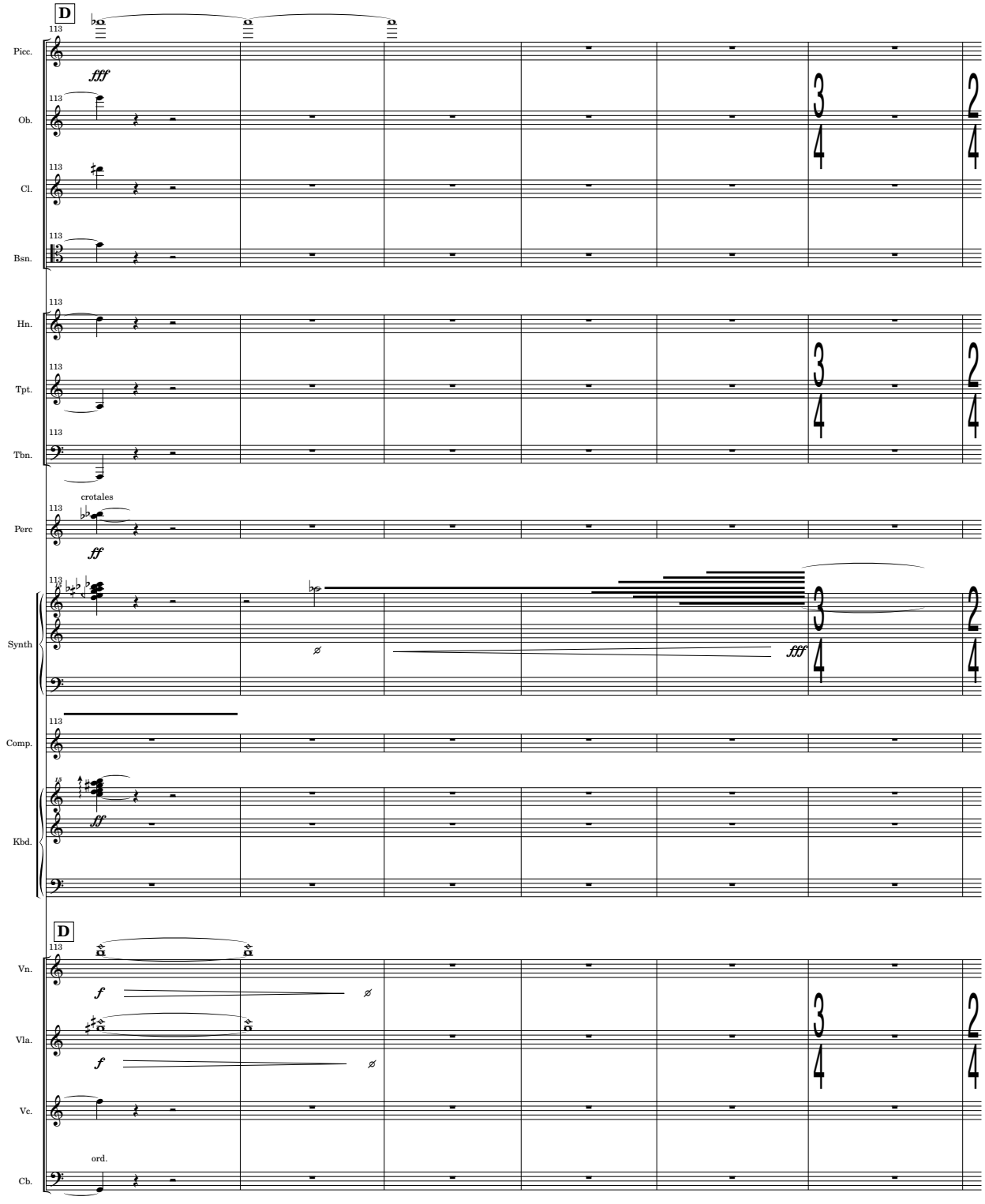

take Piccolo
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
low conga
Chinese cym.
hi bongo
/ 168
Synth
Comp.
"deep synth"
Kbd.
Vn.
Vla.
Vc.
Cb.

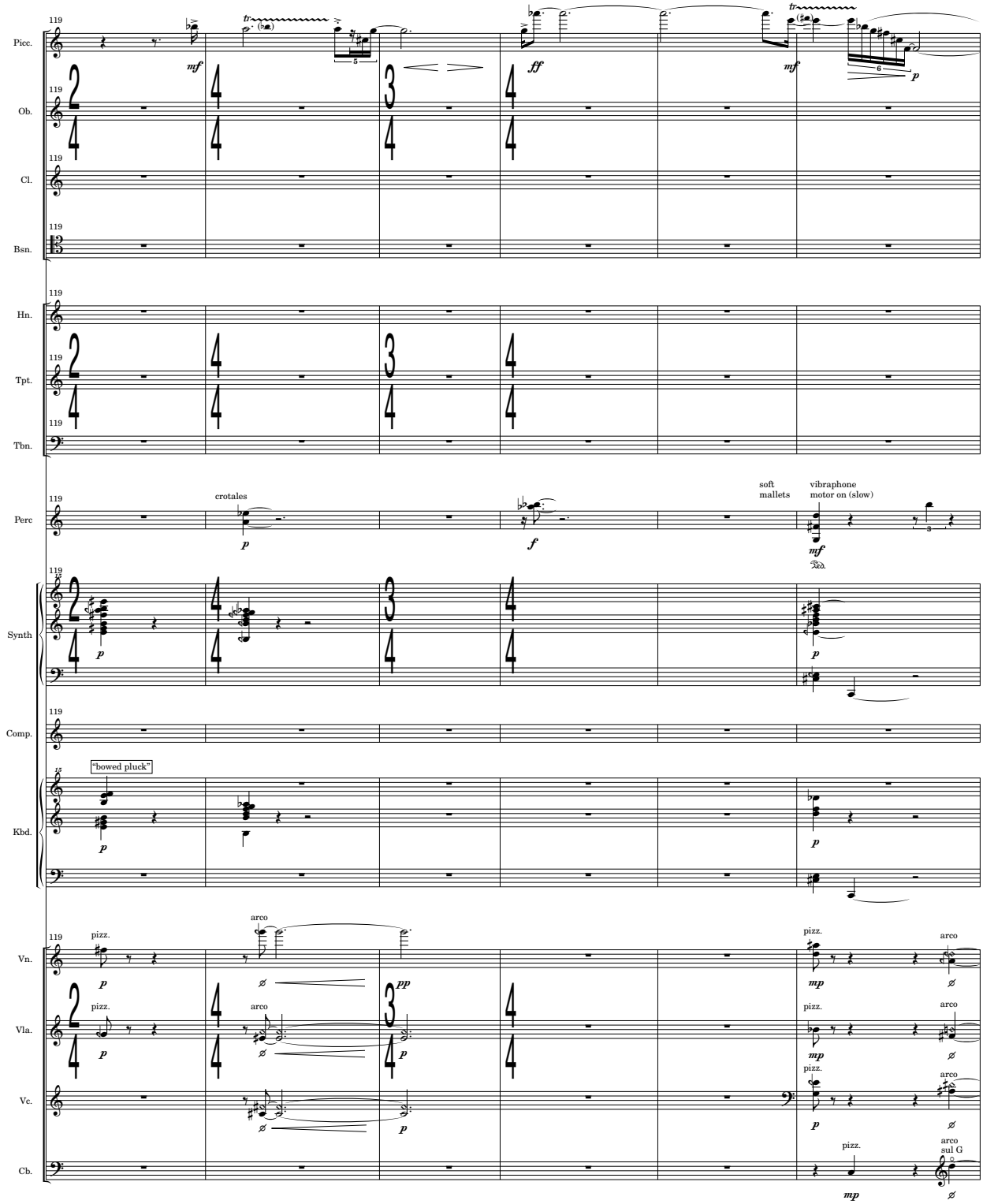

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
low bongo
temple blocks
(senza dim.)
Synth
c1 / 12
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.

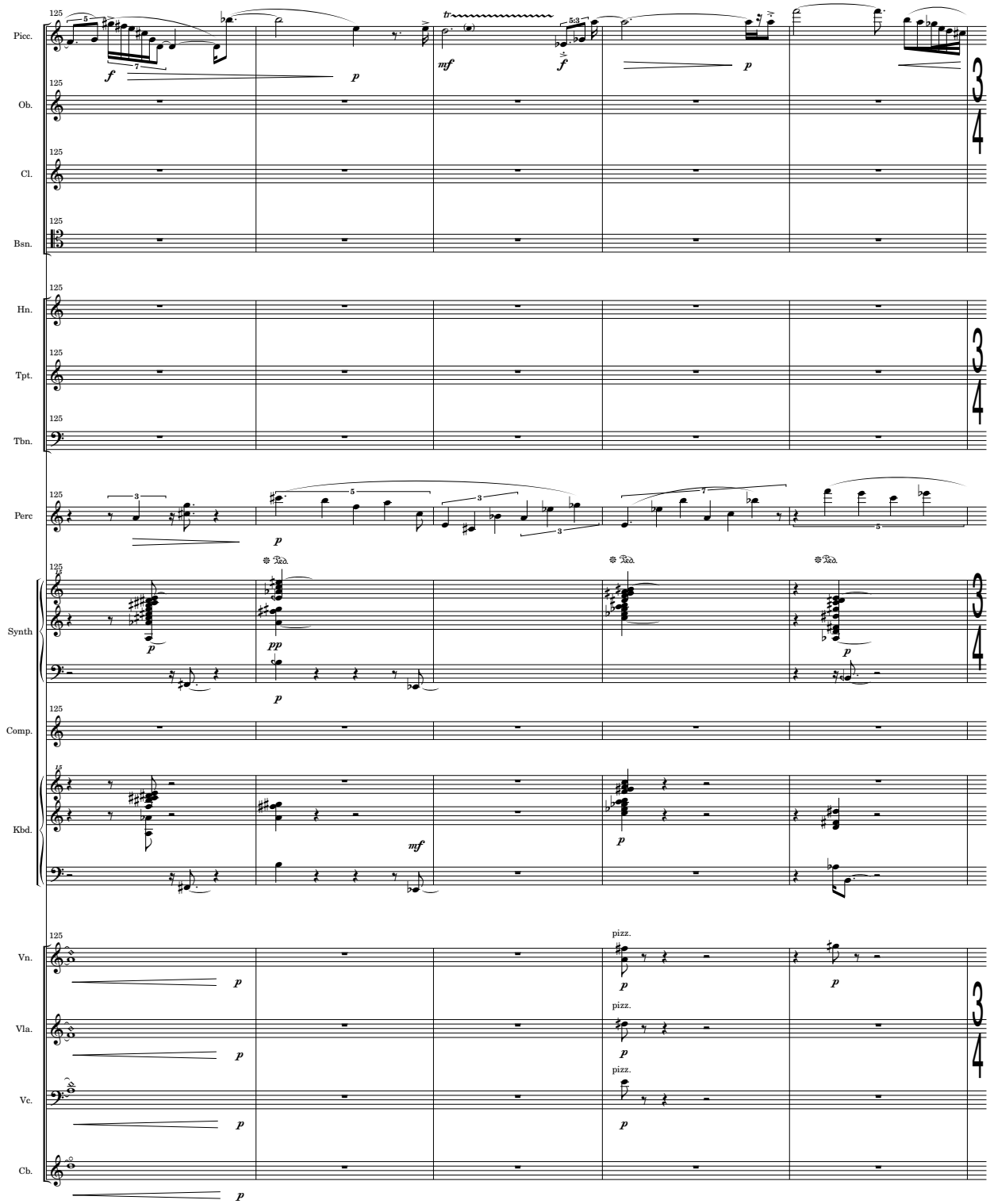

♩ = 108
Piccolo
Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
hi bongo
Synth
c3 / 20
/ 28
Comp.
Kbd.
"big pluck"
♩ = 108
Vn.
Vla.
Vc.
Cb.
pizz.
arco

