MUSICAL USE OF A GENERAL AND EXPRESSIVE PLUCKED-STRING INSTRUMENT IN SOFTWARE

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Musical Arts in the Graduate School of the Ohio State University

By

James Michael Croson, B.A., B.M., M.A.

The Ohio State University

2004

Dissertation Committee:

Professor Thomas Wells, Co-Advisor

Professor Donald Harris, Co-Advisor

Professor Burdette Green

Approved by

_____

Co-Advisor

_____

Co-Advisor

School of Music

ABSTRACT

Plucked string instrument models have enjoyed an intensive history of development in computer music, offering novel methods for synthesis and ever-better simulations of actual acoustic instruments. Research in acoustics has improved our understanding of the physics of musical instruments, and made available to composers and performers more natural sounding computer-based instruments. This paper outlines issues in the development of plucked string software models, emphasizing the opportunities for expressive musical use of these models. A general plucked string instrument emphasizing maximum expressive control is presented in implementations for two software platforms, Max/MSP and Csound. *Nukulele,* the instrument model, couples four plucked strings together for exploration of many control parameters including the effects of using different excitation impulses, dynamic function control of some parameters, resonance between the strings, and feedback effects. The instrument model is implemented two ways: one in Max/MSP for rapid prototyping and exploration, and the other for use in Csound, a score-based, repeatable, and adjustable system for making music. The effects of varying available parameter settings are illustrated by audio examples, and I offer guidance for their exploration and use. Appendices provide the Max/MSP *Nukulele* application, a Csound orchestra and score file template, and a Max/MSP application to convert data from the Max/MSP prototyping implementation to

Csound score format for high quality and repeatable rendering. This document is aimed at musicians and educators and emphasizes practical musical use: the mathematical underpinnings and theory reside in the cited literature.

DEDICATION

Dedicated to my inspiration and partner, Sherie Lindamood.

ACKNOWLEDGMENTS

I thank my advisors and instructors at Ohio State University, especially Professor Donald Harris, Dr. Thomas Wells, Dr. Burdette Green, and Dr. Marc Ainger for their example, support, encouragement, knowledge, and sensibilities, which I appreciate now and will long value.

I thank Dr. Allan Schindler for inspiration and my initiation into computer music composition. Many others at the Eastman School of Music gave so much in classes and lessons; I especially thank Syd Hodkinson.

I think now of my students, performers, collaborators, and colleagues throughout my studies, who shaped the character of my experience.

VITA

July 11, 1953 ..... Born - El Paso, Texas, USA.

1982 ..... B.A. Photography and Cinema, Ohio State University.

1992 ..... B.M. Jazz Studies, Ohio State University.

1994 ..... M.A. Music Theory, Eastman School of Music.

2001 - present ..... Graduate Teaching Associate, The Ohio State University.

FIELDS OF STUDY

Major Field: Music

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF AUDIO EXAMPLES

CHAPTER 1

INTRODUCTION

Composers using computers have long struggled against the limitations of this instrument for expressiveness. Precision, novel sounds, exactitude, and repeatability come easily, but music that sounded natural and expressive came only with effort. Designing expressiveness into an instrument model is only half of the toil: controlling the model can be difficult too. After all, acoustic instruments develop and improve over long periods of time along with, and informed by, development and improvement in performance techniques and practice. Computer instruments in software have a relatively shorter history, and the instrument models used typically enjoy an even shorter useful life cycle, often developed, used once, and discarded.

Like many, I often seek computer instrument models that achieve natural and expressive musical effects similar to those we expect of acoustic instruments, while also demanding the medium's precision, range of possible sounds, exactitude, and repeatability. While my aim is rarely exact imitation of existing acoustic instruments, I often desire something of the responsive qualities of acoustic instruments, especially the ability to deliver musical ideas and gestures convincingly and with the expressiveness that can be a primary mode of interest in a composition.

Expressiveness, naturalness, and responsiveness are closely related, and are the aim of what David Beck's calls "acoustic viability," a design aesthetic in synthesis:

> By recognizing the importance of instrument acoustics and its relationship to expression, synthetic instrument designers can build synthesis processes that respond to changes in loudness, pitch, and articulation that are consistent with our understanding of acoustic instruments (Beck 2000).

Responsiveness characterizes the way the output changes when the input is varied. An instrumentalist judges an acoustic instrument based (among others things) on its response; how hard the player has to work to achieve a range of effects in sound, how reliable and predictable are the effects, and how easy they are to control. The focus is on the way details of the sound unfold in time.

Naturalness is the quality by which we judge how an instrument's response corresponds to our intuitions about sounds in the real world, including the human voice and acoustic instruments. Separate qualities of a sound or gesture such as loudness, articulation, timbre, vibrato, etc. need to move together under larger scale variations in a way that does not contradict our intuitions very much, though it should be said that we love surprises.

Expressiveness is conveyed by variations in performance that correspond in a recognizable way to the emotional range that a player might wish to communicate to a listener. A sound or gesture will quickly be judged inexpressive if the correspondence is too static, mechanical, foreign, or non-human. One could say expressive sound is life-

like. Of course, expressiveness is often employed for structural aims such as clarity, contrast, etc, as well as for emotional response. This document focuses on the rather traditional situation where a composer has a sound or gesture in mind, and tries to achieve it with an instrument model in such a way that the expression can be varied to evoke different responses in a listener.

A host of synthesis techniques exist which can similarly benefit from more attention to expanding possibilities for expressiveness. One important category, physical modeling synthesis, is especially promising, since it relies on observation and physics to model the processes in acoustic instruments that shape the instruments' sound. The unfolding in time or morphology of these processes within notes and gestures in physical models are then more closely analogous to the processes in acoustic instruments, and expressive musical use then is based more closely on modeled natural responsiveness. Separate qualities of the sound such as loudness, articulation, etc. are then inter-dependent—a change in one effects change in the others. This inter-dependence contrasts with the earlier synthesis methods, which built up sounds from basic waveshapes and shaped them by envelopes controlling frequency, amplitude, or filtering the spectrum in time, possibly with slight randomization of certain parameters to add liveliness and diminish exact duplication from note to note. Wavetable synthesis (as used in modern samplers) achieved a certain success because the complicated attack portion was satisfying (duplicating exactly as it did the attack of some recorded instrument). While a degree of naturalness and variability was possible, musicians quickly noticed the flat similarity of the wavetable samples and method's reluctance to flex to expressive ends. While the use of high quality samples is probably the most prevalent synthesis method

today, used to simulate even large orchestras in film and television soundtrack production, physical modeling is making important inroads.

One important area of physical modeling research models the plucked string. An overview of this research is provided in the next section. Then, I introduce an elaborate plucked-string instrument implemented in two music software synthesis systems, Max/MSP and Csound. Lastly, I discuss challenges to building-in possibilities of expressive control and how these challenges can be met.

In the instrument model implementation, I retain as much possibility of expressiveness as possible, at the expense of employing a large number of control parameters. I retain the generality of the instrument, which is, thus, potentially many instruments. Several facets are emphasized which are less often investigated in the literature: the use of unusual impulses to excite the string models, the use of dynamic functions to vary parameters, and resonance and feedback effects between the strings.

The Max/MSP implementation favors real-time exploration of settings and sounds. The Csound implementation offers precise specification and repeatability at the expense of lacking the immediate feedback of performance. My interest is in having available both the immediacy and responsiveness of a real-time software system like Max/MSP and the specificity and repeatability of a run-time software system such as Csound.

This instrument model and two implementations provide solutions to compositional problems that I can use in future compositions, when my focus will be on composition rather than problem solving. I hope these solutions can be adapted for

exploring other types of synthesis, and that the implementations described here may be useful to other composers or to teachers of computer music composition.

1.1. PLUCKED STRING SYNTHESIS MODELS

This instrument model implementation I call *Nukulele* (new ukulele) is based on previous plucked string models: the Karplus-Strong algorithm (Karplus and Strong 1983), digital waveguides as formalized by Julius O. Smith (Smith 1987, 1992, 2004), and recent extensions and refinements by others (Jaffe and Smith 1983, Sullivan 1990), especially the Finnish acoustics researchers at the Helsinki University of Technology, Laboratory of Acoustics and Audio Signal Processing (Karjalainen et al. 1993, 1998, Välimäki et al. 1996).

As is well known, the Karplus-Strong algorithm feeds an impulse of random noise into a delay line, the output of which is filtered and fed back into the delay line (Karplus and Strong 1983). The output sound thus has a bright and loud attack that decays and becomes less bright over time, much as does the sound of plucked string. The algorithm is quite computationally efficient. Figure 1 shows a block diagram of the Karplus-Strong algorithm.

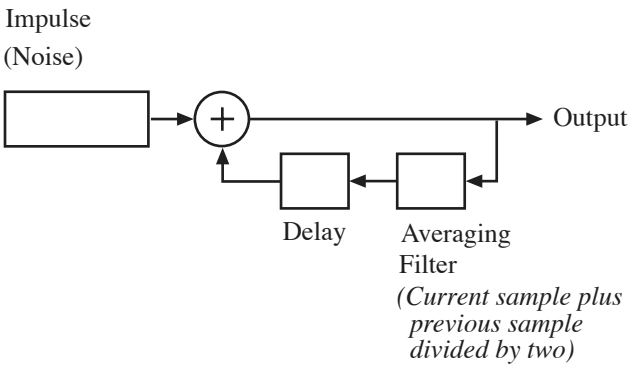

Figure 1. A block diagram of the Karplus-Strong algorithm.

