An Investigation Into The Design Of Musical Performance Instruments

Dylan Menzies-Gow
Music Technology MSc
University of York
August 1995

Abstract

The Yamaha VL1 has attracted much interest as the first generally available synthesiser to emulate the subtle dynamic response of acoustic instruments, and yet not be constrained to copy these instruments wholesale. While the VL1 is a powerful, state of the art machine, the possibility is explored here of enriching the control dynamics side of existing MIDI equipment by the computer processing of MIDI control signals with an Atari ST. The WX7 windcontroller and the polyphonic aftertouch keyboard are considered as controlling devices. This leads onto more general considerations of musical performance instruments. Csound running in real time on an SGI Indy equipped with a MIDI interface is used to explore techniques not accessible on MIDI synthesisers. Several useful examples are presented, and some ideas for future work which the author feels encouraged to undertake.

Acknowledgements

I would like to thank Ross Kirk and Andrew Hunt for their enthusiasm and help with this project. Valuable conversations with Music Technology students past and present have left their mark, I'm sure. Thankyou.

Contents

Chapter 1

Introduction

This chapter presents some motivations and first ideas for the project in an informal language. More detailed discussion of the concepts will appear later.

The role of performance

The project grows from a desire to create musical performance instruments with modern digital technology that might attain the same credibility as a classical performance instrument. Why focus on performance? There are a host of good reasons. Here are a few are listed:

1. A stage performer can add to the listening experience, and possibly respond or interact with the audience. Even on recordings, if the listener thinks the music was performed 'live' in some way it can affect the perception.

2. The performer can be an effective way of adding life and original interpretation to a written score. Even in a large section of an ensemble, the result would not be the same without the 'Life' of each performer.

3. Conversely composers are often inspired by the qualities of a particular instrument or performer. They mentally improvise.

4. The performer can improvise aloud, and generate new techniques and perhaps musical ideas dependent on these.

5. Finally there is the pleasure of playing an instrument itself: an interactive musical experience, possibly with other players.

These reasons alone account for the huge and ongoing interest all around the world, at all levels of technology in inventing new musical instruments. Of course this doesn't invalidate the use of non-performance sound. And the past has shown there is plenty of room for both, and combinations.

The current state of electronic performance keyboards and their limitations

Many current electronic synthesisers being marketed as performance keyboards lack the control possibilities seen in acoustic instruments, and are often very similar to one another. The key to there usefulness is often just the variety and novelty of the samples they contain. They are all used in a similar way: a keyboard note press triggers a sound: Aftertouch is used to filter or modulate the sound while it plays. The pitch wheel alters the pitch in a clichéd fashion.

The use of effects processors to augment instruments

Effects processors serve to enrich the response of the instrument as well as changing its sound. A good example is delay: A complex, interesting and slightly unpredictable sound can be generated with a few notes. The control of the 'delay-instrument' is more complex, and interesting than without delay: The output depends significantly on the player's input sometime before. It is natural therefore to consider the general class of instruments in which the sound output at a given time depends on the history of input by the player. This shall be the main consideration in the designs described later. In retrospect, acoustic instruments exhibit 'temporal complexity' in the control of their sound, which certainly contributes to their musical value.

Unprocessed sounds from sample-playback keyboards have a very static quality: On repetition, exactly the same sound is output. Apply an effects processor and this is not true as many effects algorithms are time dependent and/or highly sensitive to initial conditions. The control may be uninteresting note on/off but the sound in itself is interesting. This changing quality is very apparent in real instruments like the piano, and is an important design consideration later. The question arises 'how far can temporal complexity alone be musically useful without using 'changing sounds?'

The Yamaha VL1

The Yamaha VL1 is the first generally available synthesiser to emulate the rich response of acoustic instruments, and the main inspiration for this project. It is one of the few today to take an integrated approach to being a musical performance instrument rather than a synthesiser with a keyboard attached. A synthesiser may be capable of producing sounds similar to the VL1 with much effort, but a performer can only become involved and produce good music if the whole instrument is good: the physical side and the response as well as the synthesiser. The WX7 windcontroller is relatively simple and physically unappealing by comparison with a saxophone, yet it can be used to stunning effect with the VL1. This demonstrates the importance of the 'response feel' or temporal complexity of the instrument over the 'physical feel', and hence provides some validation for the use of the WX7 in the following designs. In the VL1 the synthesis is tightly bound to the control response, because it is based on a waveguide model of real instruments: While the VL1 is admired for the 'new' instruments which can be constructed, its response characteristics are inevitably constrained to the waveguide model.