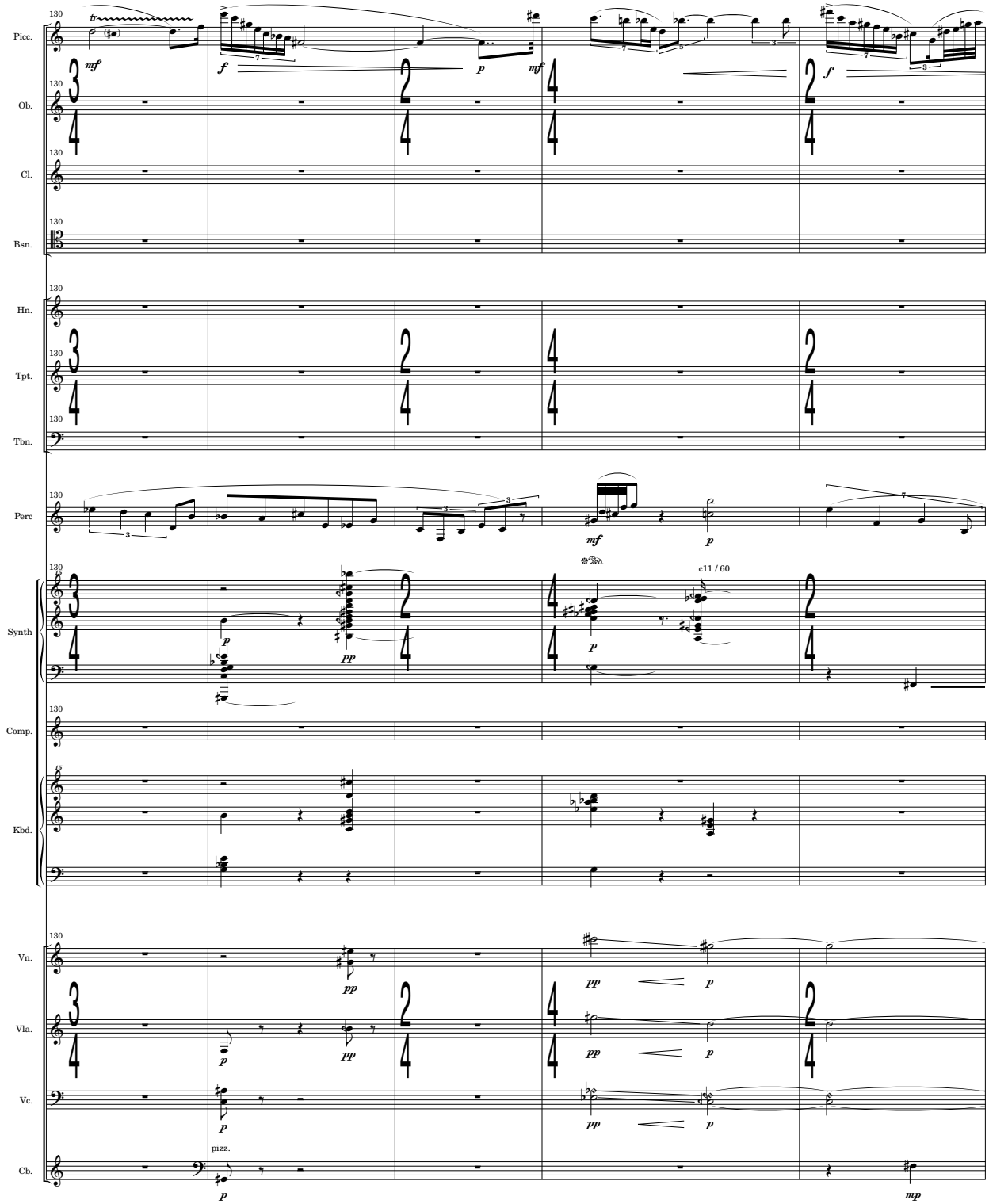

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
vibraphone
Synth
(senza pedale)
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
arco
pizz.
/ 30
/ 34
/ 36
/ 44

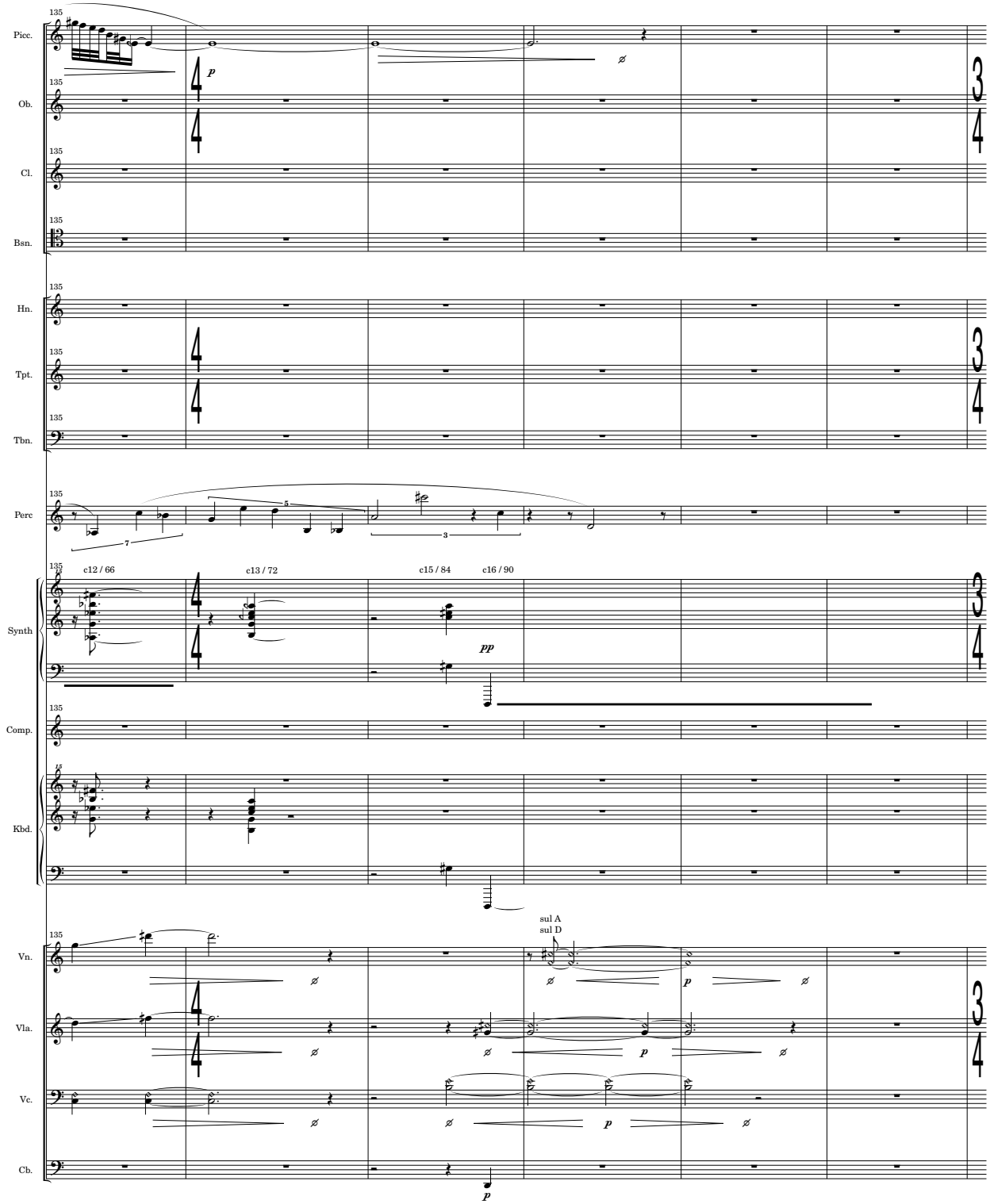

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
brass
mallets
Synth
Comp.
Kbd.
"bells"
Vn.
Vla.
Vc.
Cb.
sul E
sul C
sul G
sul G
sul pont.
gliss.
VII
I
/ 46
/ 48
/ 50
+ / 52
/ 54
/ 56
= 66
= 66