Consider a real plucked string (though much here is applicable to struck and bowed strings too): the pluck initiates a wave that propagates along the string, vibrating at a frequency dependent on the length and tension of the string. The wave carries the shape of the original pluck. The pluck on a real string, in fact, sends a wave towards the bridge and another in the opposite direction—on a guitar, towards the end stopped by a finger at a fret. In software models, the delay line models the circulation of the wave around the path of the closed string. Filters model the physical and mechanical losses at the bridge and fret (as the wave on a real string radiates into the body of the instrument and then into the air). Actually, a real string vibrates in three dimensions: first is that perpendicular to the length of the string and in the plane of the flat top of the body, that is, towards and away from the other strings; second is that perpendicular to the string and perpendicular to the plane of the body, that is, towards and away from the body. And the third is longitudinal, lengthening and shortening of the string, especially on instruments which allow more free motion, for instance, at a tuning pin. (A torsional vibration is

usually ignored in plucked string computer models, but can be important in bowed string models since a bow forces more twisting in the string.)

Digital waveguides, as formalized by Smith, model the behavior of a string by using two delay lines, which model the waves towards the bridge and the fret (Smith 1987, 1992, 1997b, 2004). A complete model would include filters to simulate the filtering due to the body of the instrument, the size and density of the material in the string, as well as the air through which the resultant sound must travel. *Commuted* waveguides subsume many of these additional factors into the sound of the impulse for efficiency (a mathematical convenience moving the bridge/fret/body filter from the end of the signal chain to the beginning with the impulse), rather than supplying each modeling block in the software algorithm (Smith 1987, 1993, Karjalainen et al. 1993). Effects due to the body of the instrument, the plectrum, string constitution, etc., are then automatically included, but are then not variable. (It should be noted that patents exist for both the Karplus-Strong algorithm and digital waveguides.) Example 2 shows a block diagram of a digital waveguide.

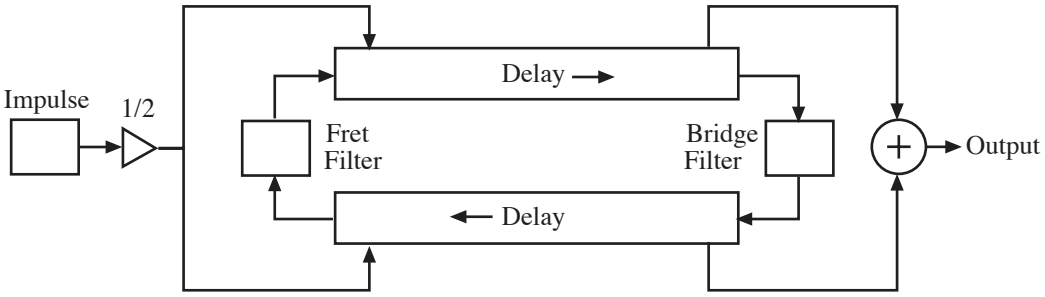

Figure 2. Block diagram of a digital waveguide.

The Helsinki group has investigated using various carefully crafted impulses obtained from recordings of guitars and other instruments, resulting in very realistic emulation by synthesis. Measurement and extraction techniques are described in several papers (Erkut et al. 2000, Karjalainen et al. 1998, Välimäki et al. 1996, Tolonen 1998, Tolonen and Välimäki 1997). Excitation impulses have been extracted from high quality recordings taken in an anechoic chamber. Fairly realistic impulses can be derived by editing a suitable short soundfile from the attack of a plucked string tone, finding the resonances of the instrument common to different pitches and removing them with a notch filter from the impulse (Karjalainen et al. 2000) or by inverse filtering the sample with the impulse response of the computational model (Välimäki et al. 1996). Another technique is by FFT analysis and resynthesis with sinusoids and noise (Serra 1997). The partials above a certain threshold of stability are separated into a harmonic portion as data, and resynthesized. This harmonically steady sound is then subtracted from the

output of the plucked string model (excited by the original soundfile), leaving the impulse.

Much more attention has been paid to the extraction of body resonances (Karjalainen and Smith 1996, Karjalainen et al. 2000, Penttinen et al. 2001, Bank et al. 2002 and others). Outside of a commuted waveguide model, interest is focused on variable body resonance for generality (say, for virtual instruments in a family like the viols) or for novel effects (such as impossibly sized or oddly shaped virtual instruments). Body resonances can be extracted by methods similar to those mentioned above to extract the excitation impulse. Since the most important body resonances are usually fairly low frequencies with long decays, they can easily be separated from the harmonically steady data. Body resonances can also be extracted by the technique of striking an actual instrument with an impulse hammer and recording the impulse response, so they can be later convolved with the final synthesized tones from the plucked string model (or with the excitation impulse in the case of the commuted waveguide model). One technique is to deconvolve (division of spectra) signals recorded from the bridge with that from a pickup in the sound hole (Karjalainen et al. 2000). Body resonances are then modeled computationally in various filtering methods, always with added computational expense.

Without access to very high quality recording situations and the numeric processing involved in these extraction techniques, I have sometimes simply taken the attack portion of recorded guitar tones and extracted the impulses. One must remove at least the strongest partials, since anything pitched will repeat in the delay line, and not adjust to the frequency specified by the length of the string. In any case, the extraction

techniques necessary for acoustics research are not necessary for my more general, not necessarily realistic implementation.

Further extensions in the literature have included the addition of filters to model a variable pluck point along the string, the variation of the impulse to simulate different plectrums or plucking styles, the addition of still more filters and effects to model the workings of an electric guitar (Sullivan 1990), addition of filters to achieve a more acoustic sound from an electric guitar (Karjalainen et al. 2000), and the coupling of strings (Tolonen et al. 1998). Researches have attempted to model various instruments such as the mandolin, the banjo (Välikmäki et al. 1996), the oud, the lute (Erlkut et al. 2001), the Turkish tanbur (Erlkut et al. 2000b), the Finnish kantele (Erkut et al. 2002, Välikmäki et al. 1999b), and the clavichord (Välikmäki et al. 2003, 2000). The use of digital waveguides, as a general technique, is also applicable to struck string instruments such as the piano and the hammered dulcimer, bowed instruments such as the violin family, and all sorts of wind instruments (Smith 2004).

One long standing problem in plucked string modeling was the need for fractional delays, because the nature of a digital delay line limited the available frequencies to integer multiples of the number of samples in the delay lines. An implementation of a simple delay line/filter loop results in pitches that are close to useful, but grow more noticeably out of tune at higher frequencies. The pitches are as accurate as the length of the delay line (in integer samples) will allow, but finer pitch adjustment requires calculating fractional lengths; that is, interpolating between the samples. This was found to be possible and efficiently implemented by an allpass filter given the correct coefficient (Laakso et al. 1996, Välikmäki 1995, Välikmäki et al. 1995, Vlimki and

Laakso 1998). The coefficient is calculated by splitting the nominal frequency into an integer portion (to be sent to the delay line to determine length) and a fractional portion (to be sent to the allpass filter as a coefficient).

The Helsinki research group has made many of these contributions. In recent work, these researchers have modeled a guitar, substituting high quality impulses derived from actual guitars and adding a damping impulse (also derived from actual recordings of guitars) to model the end of a note. They collapsed the digital waveguide model of two delay lines into a loop using a single delay line (SDL) for efficiency (Karjalainen et al. 1998). Their results are very successful and quite convincing: their work was a starting point for my own. Figure 3 displays a block diagram for the single delay line model. Note the inputs and outputs for coupling with other strings.

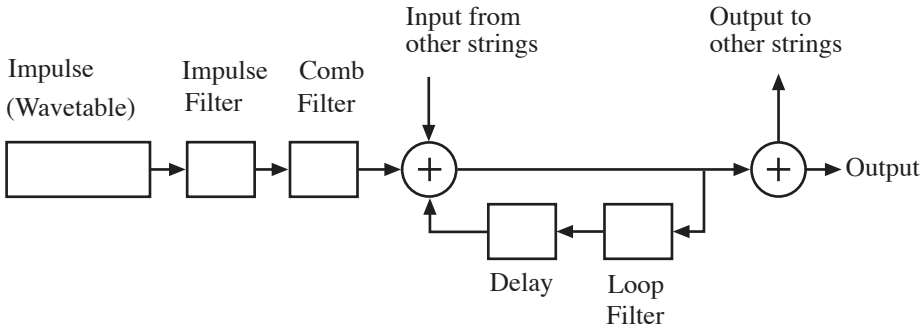

Figure 3. Single delay loop (SDL) block diagram.

Further investigations by the Helsinki group explore the use of two waveguides (or SDL models), slightly mistuned to each other to model the horizontal and vertical vibrations (that is, parallel to the string, and parallel and perpendicular to the plane of the

flat top of the instrument body) that cause the familiar beating effect observable in actual string recordings (Karjalainen et al. 1998).

Their most current researches investigate the slight lengthening and shortening of an actual string (Karjalainen et al. 2001, Tolonen et al 2000, 1999, Välimäki et al. 1999a, 1999b, 1998). This *tension modulation* is a non-linear modulation dependent on the strength of the impulse signal, and accounts for the pitch glide effect, which is more prominent in some instruments such as the kantele than in the guitar (Järveläinen et al. 2001a). They have modeled it effectively with time-varying fractional-delay (TVFD) filters. The slight frequency modulation and amplitude modulation is fed both to the output of the model and recirculated into the delay loop (as it would be in an actual string). Besides a slight pitch and volume modulation in the resulting sound, the tension modulation also affects timbre dynamically, accounting for missing harmonics heard but theoretically canceled when the string is plucked at half the distance between the bridge and fret. (The realism which might be afforded by adding this factor is beyond my current need and focus, but the liveliness which it might add to a plucked string synthesis model offers much potential, and I intend to include it in future implementations. The dynamic functions that I have included in my implementation can be used to simulate at least some of the most obvious effects of tension modulation.)

The literature also describes efforts to build these methods into expressive instruments in software composition environments (Cook 2002, Laurson 2000, Laurson et al. 1999). Plucked string models are included in open source software packages such as Supercollider, Common Music, the Synthesis Tool Kit (also available as Max/MSP or

Pure Data objects in the PerColate package), as objects for Max/MSP in the proprietary Modalys software, and in commercial synthesizers.

CHAPTER 2

NUKULELE

I built *Nukulele* in the Max/MSP programming environment and in Csound. This section describes the user interface, the synthesis engine, and the resonance matrix.

In my work, my goal was not a convincing emulation of a particular instrument, but rather a more general instrument that might be able to produce the sounds of a range of plucked string instruments and especially non-existent, new instruments. *Nukulele's* capability of using various excitation signals, even soundfiles unrelated to plucking, which offers a wealth of possible sounds and textures. *Nukulele* includes a number of dynamic modulations for expressive effects: amplitude, pitch (besides the specification of a given note), pitchbend, and vibrato—all controlled by envelopes. Also, *Nukulele* links four strings together so they can resonate through each other; the resonances from and to each string are also variable. The Helsinki group has published on coupling strings, their coupling strategy avoided any possibility of feedback (Karjalainen et al. 1998, Tolonen et al. 1998), whereas I sought the ability to exploit feedback just as it is often used with an electric guitar.

This larger number of available controls gives the instrument the potential to be quite flexible and expressive, although difficult to control, especially in real-time. I have emphasized the instrument's ability to explore possible sounds rather than try to make it

easily playable, though some interested person could extend the implementation in that way for use with a guitar controller or other input devices. Also, one could extend it to more than four strings, say for a guitar or zither. For my purposes, the Max/MSP environment offers enough control to prototype useful settings. To build up larger blocks of music, I wrote a Max/MSP application that will format and deliver the parameter data in Csound score format. I have used *Nukulele* successfully in two musical works to date, and hope improvements will facilitate its use in more.

2.1. USER INTERFACE

The user interface in the Max/MSP version presents four modules—one for each string—and some controls common to all. Notes are initiated by midi input or by a graphical piano keyboard that is clicked by the mouse. A slider allows pitch bending (also responsive to midi input). Figure 4 shows a screen shot of the Max/MSP application's main screen.

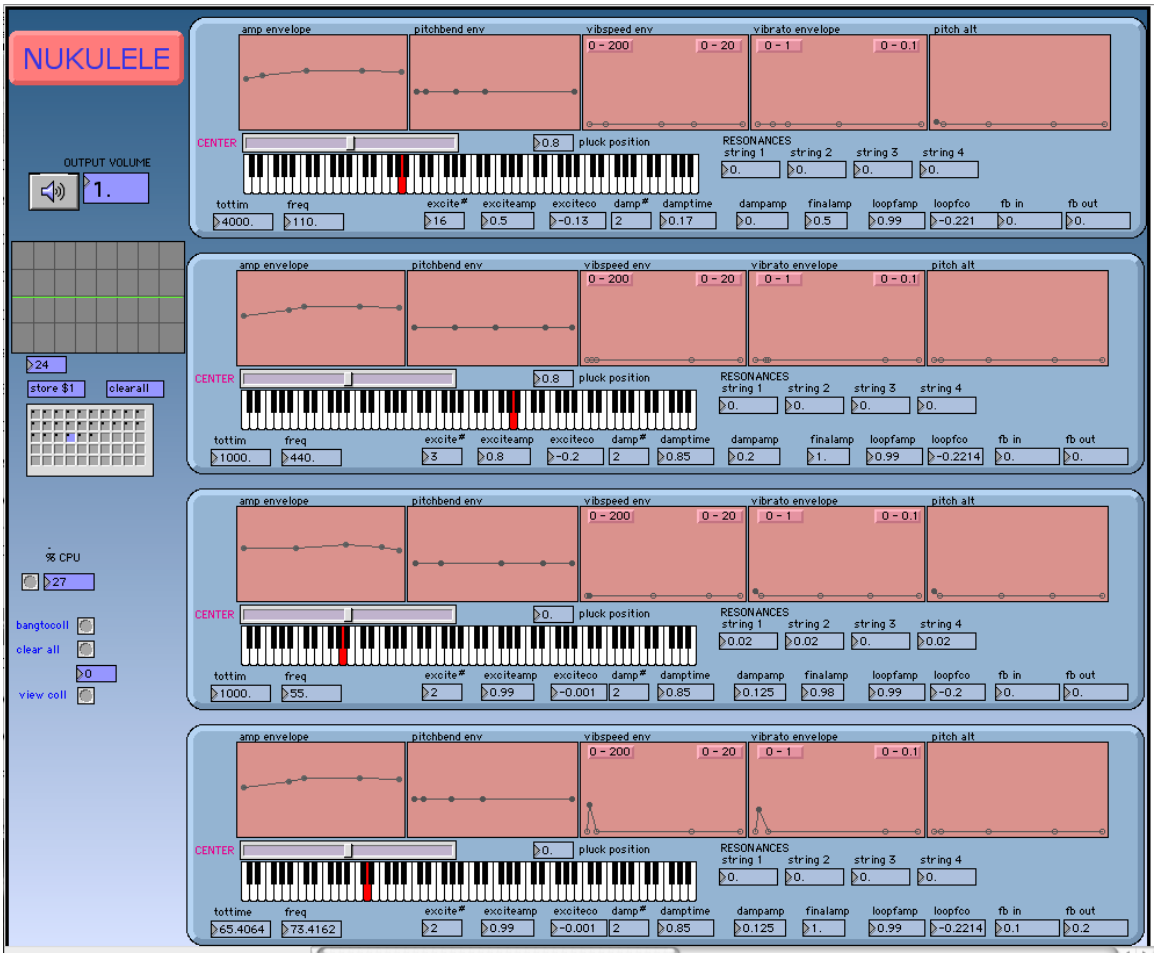

Figure 4. *Nukulele* main screen.

Besides the graphical breakpoint envelope interfaces, entry fields for each string accept numbers for setting various parameters, as shown in Table 1.

(tottime)	total time
(freq)	frequency (also displays what is set by the keyboard, but can adjust the pitch of an open string resonating with the input from other stings)
(ppos)	pluck position (0.0 to 1.0)
(excite#)	number identifier of the exciter soundfile
(exciteamp)	amplitude of the exciter signal
(exciteco)	coefficient of a filter applied to the exciter signal
(damp#)	number identifier of the damping soundfile
(damptime)	percentage of the total time until the start of the damping soundfile
(dampamp)	amplitude of the damping signal
(finalamp)	final amplitude at the end of a note
(loopfamp)	amplitude of the loop filter
(loopfco)	coefficient of the loop filter
(fbin)	amplitude of the signal from all other strings for resonance
(fbout)	amplitude of this module's signal available to other strings as resonance
(string1-4)	individual amplitudes for signals from each string

Table 1. Numeric parameter settings in the Max/MSP *Nukulele* application.

Five graphical function editors allow manipulation of breakpoints of envelopes (time-varying functions which can be applied to various signals) for amplitude, pitchbend, vibrato speed, vibrato amplitude, and pitch alteration. The amplitude envelope allows for sounds that have a slow attack or a swell in the middle, for instance. The pitchbend can alter the pitch by a second above or below, similar to the pitchbend wheels on many synthesizers. The two vibrato envelopes, amplitude and speed, allow for usual vibrato as well as interesting frequency modulation (FM) effects, and include a switch to

change the range of amplitude or speed. The pitch alteration envelope could be used in many ways (or not at all), but the impetus to include it was the familiar slight pitch change in a plucked string often called pitch glide - higher at the beginning when the string is stretched and tense, and lower as the energy dissipates and approaches a state of rest.

The Csound implementation uses the same instrument structure and parameters, but responds to input from a score—a list of notes with parameters—and so events can be scheduled. Note events, with the large number of parameters, can be copied and pasted into a text document and start times for each note typed in. The Max/MSP version can be used to explore sounds and settings, and then export the settings in a format suitable for Csound. One needs then only to specify which string (in Csound, instruments 1 - 4), the start times, the durations for each note (if not appropriate as given from the translation from the Max/MSP *Nukulele* implementation), and the overall amplitude for the note.

2.2. SYNTHESIS ENGINE

At the heart of the sound production part of the instrument is a loop comprising a delay line (modeling the recirculating waves) and loop filter (to shape energy loss over time). As mentioned above, the length of the delay line depends on the integer portion calculated from frequency, and an additional allpass filter handles fine pitch adjustments—the fractional delays. An exciter signal from a soundfile is fed into the delay line/filter loop exciting the circuit that resonates at a frequency dependant on the length of the delay line. The damping signal is also fed in a specified time later. An

amplitude envelope is also in the loop, and can reduce the amplitude at the time of damping, attenuating the signal recirculating through the loop, mimicking a string brought to a state of rest.

The exciter signal is also filtered by a comb filter, allowing control over pluck position to simulate plucking at the bridge or sul tasto effects.

The loop is tapped for output and sent to the computer's audio converter (in Max/MSP) or written to a soundfile (in Csound).

Some difficulties are peculiar to the Max/MSP version. First, Max/MSP specifies a "signal vector"; the number of samples calculated at one time. The smallest available setting is two samples, allowing shorter delay times for higher frequencies, but obviously more computationally intensive. A setting of 128 samples allows a fairly wide pitch range and a low computational overhead. (On my 867 MHz processor, a setting of 2 samples for the signal vector uses up to 70% of the computer's processing, while the setting of 128 uses about 25%). Also, in Max/MSP objects are connected graphically corresponding to logical connections, and yet I found that certain possible layouts would not work. One could not, for instance, embed a process in a subpatcher or send a signal via the send~ and receive~ objects without introducing delays which would alter the timing in the delay loop. These difficulties are not present in the Csound implementation.