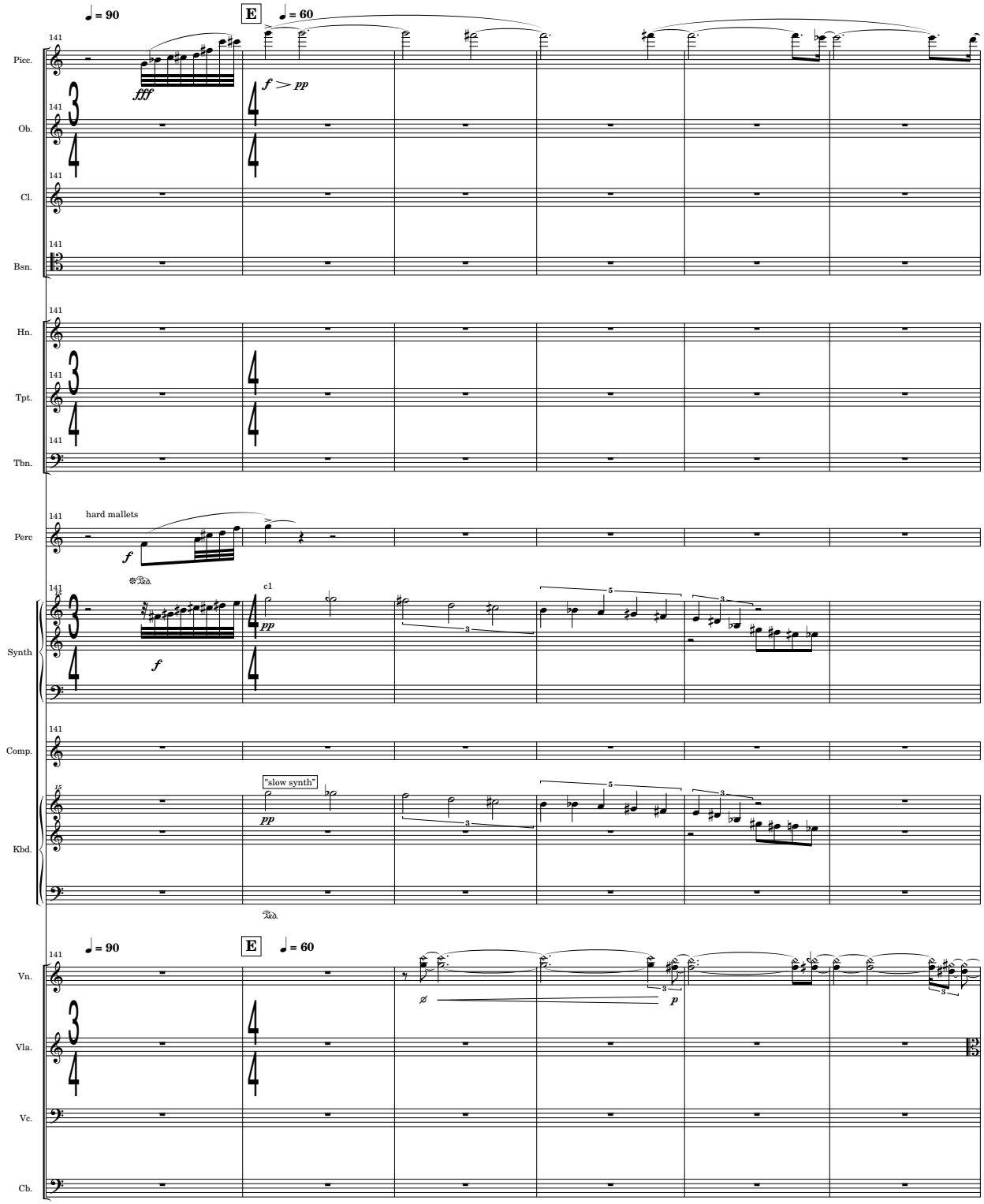

D
Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
crotales
Perc
Synth
Comp.
Kbd.
D
Vn.
Vla.
Vc.
ord.
Cb.

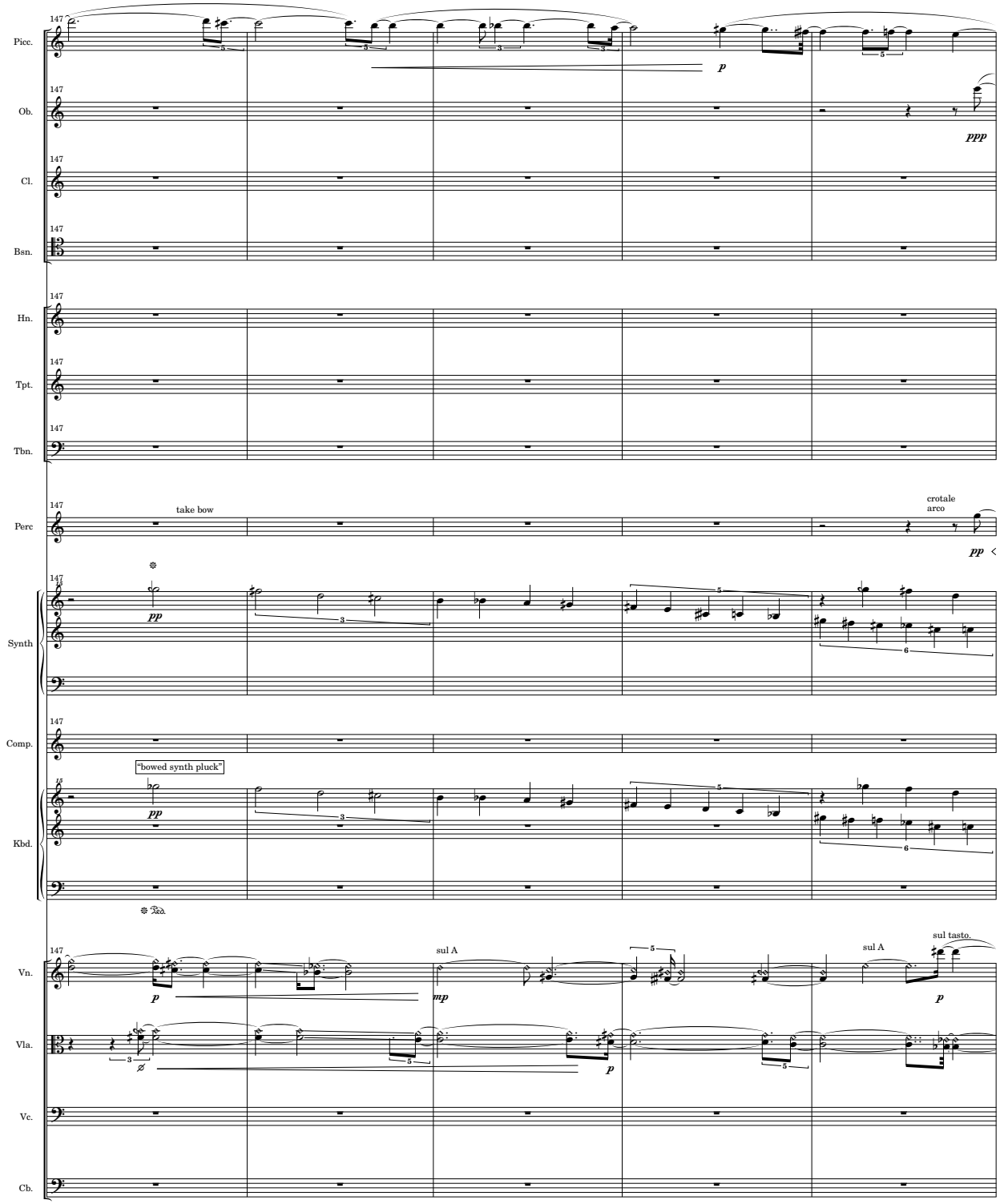

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
crotales
soft mallets
vibraphone motor on (slow)
Synth
Comp.
"bowed pluck"
Kbd.
Vn.
Vla.
Vc.
Cb.
pizz.
arco
sul G

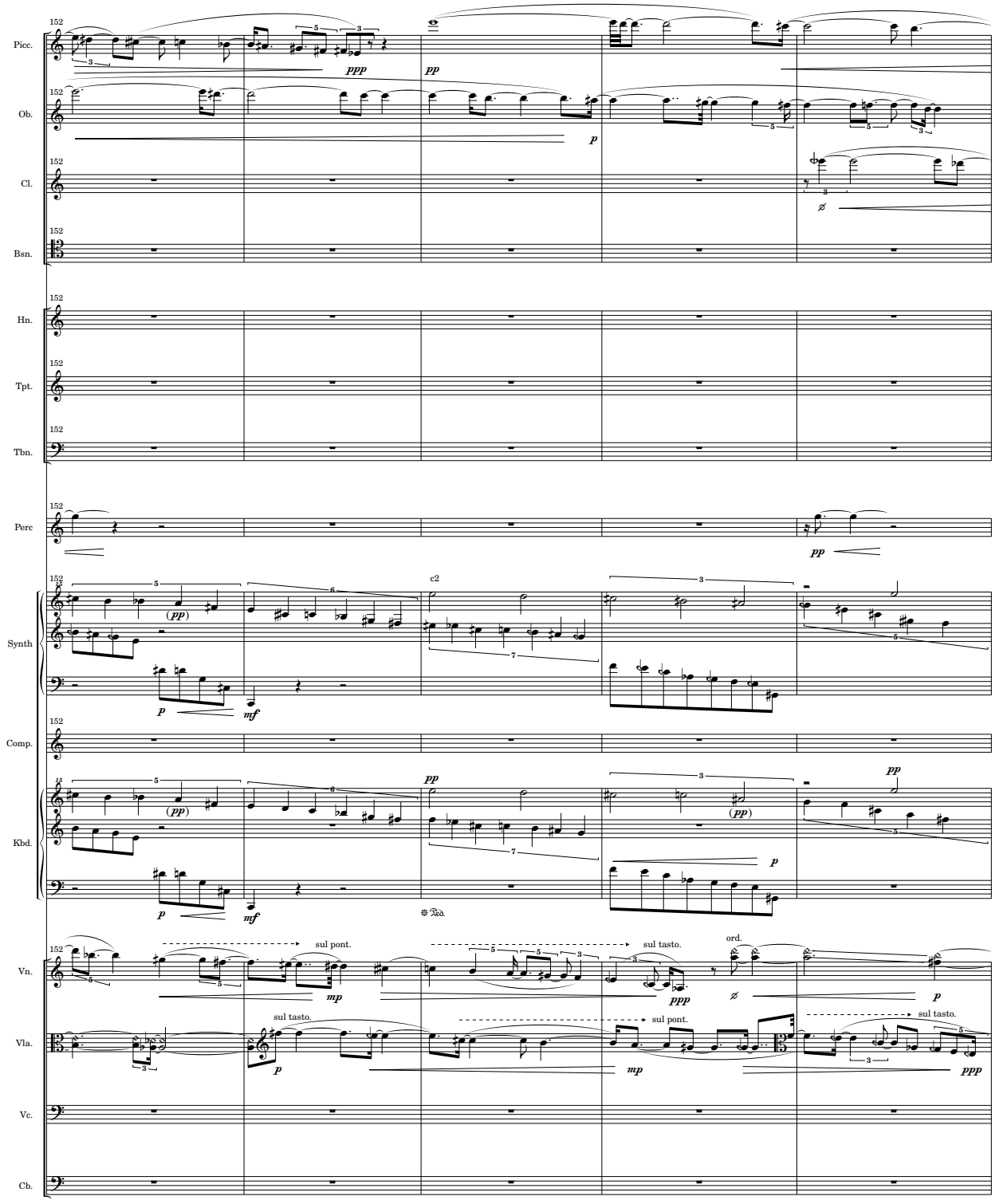

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
pizz.