2.3. FEEDBACK AND RESONANCE MATRIX

Each string makes available an output signal to all four strings (including itself), and accepts input from all four strings (including itself). Resonances are quite useful, and

add much interest (possibly realism) to sounds. At low resonance gain levels, the results are very subtle. At slightly higher resonance gain levels, one can hear the natural harmonic series emphasized. At high levels, the signal can begin to grow when the signal in each cycle of the loop is louder than in the cycle before. Usual dynamics processing (compression, limiting, or clipping) was not appropriate as I did not want to alter the spectrum of the sound, introduce artifacts, or otherwise alter the sound (especially of the recirculating copies of the exciter signal). I wanted automatic assistance in responding to growing feedback rather than constant compression.

Currently, the feedback control system entails a convention of only using impulses of 80% of the maximum gain, leaving 20% headroom with which to allow for feedback effects. The average output of the loop is tested: if it is over a given threshold, the amplitude of the input from other strings and the gain of the feedback loop are diminished. (From 80% to 90%, the gain is gently diminished. From 90% to 99%, the gain is more drastically diminished, even to zero.) As a safeguard, a clipping filter is inserted after these controls. One trial implementation resulted in the gain slowly oscillating as the signal was pushed below the threshold, the feedback control disengaged, and then feedback rose again. Now, the gain is permanently diminished; that is, it does not return to a higher gain as the signal diminishes below the threshold. To get another event with feedback, one has to restart the loop gain envelope with a new note. A more natural effect can be achieved by filtering rather than overall gain reduction, but I want to keep the software simple and efficient for the time being.

2.4. MORE INSTRUMENT CONTROLS

The user interface in Max/MSP has a few miscellaneous controls. A preset bank allows all parameter settings to be stored and retrieved. Also, included are an adjustable overall amplitude, the usual selector switch to stop and start audio processing and output, and a button to export current settings for later conversion to Csound score format.

CHAPTER 3

MUSICAL USE

In this section, I discuss useful parameter settings and some of the effects on the sound which can be expected, and I offer directions for exploration. I will make it clear whether I am discussing the Max/MSP or Csound version, but the emphasis here is on the instrument in general rather the peculiarities of each software version.

3.1. IMPULSES

The impulse used has a great influence on the sound. The Karplus-Strong algorithm uses simple random noise. Example 1 uses the Csound "pluck" opcode using simple averaging for the loop filter and noise (random values for each sample) as an impulse.

Audio example: click to play.

Example 1. Simple Karplus-Strong synthesis.

Chapter 1 described methods for extraction of realistic excitation impulses. Several examples follow that use such impulses with the *Nukulele* implementation.

Example 2 is one of the soundfiles of an impulse from examples online from a HUT Acoustics Lab paper (Erkut et al. 2000).

Audio example: click to play.

Example 2. An impulse from sound examples from Erkut et al. 2000.

Example 3 is one of the soundfiles of a damping signal from the same set of examples (Erkut et al. 2000).

Audio example: click to play.

Example 3. A damping signal from Erkut et al. 2000.

Example 4 is a musical figure using the impulses and damping signals above in a naturalistic way. One should bear in mind that these are simple, unadulterated monophonic soundfiles, and could be greatly enhanced with various post-processing techniques such as reverberation and placement in a stereo field.

Audio example: click to play.

Example 4. A musical figure using the *Nukulele* implementation with impulses and damping signals from Erkut et al. 2000.

The Helsinki researchers use a database of such impulses and damping signals that are varied according to different playing techniques. A number of excitation impulses and damping impulses can be used to make a particular instrument model which can emulate, for instance, different pluck intensities from soft to hard, the use of the finger or plectrum, etc., in different playing techniques. One could imagine the usefulness of damping signals that capture the unvoiced glissando to the next note or the noise of finger slides between notes (which players usually attempt to suppress). Others impulses or damping signals could be fabricated for related special effects such as scraping a metal string, plucking harmonics, activating the string by sudden and hard placement of the finger of the left hand, and others.

In the Max/MSP implementation, whatever available impulses or damping signals are desired must be edited into a table of impulses selectable by number from the interface. (In the list within the software, the 20 soundfiles are named ex01, ex02, etc. Most convenient is to have a folder of such soundfiles, named accordingly. Then, one can simply substitute folders to change banks of available exciter signals.) In Csound, the soundfiles can be specified in the score by name.

Again, my aim was not realistic emulation like the researches mentioned above, but rather easy exploration and fabrication of useful, interesting, and expressive instruments. Nothing limits one to excitation signals that capture actual plucking or damping. Various percussive sounds work quite well too, such as the metal hit of a triangle, a tomtom, a marimba strike, or even synthesized sounds. Various forms of noise, filtered and/or shaped by an envelope, can be quite useful too. With any of these sources used for excitation, it is possible to make a family of related impulse or damping sources

to emulate varied but related plucking or damping styles within a particular conceptual instrument as would be available to a performer on a real instrument. Example 5 presents four percussive sounds suitable for exciting the string model. Example 6 runs these excitation signals through the *Nukulele* implementation.

Audio example: click to play.

Example 5. Four percussive sounds: a rock click, a hit on an African soprano marimba, a hit on a five-gallon plastic water jug transposed higher, and a hit on a wine glass.

Audio example: click to play.

Example 6. The impulses of Example 5 used in the *Nukulele* implementation with the same parameter settings.

Example 7 contains four bassy drum sounds: a deadened hit on a low drum, a hit on a five-gallon plastic water jug which has a lot of resonance and a long decay, a hit on a five-gallon plastic water jug transposed higher, and a hit on a low drum with a lot of resonance and a long decay. In Example 8, these sounds are used as excitation signals in the *Nukulele* implementation. The resonances within these excitation signals add much that could be heard as the resonance of an instrument body. At the very least, they add much character to the sound.

Audio example: click to play.

Example 7. Four bassy drum sounds.

Audio example: click to play.

Example 8. The impulses of Example 7 used in the *Nukulele* implementation with the same parameter settings.

In Example 9, these impulses are used in a musical context.

Audio example: click to play.

Example 9. A musical figure using the impulses of Example 7, and varying parameters for expressiveness.

The virtual instrument portrayed in Example 9 does not sound like a professional instrument, cobbled together as it is of four widely varying impulses, but it does exhibit some degree of naturalness and expressiveness, such as one could expect from a real, generally available instrument.

Examples 10 and 11 display other percussive sounds used as impulses.

Audio example: click to play.

Example 10. Ten wood clicks for use as impulses.

Audio example: click to play.

Example 11. The wood click impulses of Example 10 used in the *Nukulele* implementation with the same parameter settings, followed by an arpeggio.

Longer, more sustained excitation sources may be used too, and these can be manufactured or altered according to desire. A long sample of a bowed string instrument, for instance, will still play into the harmonic resonances of the open virtual strings of the instrument model, and may be shaped by the envelopes and settings. Extremely long sounds can be used to generate more ethereal textures. These longer soundfiles especially benefit from pitch alteration (outside of the pitch alterations of the virtual strings themselves), which can be simply done by varying the playback speed of the soundfile. This kind of pitch alteration can be fixed overall, or can be varied in time according to an envelope. This capability is not included in either software version, but I have used this to good effect in the Csound version with a few additional lines of code on an ad hoc basis.

Examples 12 and 13 illustrate. Example 12 is a long soundfile of a synthetic bowed string instrument. Example 13 presents the impulse of Example 12 in the *Nukulele* implementation with the added feature that the playback speed of the impulse is slowly increased over its length. One can hear the harmonic resonances of the implementation's virtual string excited by sympathetic frequencies in the impulse as they sweep past in pitch.

Audio example: click to play.

Example 12. A ten-second soundfile of a synthetic bowed string instrument to be used as an impulse.

Audio example: click to play.

Example 13. A musical gesture using the impulse of Example 12, with the speed of playback of the impulse soundfile slowly increasing.

3.2. PARAMETERS

3.2.1. THE ENVELOPE, EXCITATION, AND DAMPING

The total time of the instrument sets the length of the envelope. If the final amplitude is 1.0, the string will continue to sound (in Max/MSP). The excitation signal always begins immediately but the damping signal begins at a set percentage of the total time. If one sets the damping time to an appropriate but rather unusual length, one can have two impulses within a longer, overall note, or even combine the impulse and damping signal at the beginning of the note.

The pluck amplitude can vary the amplitude of the excitation signal, and the damp amplitude varies the amplitude of the damping signal. The pluck filter coefficient can vary the timbre of the excitation signal, but also is applied to a similar filter for the damping signal.

The pluck position ranges from 0.0 to 1.0, and sets the percentage of the length of the string where the pluck occurs. Thus, values near 0.0 correspond to a bright but not-so-loud plucking at the bridge, and values in the middle produce sounds that are more flute-like and have more volume, as in sul tasto playing.

Audio example: click to play.

Example 14. Five notes using a click of two rocks struck together illustrating the pluck position filter: filter settings of 0.1, 0.5, 0.8, 0.99, 0.9999.

3.2.2. THE LOOP FILTER

The loop amplitude sets the gain of the recirculating signal in the string. The loop filter coefficient adjusts the quality of the filter applied to the recirculating signal, and can determine how quickly the sound decays—with no filtering, it would simply continue to recirculate. The use of a lowpass filter causes the sound to begin bright and for the higher frequencies to decay faster.

Audio example: click to play.

Example 15. Five notes using a bassy tomtom as impulse illustrating varying the loop filter amplitude: amplitude settings of 0.9999, 0.990, 0.890, 0.690, and 0.490.

Audio example: click to play.

Example 16. Five notes using a bassy tomtom as impulse illustrating varying the loop filter coefficient: coefficient settings of 0.001, 0.010, 0.221, 0.421, and 0.721.