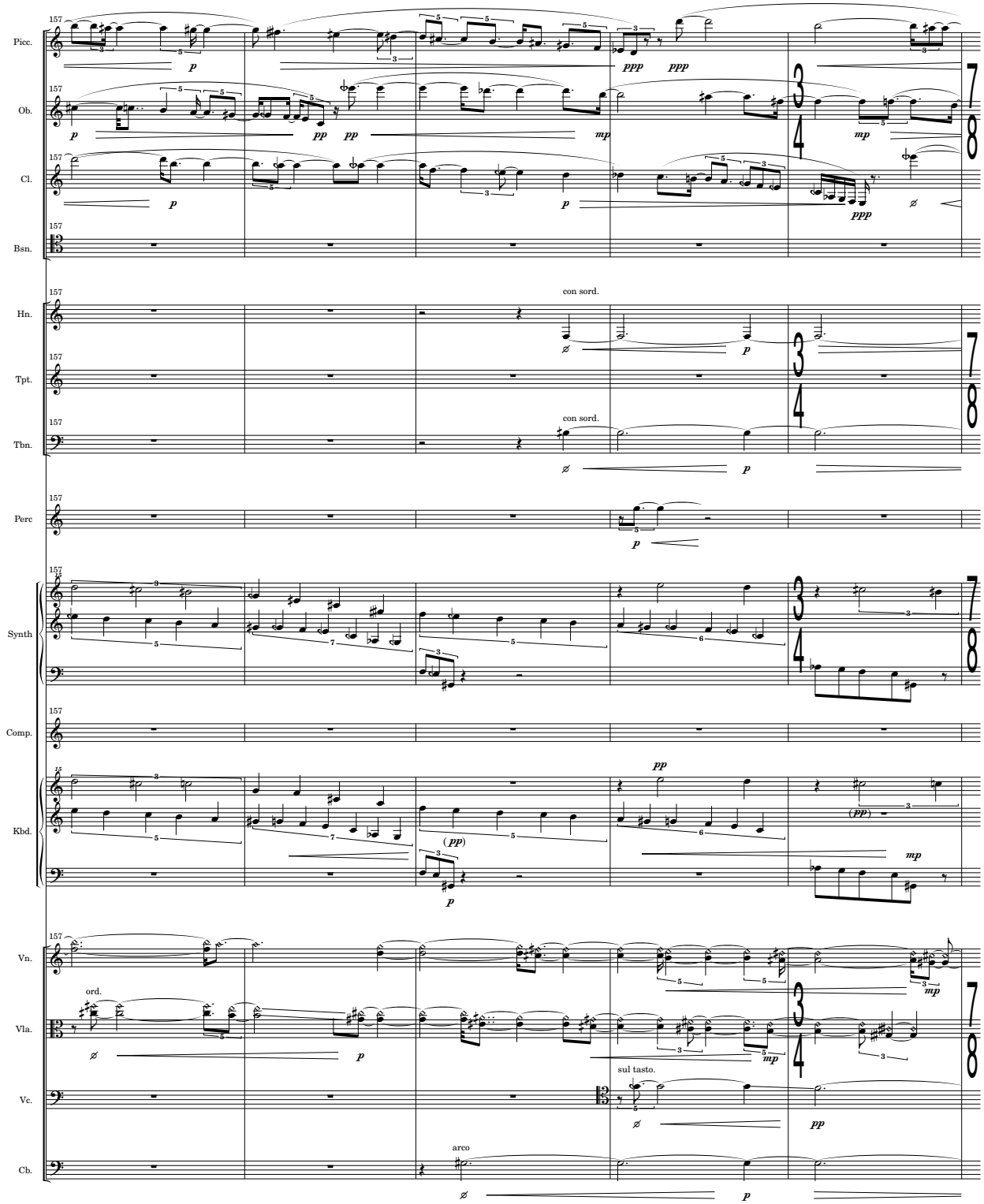

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
c11 / 60
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
pizz.

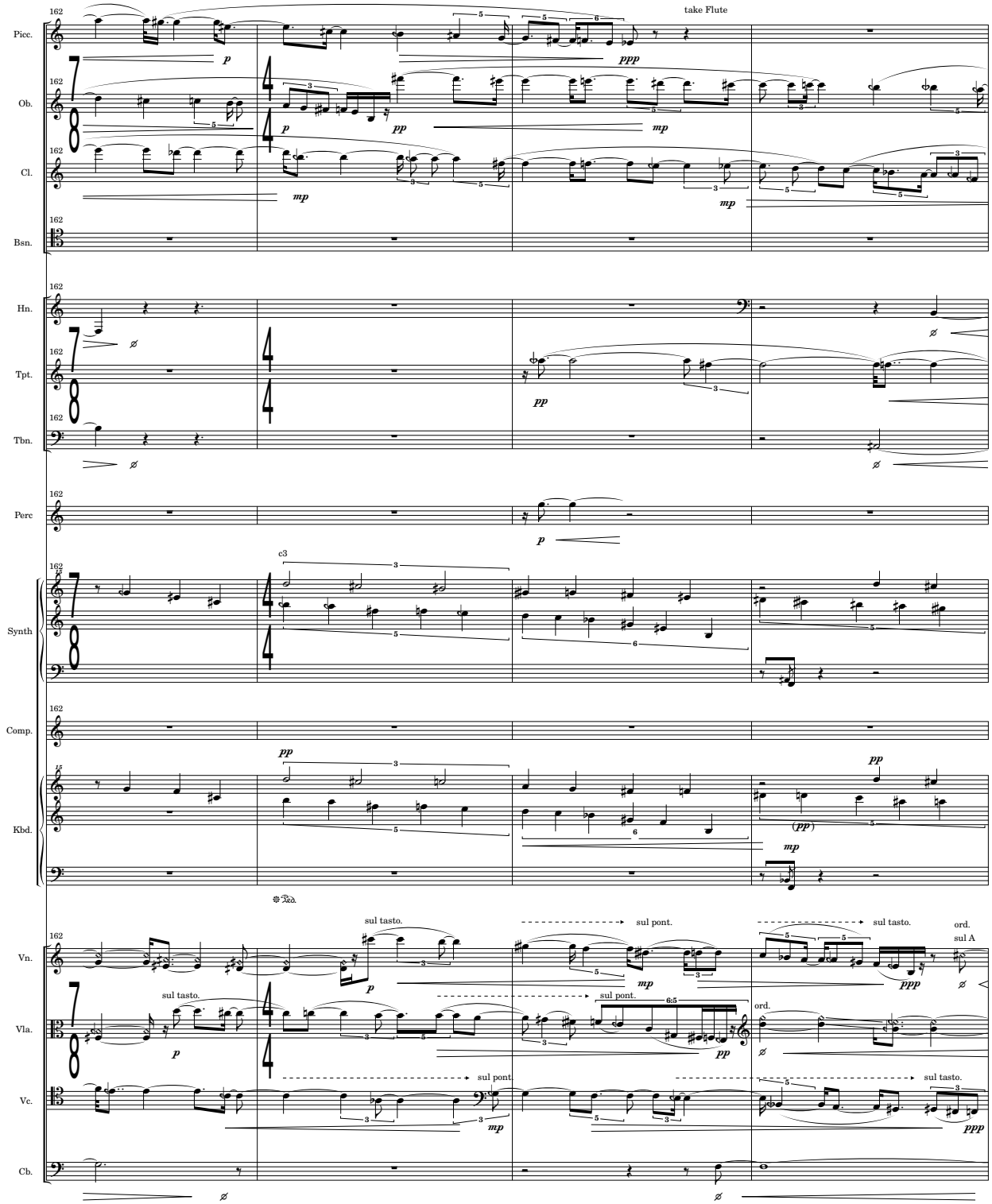

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
135
c12 / 66
c13 / 72
c15 / 84
c16 / 90
sul A
sul D
p
pp

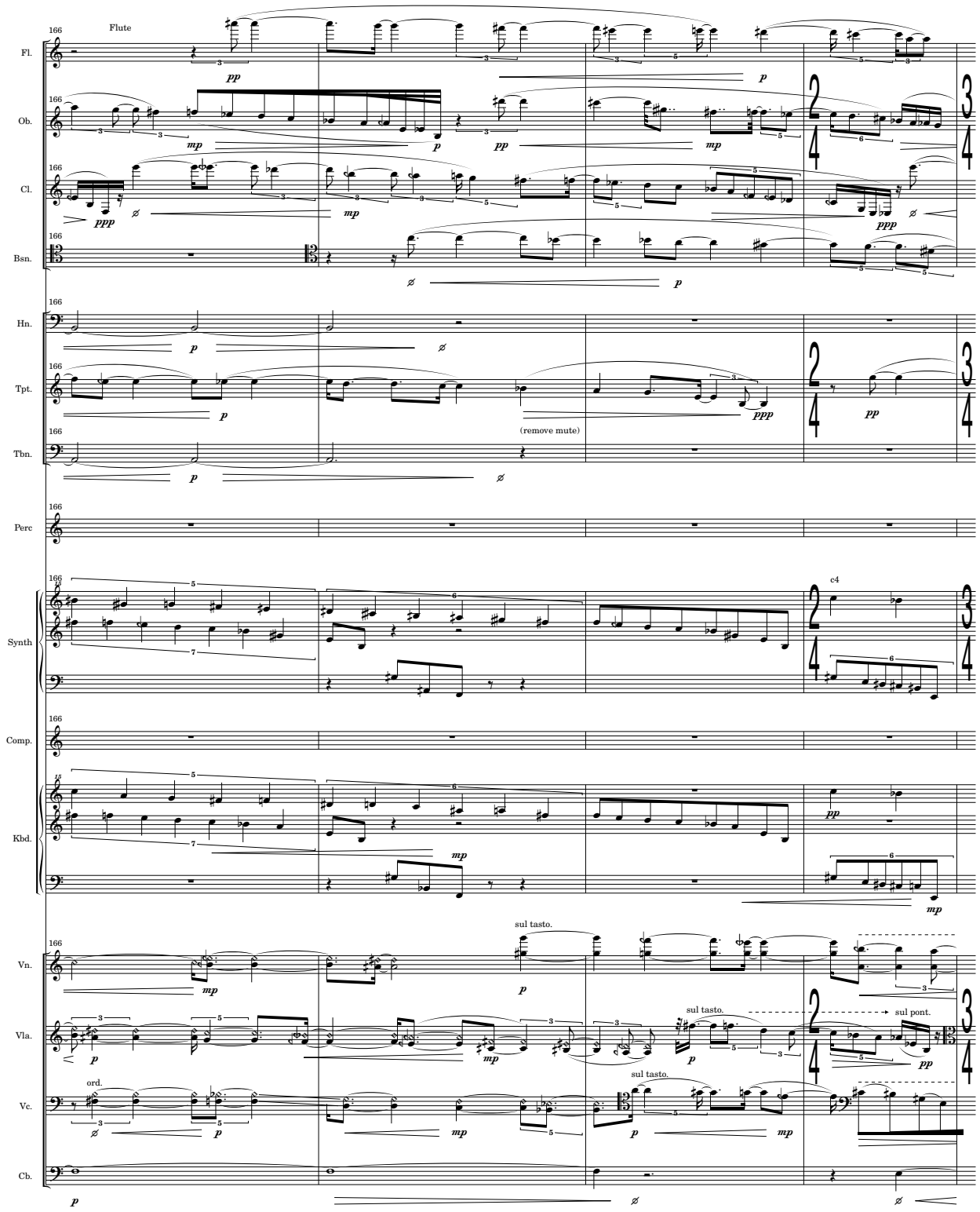

♩ = 90
E
♩ = 60
Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
hard mallets
Synth
Comp.
Kbd.
"slow synth"
Vn.
Vla.
Vc.
Cb.

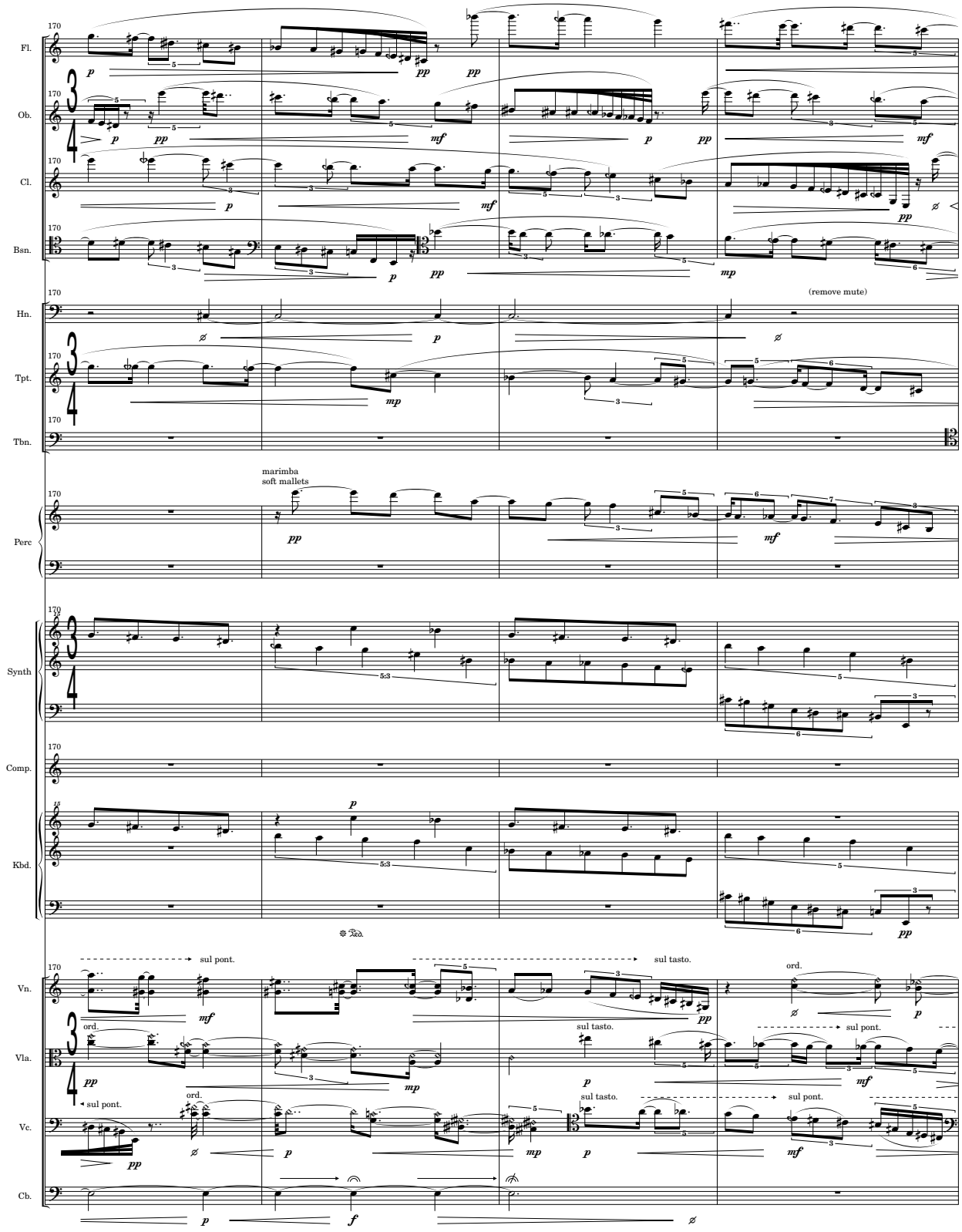

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
take bow
crotale
arco
Synth
Comp.
"bowed synth pluck"
Kbd.
Vn.
sul A
sul A
sul tasto.
Vla.
Vc.
Cb.

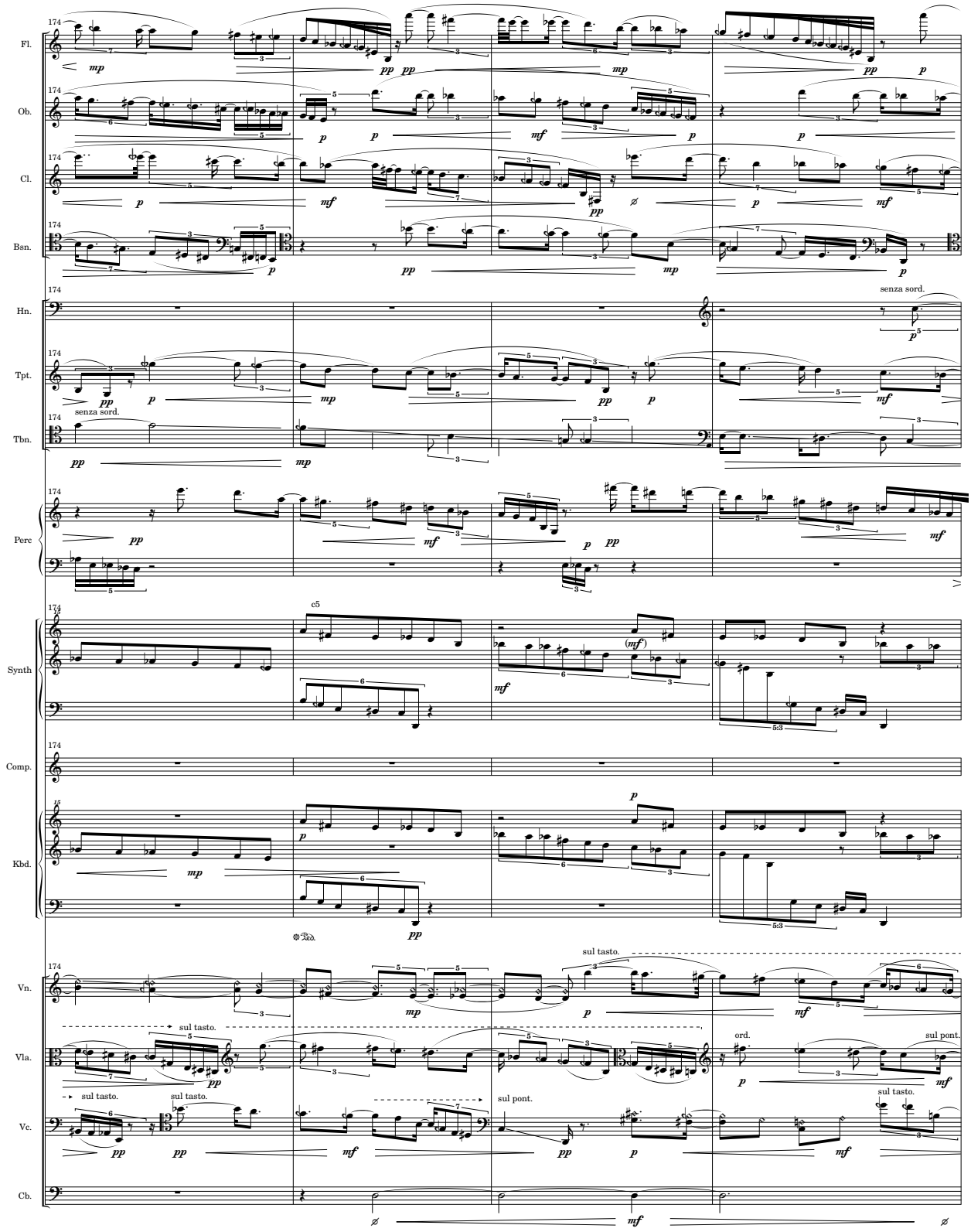

Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
sul pont.
sul tasto.
ord.