3.2.3. FREQUENCY

The basic frequency is specified by which note is played on the Max/MSP keyboard or the frequency field in the Csound score. As mentioned, the frequency sets the length of the delay line to integral multiples of the samples, and a fractional part is calculated to send to the allpass filter for adjusting the delay between integral multiples of the samples. In the Max/MSP version, the signal vector size in the DSP window must be set fairly low for high frequencies to be available, but this taxes the processor accordingly. Other pitch alterations are described below under Functions.

Very low frequencies (below 27.5 Hz or A0 on the keyboard in the Max/MSP version) result in echo-like delays. Rhythmic textures can be explored by using different very low frequencies on different strings. Of course, the further frequency modulations described below under "FUNCTIONS" can vary the time of these echo-like delays as well.

Audio example: click to play.

Example 17. Low frequency delay effect, using the bassy tomtom as impulse for three notes: frequencies are 20 Hz., 15 Hz., and 5 Hz.

Audio example: click to play.

Example 18. Modulated low frequency delay effect on three strings with pitch alteration varying frequency, using the rock as impulse: frequencies are 5 Hz., 8 Hz., and 12 Hz.

3.2.4. FUNCTIONS

All of the functions divide the total time into four segments with five breakpoints. The first and last breakpoints are fixed at the beginning and end in time but not in value. The other three can be adjusted in time and value.

The amplitude function makes an envelope which is applied after the envelope handling the excitation and damping. A slower rise in the beginning can take the edge off a sharp attack, or even make a real crescendo with the sustaining part of the looped signal. One can make a swell in the middle or a more severe decrescendo than given by the final amplitude field in the main loop envelope. This would correspond to the use of a volume knob or foot pedal applied to an electric guitar signal, but here is available for each string.

Audio example: click to play.

Example 19. Three notes using a short blast of air with a rapid decay as impulse: amplitude function normal, with a slow attack, and with a sharp attack that diminishes and with a swell.

The pitchbend function is limited to two semitones above or below the basic pitch.

Audio example: click to play.

Example 20. Three notes using a hit on an African soprano marimba illustrating simple pitchbend functions.

The vibrato speed and vibrato amount functions can be adjusted to make time-varying vibrato, but they can also be used for frequency modulation effects. I have found that applying strong and fast frequency modulation to the beginning of a note can open a range of new sounds even with other settings remaining the same. Switches are available to toggle between two ranges for vibrato speed (0 - 20 or 0 - 200) and amplitude (0 - 0.1 and 0 - 1.0). Refer back to Examples 4 and 9 for natural vibrato effects. Example 21 illustrates some FM modulation effects.

Audio example: click to play.

Example 21. Four notes using a hybrid impulse of a hit on a metal wine glass rack and a hit on a plastic jug illustration various vibrato functions as FM modulation: no vibrato/FM, heavy FM modulation applied to the attack, heavy FM modulation applied to the attack with a swell of FM modulation during the sustain portion, and the same but with a multiplier of one tenth.

Finally, the pitch alteration function can be used to simulate pitch glide, raising the pitch slightly at the beginning of a note, corresponding to the increased tension of the string immediately after the pluck. The maximum amount of pitch alteration is limited to two about semitones above the basic pitch. (This is much more than is characteristic for natural sounding pitch glide effects, but is present for exploration of non-natural effects.)

Example 21 presents some pitch alterations. Notice that even subtle changes in the pitch on the onset of a note affects the spectrum which circulates in the delay line.

Audio example: click to play.

Example 22. Four notes using a short blast of air as impulse illustration pitch alteration functions: none, a little alteration at the beginning of the note, too much alteration at the beginning of a note, and the same with a silly rise during the sustain portion to show the range available.

3.2.5. RESONANCE AND FEEDBACK BETWEEN STRINGS

Each string has a gain control for input and output of resonance, as well as separate gain controls for input from each string (including itself). One can explore feedback into the same string that is being played, or any combination of strings. (One can use to simulate a double delay line (as in a standard waveguide), slightly mistuning one from the other to achieve beating due to vibrations in two dimensions.

A guitarist using feedback has the immediate capability of responding to even minor changes in sound, so feedback is easier to control than in software. Standard automatic dynamic processing such as compression or limiting can alter the spectrum of the sound and introduce noticeable artifacts that I wanted to avoid. My current feedback control system tests the output signal (averaged over a short time interval), and, if approaching the maximum amplitude, will diminish the resonance gain as well as the output gain so the feedback will not continue to grow. The feedback gain is diminished when the output signal reaches 80% of the maximum of which the system is capable

(maxamp)—gently between 80% and 90%, more forcefully above 90%, and clipping at just below 100%.

The next seven examples show increasing resonance between three strings, from none to what would be called "unnatural". The examples all hold the settings in the score unchanged between examples, except the resonance parameters which are increased slightly with each succeeding example. The impulse is a blast of air.

Audio example: click to play.

Example 23. No resonance.

Audio example: click to play.

Example 24. More resonance.

Audio example: click to play.

Example 25. More resonance.

Audio example: click to play.

Example 26. More resonance.

Audio example: click to play.

Example 27. More resonance.

Audio example: click to play.

Example 28. More resonance.

Audio example: click to play.

Example 29. Too much resonance.

At low resonance gains, the influence of other strings is quite natural and pleasant, but with more feedback in the system, the sound sometimes seems artificial and shrill. I may need to add some filtering instead of—or in addition to—resonance gain reduction for a more desirable sound. This filtering might model the effects of pickup, electronics, speaker, air, and some delay which are in the signal path of feedback in an acoustic situation.

In the Csound implementation, no automatic feedback control is needed, as one can tell before the sound is played whether it exceeds the available amplitude range. Examples 30 through 32 show various feedback effects in musical figures. Factors used to control feedback are the resonance settings, the length of the notes, and, of course, frequency. Notes in certain simple harmonic ratios tend to reinforce each other and feedback much quicker. For instance, a perfect fourth, unison, octave, perfect fifth, and a major tenth are most susceptible intervals. The spectrum of the impulse (affected by other settings in the implementation) circulating in the string also has modes that can interact with the modes of other strings and feedback. One effective feedback control is slight pitch change, such as pitchbend, pitch alteration, or vibrato, which can quickly defeat the

feedback. Example 30 is simple feedback controlled by length of notes and the resonance settings. Example 31 is a more complicated counterpoint of feedback which employs the length of notes, resonance settings, and pitch alteration to control the feedback. Example 32 is similar, but illustrates some unexpected resonance not at all evident from the simple impulse used.

Audio example: click to play.

Example 30. A simple musical figure using a blast of air as impulse and illustrating feedback between two strings.

Audio example: click to play.

Example 31. Another musical figure using a bassy tomtom as impulse illustrating feedback effects on three strings.

Audio example: click to play.

Example 32. Another musical figure using a hit on a five-gallon plastic water bottle with a lot of resonance as impulse illustrating feedback effects on three strings.

3.3. EXPORT TO CSOUND

In the Max/MSP implementation, when one has adjusted settings to ones satisfaction, one presses the button to save it to a data container (called a "coll"), which is saved to disk. As it stands now, the *NukuleleToCsound* exporter is a separate program. One simply launches it, and then processes the saved coll into another text file, which can be cut and pasted into a Csound score file.

CHAPTER 4

DIFFICULTIES, OPPORTUNITIES, AND CONCLUSIONS

The most painful limitation in the Max/MSP implementation is the mouse input to the graphic piano keyboard. The other settings also must be set individually, which is quite a bit different than the situation with a traditional musical instrument. One could improve the interface to be more playable, but only at the expense of some of the generality of the instrument model as it is. I could imagine altering the program to maximize being playable for one particular preset, having already decided on some settings. Thus, a possible workflow would be to use the *Nukulele* model to zero in on a particular instrument, say, a bass guitar-like sound, and then spin off a copy of the program which could then be edited to make it more playable in ways appropriate to it.

Another limitation is the lack of ability to record gestures of more than one note. One should be able to design something for this with some of the sequencing capabilities of Max, but that was beyond the scope of this paper. While I would not be interested in a fully functional sequencer in my program, it would be nice to be able to record short moves—a musical phrase or so—and adjust settings on successive replays until a gesture was just right, and then export to Csound.

A difficulty is the necessary effort to conceptualize multiple strings. One has to specify which strings are open and sounding at any one time. This is not dissimilar from

honest and conscientious composing for the guitar, but it seems very different than playing one, where a left-hand position and a right-hand strum subsumes many decisions in an instant.

With the promise of faster computers, the model could be altered to include the double delay line for vertical and horizontal vibrations and allow for the addition of the non-linear effect of the subtle lengthening and shortening of the string. One could investigate the modeling of different resonant bodies through which the strings could sound. Electric instruments could be modeled by simulating the use of a pickup, the effect of an amplifier's electronics, and other common effects such as distortion and compression which would enhance the feedback effects. Again, I might not be interested in emulating an actual electric guitar, but it does seem that a whole range of interesting sound possibilities might open up.

A way to increase efficiency would be to write what is know as an "external object", a C program which can then be incorporated into the Max/MSP environment. This likely would give a performance increase, as well as simplifying the interface presented to the user. In Csound, one could write an "opcode" which would have similar possible efficiency and simplification benefits.

One problem with which I struggled through development of the pair of implementations was the conforming of the two to each other. At this point, they operate the very nearly the same; that is, the settings from the Max/MSP implementation, exported to Csound, sound the same when rendered in Csound. If further development is pursued, one might run into situations that cannot be reconciled between the two implementations.

Csound, in fact, can be arranged to run in real-time and respond to MIDI input, and graphical interface capability is available, though users computing on the Windows and Linux platforms currently have an advantage over Macintosh users in this regard. Some interested person could design an interface so that prototyping and scheduling could be done in the same software. Also available is a Csound object for Max/MSP which may offer an easy way to schedule events within the MSP environment.