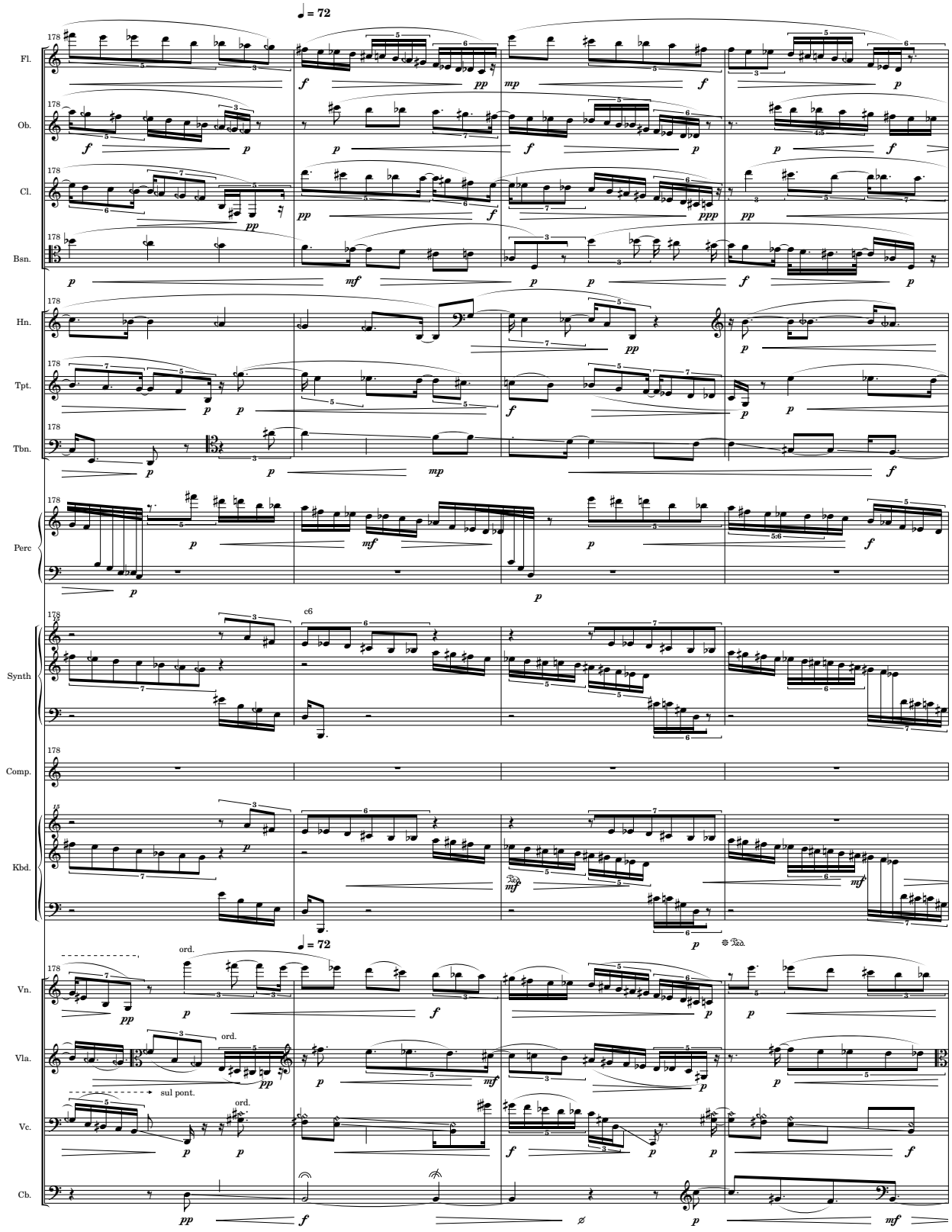

157
Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
con sord.
con sord.
ord.
sul tasto.
arco
3/4
7/8

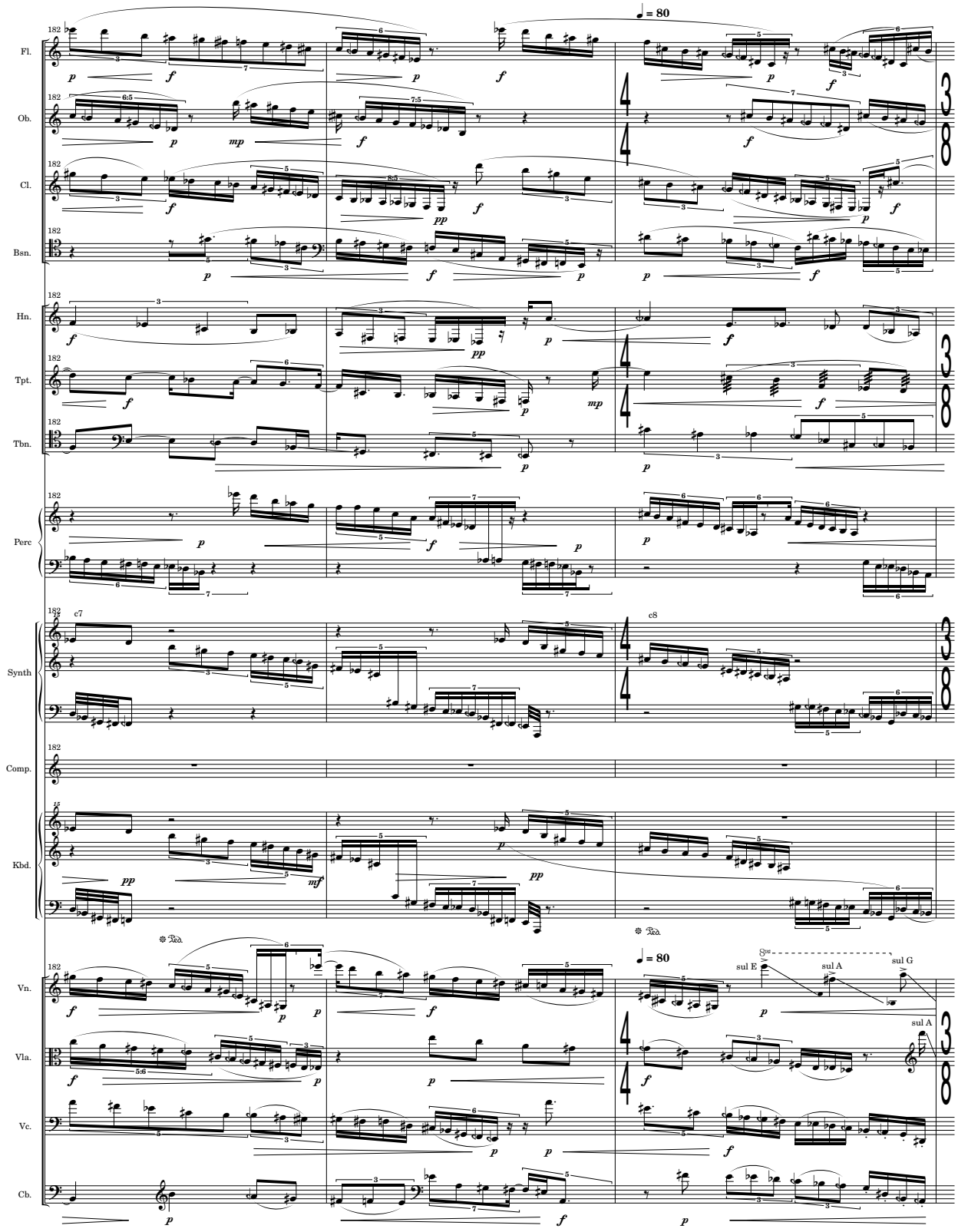

take Flute
Picc.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
sul tasto.
sul pont.
ord.
sul A

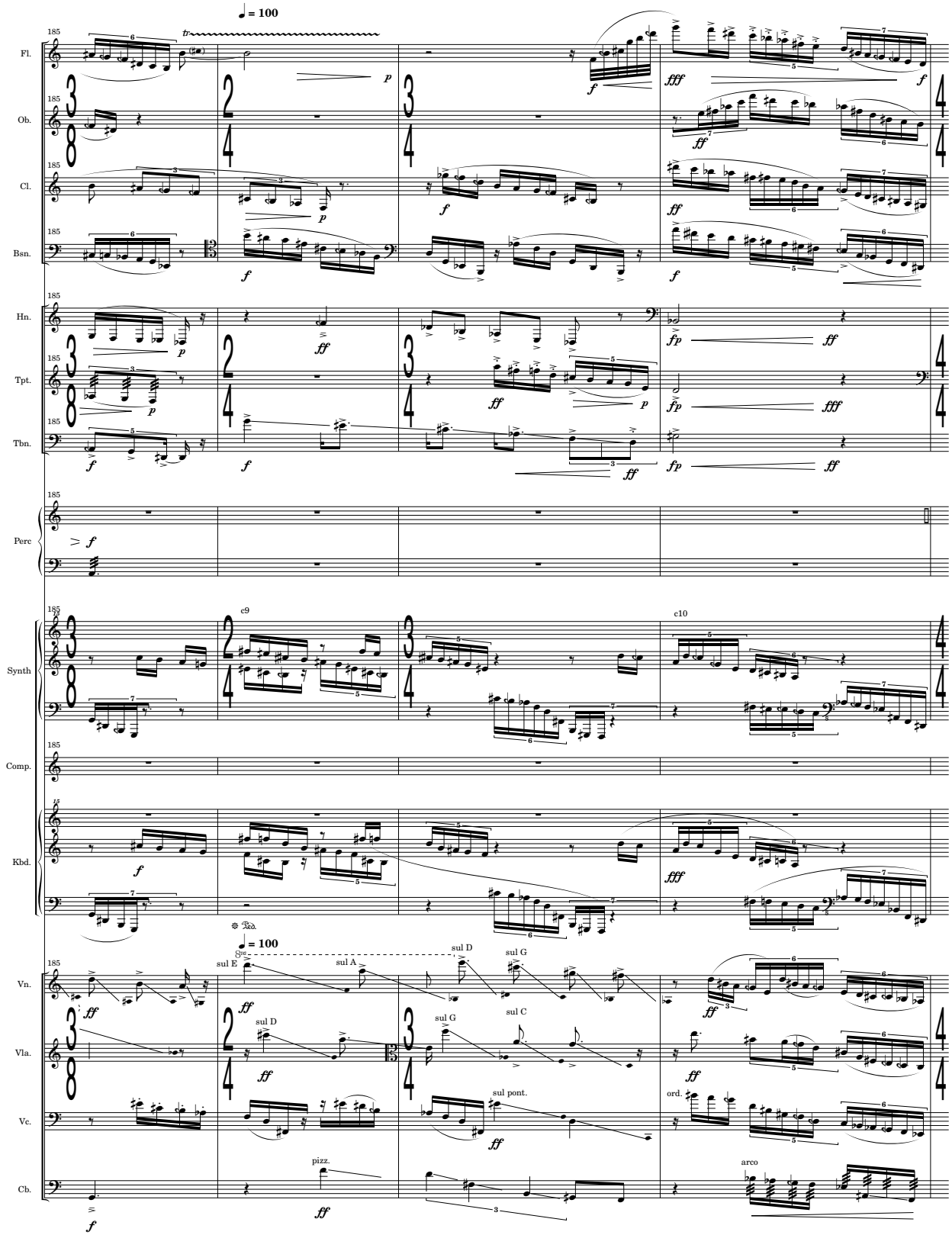

166
Flute
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
(remove mute)
Tbn.
Perc
Synth
c4
Comp.
Kbd.
Vn.
sul tasto.
Vla.
sul pont.
ord.
Vc.
Cb.

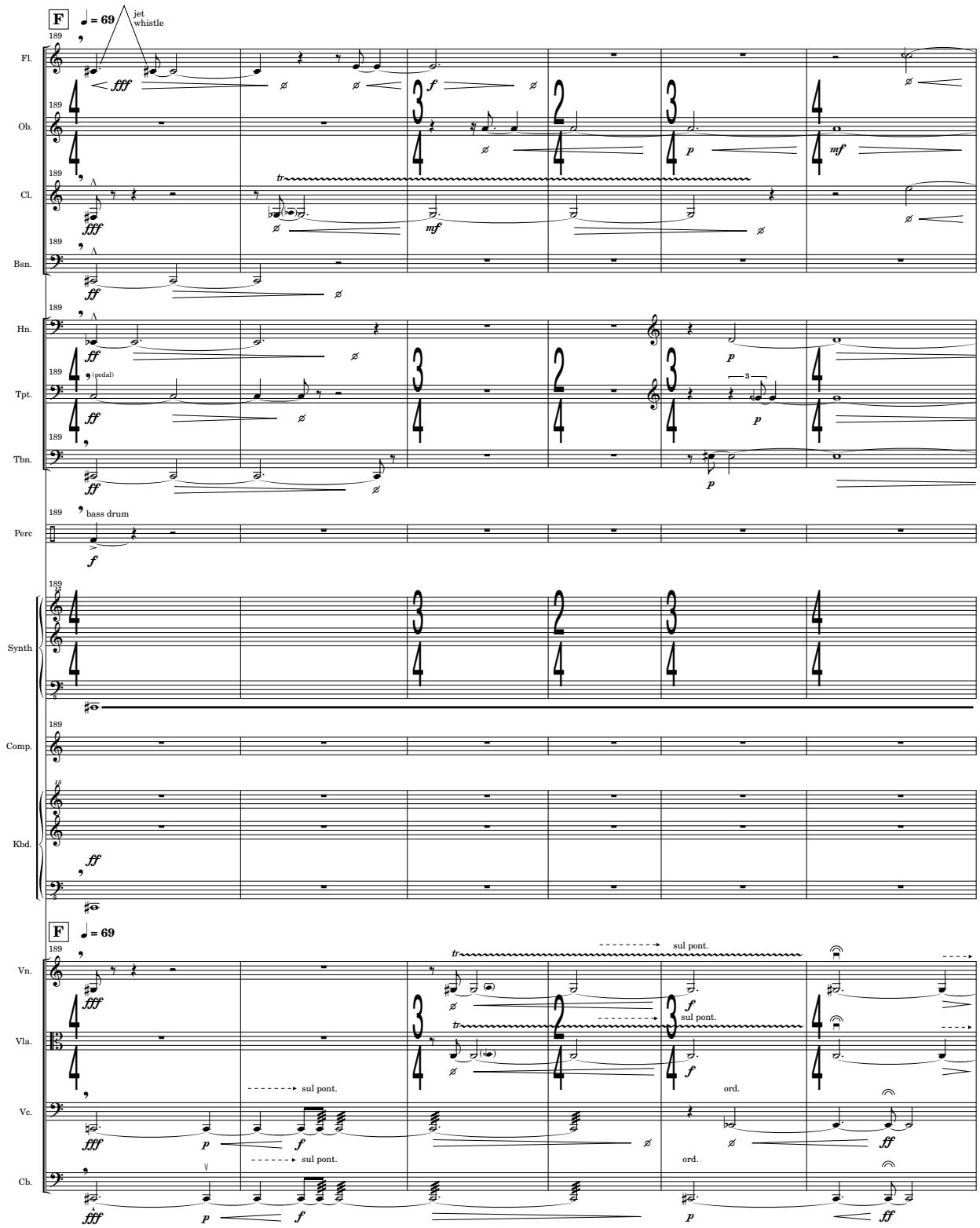

Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
170
(remove mute)
marimba
soft mallets
sul pont.
sul tasto.
ord.

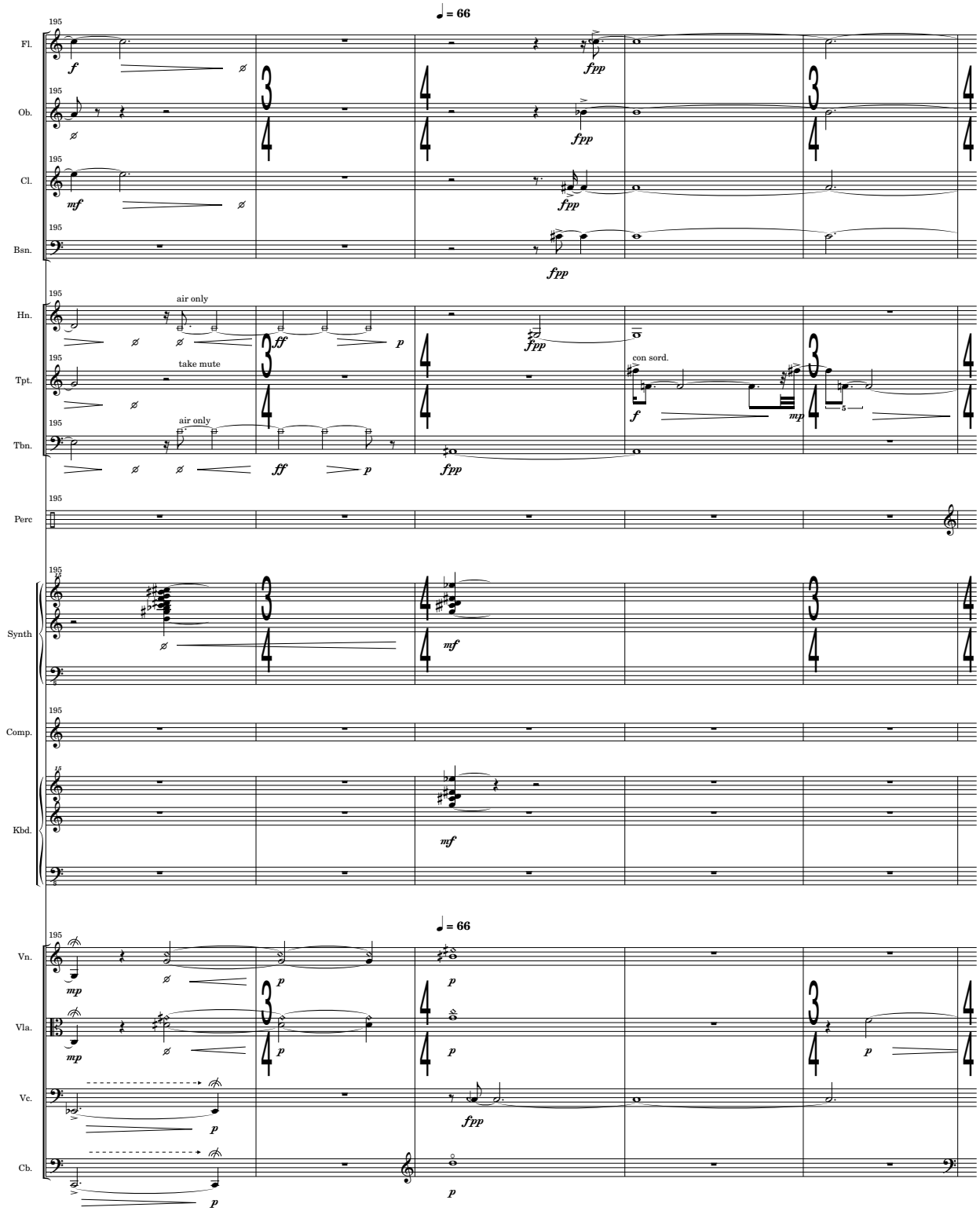

174
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
senza sord.
c5
sul tasto.
ord.
sul pont.