As an example of the way the basic software implementation presented here can be extended, I can describe recent variations. First, I have an improved method of feedback control that automatically adjusts the cutoff frequency of a lowpass filter, thus regulating the overall volume available to recirculate as feedback. Of course, this alters the spectrum and so I consider it to be an extension of the basic implementation. Second, I have been exploring an inserted module for rough emulation of body resonances. A Max/MSP patcher, called "bodybuilder", uses a reverberation unit and a bank of resonant filters to emphasize ten formants, each adjustable in frequency and gain. It makes a difference where this patcher is inserted. Just before the final output is one possibility. Another is to place it in the path of the resonance signal available to other strings, but this would require four of these computationally expensive units. Better, perhaps, would be to simplify the coupling matrix, send each string's resonance output to one common place, and insert the bodybuilder patcher there. Of course, this would diminish the available control over the individual strings and their resonance input and output. And one has to now consider which excitation signals are appropriate: excitation signals that include body resonances already, such as used in commuted waveguide synthesis, might not be appropriate. Once one decides on a useful design, the variables could be incorporated into

the main interface in Max/MSP. Conforming the Csound implementation to reflect these changes exactly might or might not be possible. Other extension of the basic software might or might not include those described here, and so forms of the software model very likely will diverge from each other.

As it is, *Nukulele* offers a starting point for more exploration through relatively simple alterations to the design, though alterations are much easier to make in the Csound orchestra than in the graphical environment of Max/MSP. More strings could easily be added. Two strings with similar settings could be coupled to simulate one string vibrating in the two dimensions (vertically and horizontally). And studio-processing effects such as compression, equalization, room placement, and reverberation can greatly enhance *Nukulele's* output.

Nukulele allows a user to explore plucked string synthesis models in software implementations that emphasize generality and possibilities of expressiveness. Compositional usage depends on the composer's interests, but the two implementations with the bridging conversion utility offer a method of working that can reward a user with novel and interesting sounds, and most importantly, with opportunities for expressiveness.

APPENDIX A

CSOUND ORCHESTRAS AND SCORES

Text files of the Csound *Nukulele* orchestra file, "nukulele.orc", and the Csound *Nukulele* score file template, "nukulele.sco", can be downloaded from the list of "accompanying files" on the main display for this ETD. The file is "nukulelecsoundorc_sco.zip"—an archive of a folder containing the two text files.

Csound *Nukulele* orchestra file code listing.

```
;*********************************************************************
; nukulele orchestra
;    Csound implementation of plucked-string model:
;    A Csound renderer for Max front end. The Max/MSP front end is only
;    for prototyping; the csound version renders scores built from
;    parameters exported from the Max/MSP version.
; Four stings in four instruments communicating via global a-rate
;    variables let the strings resonate each other via feedback. Each
;    string has a large number of variables/p-fields.
; Based on the paper and online examples and csound instruments
;    accompanying:
;    Erkut, C., V. Välimäki, M. Karjalainen, and M. Laurson. 2000a.
;    "Extraction of Physical and Expressive Parameters for Model-Based
;    Sound Synthesis of the Classical Guitar." Presented at the AES
;    108th International Convention (Paris, France, February 19-22),
;    preprint no. 5114.

sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 10
gastr1 init 0
gastr2 init 0
gastr3 init 0
gastr4 init 0

endin

; generic p-fields:
;   p2 start
;   p3 dur
```

```
;   p4 freq
;   p5 (amp)
;   p6 excite# [file to read and use as exciter]
;   p7 exciteamp [amp of exciter sample]
;   p8 exciteco [coefficient for filter applied to exciter sample]
;   p9 pluckpos [pluck position on string (sul ponticello - sul tasto)]
;   p10 damp# [file to read and use as (sample) on note damping]
;   p11 damptime [percentage of dur to start playing damp sample]
;   p12 dampamp [amp of damp sample]
;   p13 finalamp [amp at end of note]
;   p14 loopfamp [amp for loop filter]
;   p15 loopfco [loop filter coefficient]
;   p16 fbin [amp for feedback from other strings]
;   p17 compamount [amount of compression]
;   p18 fbout [amp for output to other strings]
;   p19 str1 [amp for input of string 1 (feedback)]
;   p20 str2 [amp for input of string 2 (feedback)]
;   p21 str3 [amp for input of string 3 (feedback)]
;   p22 str4 [amp for input of string 4 (feedback)]
;   p23 vibamprange [range for vibamp]
;   p24 vibspeedrange [range for vibspeed]
; each function has nine p-fields:
;  amp (exponential)
;   p25 amp starting value
;   p26 amp value 1;
;   p27 amp time% 1
;   p28 amp value 2
;   p29 amp time% 2
;   p30 amp value 3
;   p31 amp time% 3
;   p32 amp value 4
;   p33 amp time% 4
;  pitchbend
;   p34 pb starting value
;   p35 pb value 1;
;   p36 pb time% 1
;   p37 pb value 2
;   p38 pb time% 2
;   p39 pb value 3
;   p40 pb time% 3
;   p41 pb value 4
;   p42 pb time% 4
;  vibspeed
;   p43 vibspeed starting value
;   p44 vibspeed value 1;
;   p45 vibspeed time% 1
;   p46 vibspeed value 2
;   p47 vibspeed time% 2
;   p48 vibspeed value 3
;   p49 vibspeed time% 3
;   p50 vibspeed value 4
;   p51 vibspeed time% 4
;  vibamp
;   p52 vibamp starting value
;   p53 vibamp value 1;
;   p54 vibamp time% 1
;   p55 vibamp value 2
```

```
;   p56 vibamp time% 2
;   p57 vibamp value 3
;   p58 vibamp time% 3
;   p59 vibamp value 4
;   p60 vibamp time% 4
;  pitchalt
;   p61 pitchalt starting value
;   p62 pitchalt value 1;
;   p63 pitchalt time% 1
;   p64 pitchalt value 2
;   p65 pitchalt time% 2
;   p66 pitchalt value 3
;   p67 pitchalt time% 3
;   p68 pitchalt value 4
;   p69 pitchalt time% 4
; whew! all this for some expressiveness!

;
*********************************************************************
; string 1
instr 1

;;; INIT
  ifreq           =  cpspch(p4)
  iampn1          = p5
  iexcite         = p6
  iexciteamp      = p7
  iexciteco       = p8
  ipluckpos       = p9
  idamp           = p10
  idamptime       = p11
  idampamp        = p12
  ifinalamp       = p13
  iloopfamp       = p14
  iloopfco        = p15
  ifbin           = p16
  icompamount     = p17
  ifbout          = p18
  istr1           = p19
  istr2           = p20
  istr3           = p21
  istr4           = p22
  ivibamprange    = p23
  ivibspeedrange  = p24


;;; FUNCTIONS
;     AMP
;            v0   t1      v1   t2      v2   t3      v3   t4      v4
iampt1 = p27*p3
iampt2 = p29*p3
iampt3 = p31*p3
iampt4 = p3 - iampt1 - iampt2 - iampt3
kamp1 expseg p25, iampt1, p26, iampt2, p28, iampt3, p30, iampt4, p32
;display kamp1, p3
;             declick.......                  declick .....
kclick linseg .0000, .01*p3, 1, p3-.02*p3, 1, p3*.01, .0000
```

```
kamp = kamp1*kclick
;display kamp, p3

;     PITCHBEND
ipbval0 = p34
ipbval1 = p35
ipbval2 = p37
ipbval3 = p39
ipbval4 = p41

ipbval0adj = (ipbval0 <= 64 ? ( 2 ^ ( (ipbval0 - 64) *.002646) )  : //
( 2 ^ ( (ipbval0 - 64) *.002604) ) )
ipbval1adj = (ipbval1 <= 64 ? ( 2 ^ ( (ipbval1 - 64) *.002646) )  : //
( 2 ^ ( (ipbval1 - 64) *.002604) ) )
ipbval2adj = (ipbval2 <= 64 ? ( 2 ^ ( (ipbval2 - 64) *.002646) )  : //
( 2 ^ ( (ipbval2 - 64) *.002604) ) )
ipbval3adj = (ipbval3 <= 64 ? ( 2 ^ ( (ipbval3 - 64) *.002646) )  : //
( 2 ^ ( (ipbval3 - 64) *.002604) ) )
ipbval4adj = (ipbval4 <= 64 ? ( 2 ^ ( (ipbval4 - 64) *.002646) )  : //
( 2 ^ ( (ipbval4 - 64) *.002604) ) )
ipbt1 = p36*p3
ipbt2 = p38*p3
ipbt3 = p40*p3
ipbt4 = p3 - ipbt1 - ipbt2 - ipbt3
kpb linseg ipbval0adj, ipbt1, ipbval1adj, ipbt2, ipbval2adj, ipbt3, //
ipbval3adj, ipbt4, ipbval4adj
kpb = kpb
;display kpb, p3

;     VIBSPEED
ivspt1 = p45
ivspt2 = p47
ivspt3 = p49
ivspt4 = p3 - ivspt1 - ivspt2 - ivspt3
kvsp linseg p43, ivspt1, p44, ivspt2, p46, ivspt3, p48, ivspt4, p50
;display kvsp, p3
;     VIBAMP
ivampt1 = p54*p3
ivampt2 = p56*p3
ivampt3 = p58*p3
ivampt4 = p3 - ivampt1 - ivampt2 - ivampt3
kvamp linseg 52, ivampt1, p53, ivampt2, p55, ivampt3, p57, ivampt4, p59
;display kvamp, p3

;     VIBRATO
kvib oscili kvamp, kvsp, 1
;display kvib, p3

;     PITCHALT
ipaltt1 = p63*p3
ipaltt2 = p65*p3
ipaltt3 = p67*p3
ipaltt4 = p3 - ipaltt1 - ipaltt2 - ipaltt3
ipalttunused = p69
kpitchalt linseg p61, ipaltt1, p62, ipaltt2, p64, ipaltt3, p66, //
ipaltt4, p68
display kpitchalt, p3
```

```
;      PITCH ALTERATION TO DELAY TIME
kpitchmod = 1/(kpitchalt + kvib + kpb)
;display kpitchmod, p3

;;; LOOP COEFFICIENT ENVELOPE
  kloopenv       linseg      1, idamptime*p3, 1, p3-idamptime*p3, //
ifinalamp
;display kloopenv, p3
;calculations for fractional delay interpolation filter
  kfreq       =           ifreq
  ktime       =           1/(kfreq)
  itime       =           1/(ifreq)
;delay time *in samples* (integer)
iint = int(sr/ifreq)
  kint     =           int(sr*ktime)
;remainig fractional delay *in fractions of a sample*
  kfrac    =           frac(sr*ktime)

;;; EXCITATION
indexsamp = nsamp(iexcite)/sr
aindex linseg 0, indexsamp, nsamp(iexcite), p3 - indexsamp, //
nsamp(iexcite)
aexcitation tablei aindex, iexcite
aexcitation    =           iexciteamp*aexcitation

;;; DAMPING
indexdampsamp = nsamp(idamp)/sr
adindex linseg 0, idamptime*p3, 0, indexdampsamp, nsamp(idamp), p3 - //
idamptime*p3 - indexdampsamp, nsamp(idamp)
adamp tablei adindex, idamp
adamp    =           idampamp*adamp

;;; PLUCK SHAPING FILTER E(Z)
  aexcitation    filter2     aexcitation, 1, 1, 1+iexciteco, iexciteco
  aexcitation    =           iexciteamp*aexcitation

  adamp     filter2     adamp, 1, 1, 1+iexciteco, iexciteco
  adamp     =           idampamp*adamp

; Set waveguide
  awgout1   init        0
  afeedback init        0
; Input
  afeedback = gastr1*istr1 + gastr2*istr2 + gastr2*istr3 + gastr4*istr4
  ainput    =           aexcitation + adamp
display ainput, p3
; COMB FILTER FOR PLUCK POSITION
alow delayr 1/20
icombdel = itime * ipluckpos
acomb deltapi icombdel
delayw ainput
acombed = (ainput - acomb)*.5

; COMPRESSOR
; to be added
```

```
;----------------   DELAY LINE ----------------------------------
; SET MIN FREQUENCY
  alowestf    delayr      1/20
; INTEGER DELAY
  awg1      deltapn     kint * kpitchmod

; LOOP FILTER COEFF
            ; B(z)/A(z)
  aloopfilt    filter2     awg1, 1, 1, 1+iloopfco, iloopfco

; FD FILTER (ALLPASS)
  afdf      biquad      aloopfilt, 1-kfrac, kfrac, 0, 1, 0, 0

; LOOP FILTER GAIN
  awgout1   = iloopfamp*afdf*kloopenv

; DELAY WRITE
  delayw      acombed + awgout1 + afeedback*ifbin

;;;  OUTPUT
awgaout1 dcblock awgout1
aout = awgaout1
gastr1 = awgaout1*ifbout*kamp
;display ainput, p3
            out        iampn1*kamp*aout

endin

;
**********************************************************************
; instruments 2 - 4 similar.
;     copy instrument 1 and change variables instr 1 (near the top),
and the gastr1
;     variable (just after ;;; OUTPUT) to reflect the new instrument
number.
```