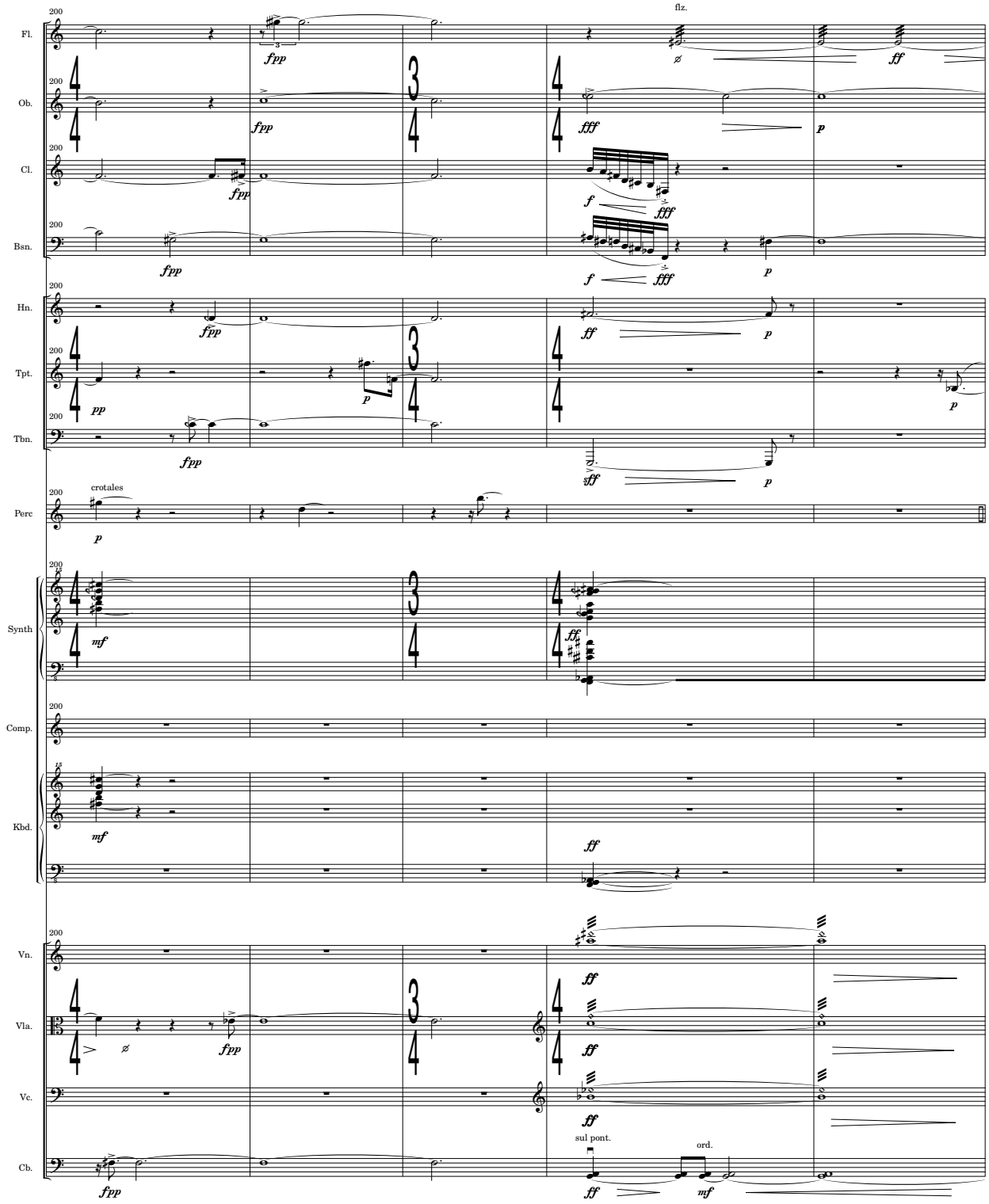

♩ = 72
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
c6
ord.
sul pont.

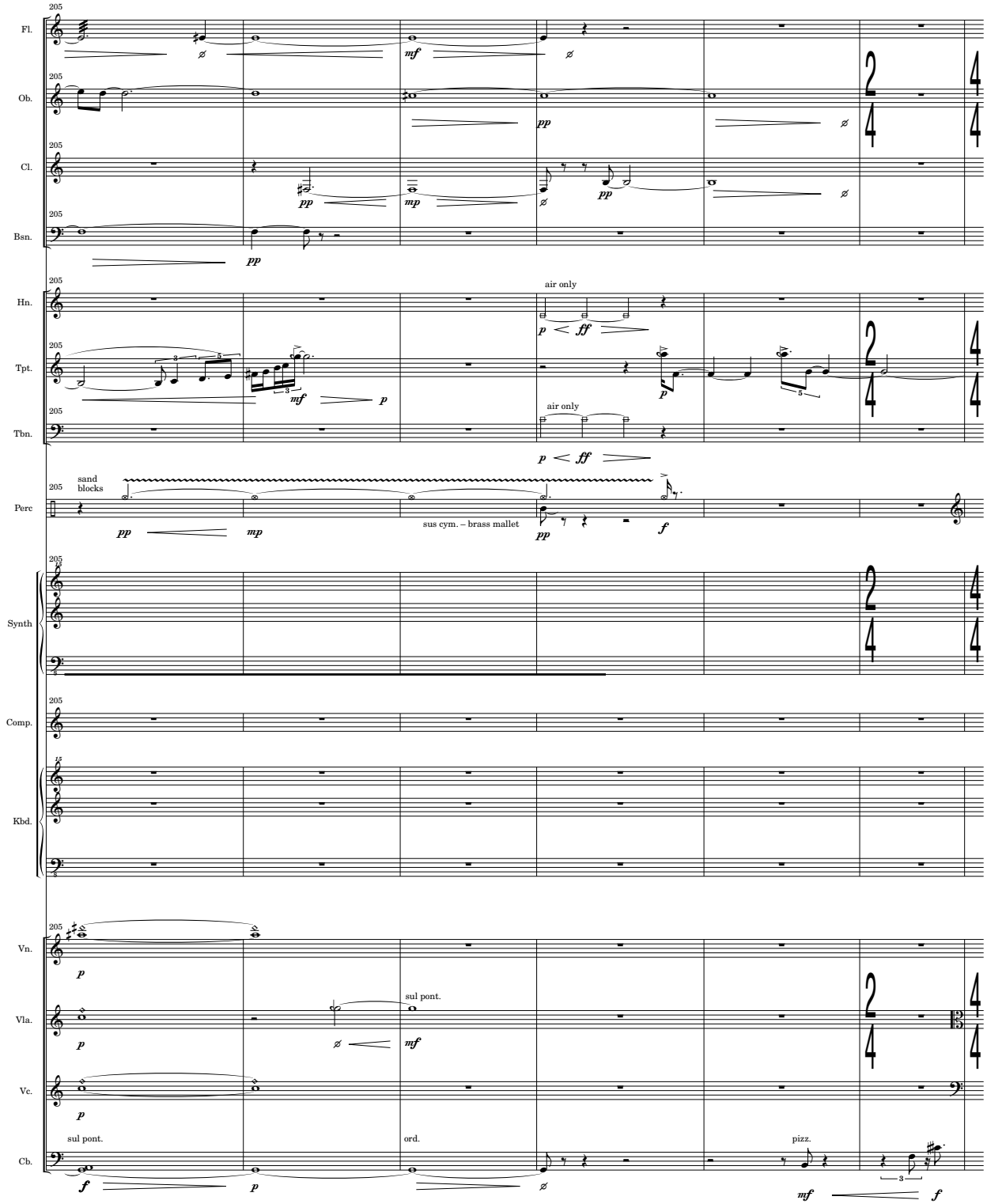

182
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
♩ = 80
sul E
sul A
sul G
sul A
c7
c8
Ped.

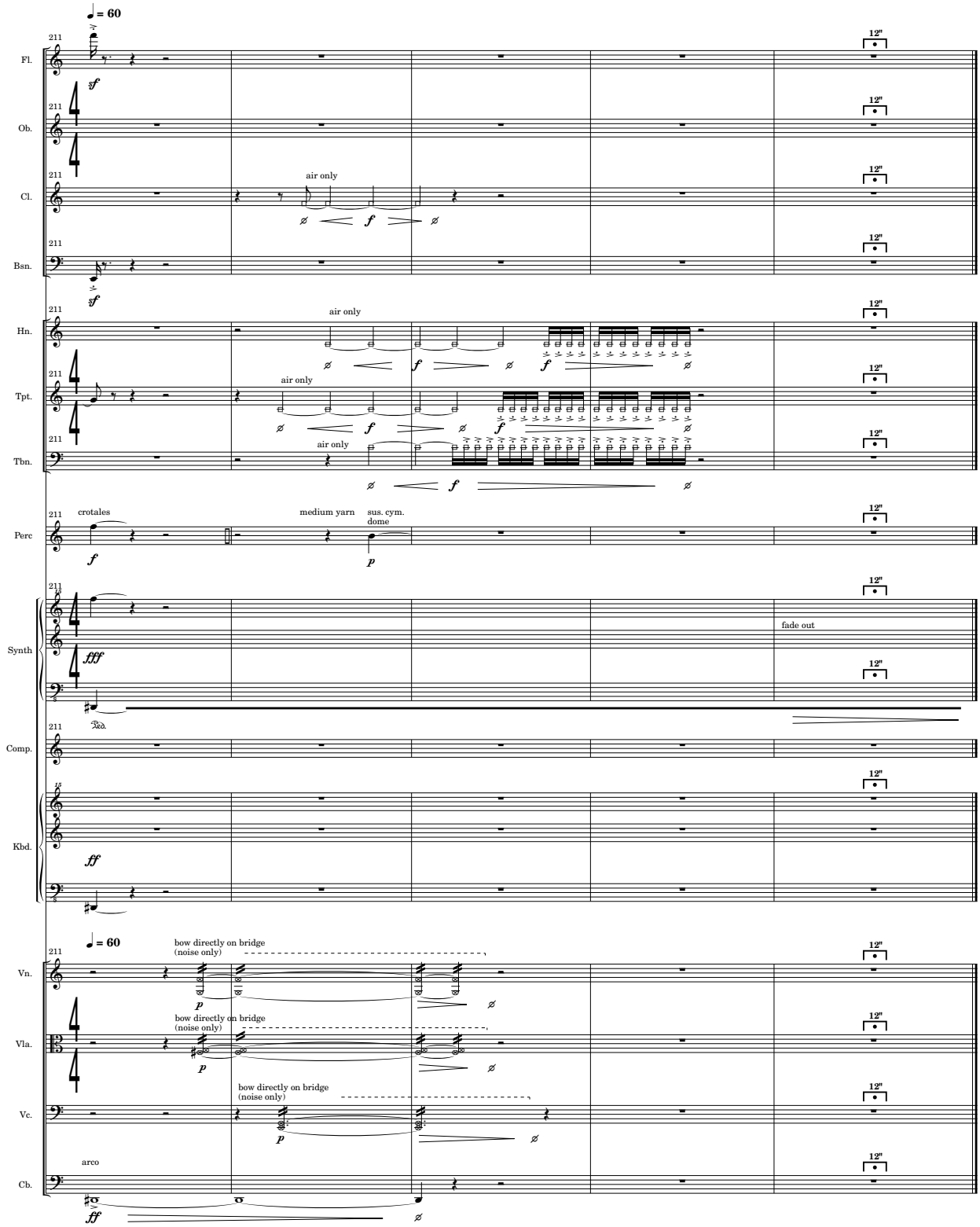

♩ = 100
185
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
c9
c10
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
sul E
sul A
sul D
sul G
sul C
sul pont.
pizz.
arco
ord.


F
♩ = 69
jet whistle
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
(pedal)
Tbn.
bass drum
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
sul pont.
ord.


♩ = 66
195
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
air only
take mute
air only
con sord.
f
mf
mp
p
ff
fpp
fpp
fpp
fpp
fpp
mf
mf
mp
p
p
p
p
p
p
fpp
p
p


Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
crotales
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
flz.
sul pont.
ord.


205
Fl.
Ob.
Cl.
Bsn.
Hn.
Tpt.
Tbn.
Perc
Synth
Comp.
Kbd.
Vn.
Vla.
Vc.
Cb.
air only
sand blocks
sus cym. – brass mallet
sul pont.
ord.
pizz.



March 14, 2008
New Haven, Connecticut