Csound *Nukulele* score file template code listing.

```
;***** nukulele.sco

; EXCITATION SIGNAL
f1 0 8193 10 1  ; Sine

f2 0 16385 -1 "ex01" 0 0 0
f3 0 16384 -1 "ex02" 0 0 0
f4 0 16384 -1 "ex03" 0 0 0
f5 0 16384 -1 "ex04" 0 0 0
f6 0 16384 -1 "ex05" 0 0 0
f7 0 16384 -1 "ex06" 0 0 0
f8 0 16384 -1 "ex07" 0 0 0
f9 0 16384 -1 "ex08" 0 0 0
f10 0 16384 -1 "ex09" 0 0 0
f11 0 16384 -1 "ex10" 0 0 0
f12 0 16384 -1 "ex11" 0 0 0
f13 0 16384 -1 "ex12" 0 0 0
f14 0 16384 -1 "ex13" 0 0 0
f15 0 16384 -1 "ex14" 0 0 0
f16 0 16384 -1 "ex15" 0 0 0
f17 0 16384 -1 "ex16" 0 0 0
f18 0 16384 -1 "ex17" 0 0 0
f19 0 16384 -1 "ex18" 0 0 0
f20 0 16384 -1 "ex19" 0 0 0
f21 0 16384 -1 "ex20" 0 0 0


t 0 60
; make instrument 10 play so global variables (resonance signals) are
available
i10 0 4

; two musical notes as examples:
;   note the added start times (p2) and duration (p3);
;   note the p4 (pitch) specified in cpspch notation
;      (octave:pitchclass);
;   and the gain setting in p5.
;1  2     3     4   5  6     7     8      9     10    11    12    \\
;13    14    15     16    17    18    19    20     21   22    23    24
i1 0.00 2.00 7.00 1.0  2.000 1.000 -0.200 0.800 7.000 0.850 0.185 \\
1.000 0.990 -0.221 0.100 0.000 0.200 0.000 0.000 0.000 0.000 0.000 \\
0.000 \\
      0.800 0.800 0.299 0.800 0.095 0.800 0.360 0.776 0.247 \\
      63.500 63.500 0.059 63.500 0.176 63.500 0.199 63.500 0.566 \\
      0.000 0.000 0.030 2.000 0.030 5.000 0.625 1.000 0.315 \\
      0.000 0.000 0.021 0.005 0.064 0.010 0.713 0.005 0.202 \\
      0.020 0.000 0.145 0.000 0.179 0.000 0.395 0.000 0.281

i2 1.00 1.00 6.05 0.5  3.000 1.000 -0.200 0.800 8.000 0.850 0.185 \\
1.000 0.990 -0.221 0.100 0.000 0.200 0.000 0.000 0.000 0.000 0.000 \\
0.000 \\
      0.800 0.800 0.299 0.800 0.095 0.800 0.360 0.776 0.247 \\
```

```
        63.500 63.500 0.059 63.500 0.176 63.500 0.199 63.500 0.566 \\
        0.000 0.000 0.030 2.000 0.030 5.000 0.625 1.000 0.315 \\
        0.000 0.000 0.021 0.005 0.064 0.010 0.713 0.005 0.202 \\
        0.000 0.000 0.045 0.000 0.279 0.000 0.395 0.000 0.281
e
```

APPENDIX B

MAX/MSP APPLICATIONS

The Max/MSP applications can be downloaded from the list of "accompanying files" on the main display for this ETD. Applications have been built for Mac OS X for users who own Max/MSP. The file is "nukulele.zip"—an archive of a folder containing the *Nukulele* and *NukuleleToCsound* Max/MSP applications, sub-patches, and excitation soundfiles.

:.

BIBLIOGRAPHY

Bank, Balázs, Giovanni De Poli, and László Sujbert. 2002. "A Multi-Rate Approach to Instrument Body Modeling for Real-Time Sound Synthesis Applications." 112th AES Convention (Munich, Germany, May 10-13), Preprint No. 5526.

Beck, Stephen David. 2000. "Designing Acoustically Viable Instruments in Csound." In *The Csound Book,* ed. Richard Boulanger. Cambridge, MA: MIT Press: 155-170.

Cook, Perry R. 2002. *Real Sound Synthesis for Interactive Applications*. Natick, Massachusetts: A. K. Peters.

Cuzzucoli, Giuseppe, and Vicenzo Lombardo. 1999. "A Physical Model of the Classical Guitar, Including the Player's Touch." *Computer Music Journal* 23(2): 52-69.

Erkut, C., M. Karjalainen, P. Huang, and V. Välimäki. 2002. "Acoustical analysis and model-based sound synthesis of the kantele." *Journal of the Acoustical Society of America* 112(4): 1681-1691.

Erkut, C., M. Laurson, M. Kuuskankare, and V. Välimäki. 2001. "Model-based Synthesis of the Ud and the Renaissance Lute." *Proceedings of the International Computer Music Conference* (Havana, Cuba, September 17-23): 119-122.

Erkut, C., V. Välimäki, M. Karjalainen, and M. Laurson. 2000a. "Extraction of Physical and Expressive Parameters for Model-Based Sound Synthesis of the Classical Guitar." Presented at the AES 108th International Convention (Paris, France, February 19-22), preprint no. 5114.

Erkut, Cumhur, Tero Tolonen, Matti Karjalainen, and Vesa Välimäki. 2000b. "Acoustical Analysis of Tanbur, a Turkish Long-necked Lute." *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* (ICASSP'00) vol. 2 (Istanbul, Turkey, June 5-9): 769-772.

Jaffe, David A., and J. O. Smith. 1983. "Extensions of the Karplus-Strong plucked string algorithm." *Computer Music Journal* 7(2): 56-69.

Järveläinen, H., and V. Välimäki. 2001a. "Audibility of initial pitch glides in string instrument sounds." *Proceedings of the International Computer Music Conference* (Havana, Cuba, September 17-23): 282-285.

Järveläinen, H. 2001b. "Applying perceptual knowledge in string instrument synthesis." *Proceedings of the MOSART Workshop on current research directions in computer music* (November 15-17, Barcelona, Spain): 187-195.

Karjalainen, M., T. Tolonen, V. Välimäki, C. Erkut, M. Laurson, and J. Hiipakka. 2001. "An Overview of New Techniques and Effects in Model-based Sound Synthesis." *Journal of New Music Research* 30(5): 203-212.

Karjalainen, M., V. Välimäki, H. Penttinen, and H. Saastamoinen. 2000. "DSP equalization of electret film pickup for the acoustic guitar," *Journal of the Audio Engineering Society* 48(12): 1183-1193.

Karjalainen, M., H. Penttinen, and V. Välimäki. 2000. "Acoustic Sound from the Electric Guitar Using DSP Techniques." *Proceedings of the International Conference on Acoustics, Speech and Signal Processing* (ICASSP'00 Istanbul, Turkey, June 5-9) vol. 2: 773-776.

Karjalainen, M., H. Penttinen, and V. Välimäki. 1999. "More acoustic sounding timbre from guitar pickups," in Proceedings of the 2nd G-6 Workshop on Digital Audio Effects (DAFX99, Trondheim, Norway, Dec. 9-11): 41-44.

Karjalainen, M., V. Välimäki, and T. Tolonen. 1998. "Plucked-string models: from the Karplus-Strong algorithm to digital waveguides and beyond." *Computer Music Journal* 22(3): 17-32.

Karjalainen, M., and Julius Smith. 1996. "Body Modeling Techniques for String Instrument Synthesis." *Proceedings of the International Computer Music Conference* (Hong Kong, August): 232-239.

Karjalainen, M., V. Välimäki, and Z. Jánosy. 1993. "Towards high-quality sound synthesis of the guitar and string instruments." *Proceedings of the International Computer Music Conference* (Tokyo, Sept. 10-15): pp. 56-63.

Karplus, K. and A. Strong. 1983. "Digital synthesis of plucked string and drum timbres." *Computer Music Journal* 7(2): 43-55.

Laakso, T. I., V. Välimäki, M. Karjalainen, and U. K. Laine. 1996. "Splitting the Unit Delay--Tools for Fractional Delay Filter Design." *IEEE Signal Processing Magazine* 13(1) (January): 30-60.

Laurson, M., C. Erkut, V. Välimäki, and M. Kuuskankare. 2001. M., "Methods for Modeling Realistic Playing in Acoustic Guitar Synthesis." *Computer Music Journal* 25(3): 38-49.

Laurson, M., C. Erkut, and V. Välimäki. 2000. "Methods for Modeling Realistic Playing in Plucked-String Synthesis: Analysis, Control and Synthesis." *Proceedings of the COSTG6 Conference on Digital Audio Effects (DAFx'00)*, (Verona, Italy, Dec. 7-9): 183-188.

Laurson. M. 2000. "Real-Time Implementation and Control of a Classical Guitar Synthesizer in SuperCollider." *Proceedings of the International Computer Music Conference* (Berlin): 74-77.

Laurson, M., J. Hiipakka, C. Erkut, M. Karjalainen, V. Välimäki, and M. Kuuskankare. 1999. "From Expressive Notation to Model-Based Sound Synthesis: A Case Study of the Acoustic Guitar." *Proceedings of the International Computer Music Conference* (Beijing, China, October 22-28): 1-4.

Levitin, Daniel J., Stephen McAdams, and Robert L. Adams. 2002. "Control parameters for Musical Instruments: a Foundation for New Mappings of Gesture to Sound." *Organized Sound* 7(2): 171-189.

Penttinen, H., M. Karjalainen, and A. Härmä. 2001. "Morphing Instrument Body Models." Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx01, Limerick, Ireland. Dec. 6-9): 50-54.

Rank, Erhard, and Gernot Kubin. 1997. "A Waveguide Model for Slapbass Synthesis." *Proceedings of the International Conference on Acoustics and Speech Signal Processing,* vol. 1: 443-446.

Roads, Curtis, ed. 1989. *The Music Machine*. Cambridge, MA: MIT Press.

Roads, Curtis. 1996. *The Computer Music Tutorial*. Cambridge, MA: MIT Press.

Schroeder, M. R. 1965. "New Method of Measuring Reverberation Time." *Journal of the Acoustic Society of America* 37: 409-412.

Serra, X. 1997. "Musical sound modeling with sinusoids plus noise." In *Musical Signal Processing* C. Roads, S. T. Pope, A. Piccialli, and G. DePoli, eds. Lisse: Swets & Zeitlinger : 91-122.

Smith, Julius O. 2004. *Digital Waveguide Modeling of Musical Instruments*. Center for Computer Research in Music and Acoustics (CCRMA), Stanford University. Web published at http://www.ccrma.stanford.edu/~jos/waveguide/.

Smith, J. O. 1997a. "Acoustic modeling using digital waveguides." In *Musical Signal Processing,* C. Roads, S. T. Pope, A. Piccialli, and G. DePoli, eds. Lisse: Swets & Zeitlinger: 221-263

Smith, J. O. 1997b. "Principles of digital waveguide models of musical instruments." In *Applications of Digital Signal Processing to Audio and Acoustics*, M. Kahrs and K. Brandenburg, eds. Boston, MA: Kluwer: 417-466.

Smith, Julius O. 1996 "Physical Modeling Synthesis Update." *Computer Music Journal* 20(2): 44-56.

Smith, J. O. 1993. "Efficient synthesis of stringed musical instruments." *Proceedings of the 1993 International Computer Music Conference* (Tokyo): 64-71.

Smith, Julius O. 1992. "Physical modeling using digital waveguides." *Computer Music Journal* 16(4): 74-91.

Sullivan, C. R. 1990. "Extending the Karplus-Strong algorithm to synthesize electric guitar timbres with distortion and feedback." *Computer Music Journal* 14(3): 26-37.

Smith, Julius O. 1987. "Music applications of digital waveguides." Tech. Rep. STAN-M-39, CCRMA, Music Department, Stanford University, 1987, a compendium containing four related papers and presentation overheads on digital waveguide reverberation, synthesis, and filtering.

Tolonen, Tero, Vesa Välimäki, and Matti Karjalainen. 2000. "Modeling of Tension Modulation Nonlinearity in Plucked Strings." *IEEE Transactions on Speech and Audio Processing* 8(3) (May): 300-310.

Tolonen, Tero. 1999. "Methods for Separation of Harmonic Sound Sources using Sinusoidal Modeling." AES 106th Convention, Munich, Germany, May 8-11, 1999.

Tolonen T., C. Erkut, V. Välimäki, and M. Karjalainen. 1999. "Simulation of plucked strings exhibiting tension modulation driving force." *Proceedings of the International Computer Music Conference* (Beijing, China, October): 5-9.

Tolonen, Tero. 1998. *Model-Based Analysis and Resynthesis of Acoustic Guitar Tones.* Master's thesis. Report no. 46, Helsinki University of Technology, Department of Electrical and Communications Engineering, Laboratory of Acoustics and Audio Signal Processing, Espoo, Finland.

Tolonen, T., V. Välimäki, and M. Karjalainen. 1998. "A new sound synthesis structure for modeling the coupling of guitar strings." *Proceedings of the IEEE Nordic Signal Processing Symposium (NORSIG'98),* (Vigsș, Denmark, June 8-11): 205-208.

Tolonen, Tero, and Vesa Välimäki. 1997. "Automated Parameter Extraction for Plucked String Synthesis." *Proceedings of the International Symposium on Musical Acoustics* (ISMA97, Edinburgh, Scotland, August 19-22), vol.1: 245-250.

Välimäki Vesa, Mikael Laurson, and Cumhur Erkut. 2003. "Commuted Waveguide Synthesis of the Clavichord." *Computer Music Journal* 27(1): 71-82.

Välimäki, Vesa, Mikael Laurson, Cumher Erkut, and Tero Tolonen. 2000. "Model-Based Synthesis of the Clavichord." *Proceedings of the 2000 International Computer Music Conference*, (Berlin, Germany, August 27 -September 1): 50-53.

Välimäki, V., T. Tolonen, and M. Karjalainen. 1999a. "Plucked-String Synthesis Algorithms with Tension Modulation Nonlinearity." *Proceedings of the International Conference on Acoustics and Speech Signal* (ICASSP'99) vol. 2 (Phoenix, Arizona, March 15-19): 977-980.

Välimäki, Vesa, M. Karjalainen, T. Tolonen, and C. Erkut. 1999b. "Nonlinear Modeling and Synthesis of the Kantele - a Traditional Finnish String Instrument." *Proceedings of the 1999 International Computer Music Conference*, (Beijing, China, October 22-27): 220-223.

Välimäki, V., T. Tolonen, and M. Karjalainen. 1998. "Signal-Dependent Nonlinearities for Physical Models Using Time-Varying Fractional Delay Filters." *Proceedings of the International Computer Music Conference* (Ann Harbor, Michigan, October 1-6): 264-267.

Välimäki, Vesa, and Timo I. Laakso. 1998. "Suppression of Transients in Time-varying Recursive Filters for Audio Signals." Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP98, Seattle Washington, May 12-15) vol. 6: 3569-3572.

Välimäki, V., J. Huopaniemi, M. Karjalainen, and Z. Jánosy. 1996. "Physical modeling of plucked string instruments with application to real-time sound synthesis." *Journal of the Audio Engineering Society* 44(5) (May): 331-353.

Välimäki, V. 1995. *Discrete-Time Modeling of Acoustic Tubes Using Fractional Delay Filters*. PhD thesis, Report no. 37, Helsinki University of Technology, Faculty of Electrical Engineering, Laboratory of Acoustic and Audio Signal Processing, Espoo, Finland.

Välimäki, V., T. I. Laakso, and J. Mackenzie. 1995. "Elimination of transients in time-varying allpass fractional delay filters with application to digital waveguide modeling." *Proceedings of the 1995 International Computer Music Conference* (Banff): 327-334.

Vlimki, V. and T. I. Laakso. 1998. " Suppression of Transients in Variable Recursive Digital Filters with Novel and Efficient Cancellation Method." *IEEE Transactions on Signal Processing* 46(12): 3408-3414.