The original instrument designs of the project

In this project the emphasis is on looking at generalised abstract notions of response, whilst giving considerations towards the response of acoustic instruments. The design philosophy of 'something old , something new' applies.

MIDI control processing

The original idea for implementing complex response was to process 'raw' MIDI control signals from a MIDI controller, and produce MIDI output for driving a synthesiser. MIDI processing is not new: The MAX program from IRCAM is widely used. However, no examples of processing on the short time scales associated with playing acoustic instruments, could be found in The Computer Music Journal.

MIDI control with Csound

Running Csound in real time on an SGI Indy with a MIDI interface offers further possibilities for instrument design. The Indy becomes very flexible MIDI tone module. The control processing and synthesis are more closely bound.

Sound processing

The Indy has 4 audio outputs which has possibilities for real time spatialisation. This falls outside the main body of the project, although still relevant in the wider context of performance instruments.

The next chapter looks at instruments past and present, before gathering together some of the general properties of successful instruments and focusing on formal descriptions of them.

Chapter 2

Background

2.1 Review of instrument designs

The reader is likely to be familiar with the properties of the classical acoustic instruments, possibly to a high degree for some instruments. A great deal of time could be spent looking at these in detail, but since the thrust of this project is practical rather than theoretical I will look here at just two; the tenor saxophone and the pianoforte. The two controllers I shall be using are based on the physical operation of these instruments.

The final section looks at the wealth of instruments that have been built using 'modern' technology over the last century. Although not discussed here, there exists a vast array of non-western, and experimental-acoustic instruments. The reader is referred to Sawyer(1977), Jenkins(1983) and Krishnaswamy (1971).

2.1.1 The tenor saxophone

Initially developed as an orchestral instrument by Adolf Sax in the mid 19th century, the wide and particularly 'human' expressive qualities of the saxophone have made it popular as a solo or small group instrument within many styles of music. The instrument is notoriously variable in character, depending on the manufacturer and its condition. The mouthpiece is particularly critical in this sense. Broadly speaking however, styles of playing range from a softer, quieter sound using a thinner reed, to a more powerful, resonant sound using a thicker one. A player should be able to find a reed which gives a good compromise between the two, and a wider expressive, tonal and dynamic range.

Playing the saxophone

First consider the saxophone's behaviour in detail from the moment a musician touches it. Some examples are provided on the tape recording. The body is very solid, physically and visually appealing. It is not too great a stretch of the imagination to see it as a work of modern sculpture.

Starting and ending a note

The moment a key is pressed the body vibrates with satisfying thunk (with a microphone this can be a useful percussive effect). If the mouthpiece is blown with a loose (lower) lip a noisy hiss is heard, amplified by the body. As the lip closes the hiss changes quality - becomes less uniform and predictable. This will vary according to how much condensation has accumulated: It may or may not be desirable. It is possible to gently merge the hiss into a quiet tone for all but the lowest notes. To start a note sharply the tongue is used to block the mouthpiece, pressure is built up in the mouth and then release by quickly removing the tongue. This produces a fairly chaotic attack to the note, but if pressure is sustained the note will smooth out. An important point here is that the attack depends on the shape of the input pressure pulse and not just on its onset time character. This is what I mean by dynamic response. Not only does the player hear the sonic result, but vibrations of the reed are felt directly on the lip. This provides an important additional cue to the state of the instrument.

While a note sounds

While the note is playing its tone can be varied by changing breath pressure, lip pressure, position of lip and the shape of the mouth. The volume and pitch are also affected by these parameters but in different ways: large slides in pitch downwards can be achieved by loosening the lip (smaller upward slides by tightening). There is a subtle interplay between the breath pressure and the lip pressure (which determines how open or closed the lip is). Loosening the lip means the breath flow must be increased to maintain the same pressure. But the pitch is affected by both lip and breath pressure indecently! The higher notes are more sensitive to pitch variation, and even on good Saxophones a fair amount of player listening is required to centre the pitches. This has the advantage that temperament can be adjusted on the fly as for strings and choirs and solo singers. Saxophone quartets sometimes use this technique explicitly in written scores. Vibrato is easily executed by slight variation of lip pressure. At low volume this has the dual effect of modulating the volume of the breath noise, creating a characteristic mixed effect much used by jazz musicians.

Changing notes

Now consider what happens as the player switches notes while maintaining breath pressure. For simple key changes between adjacent notes the sound changes quite smoothly, with a slight attack sound to the second note. For more bigger jumps the joining sound becomes more complex and even unpredictable, but still relatively subtle compared to the attack of a new note after silence. A subtle and interesting technique, widely used, is to change the fingering, possibly by opening upper holes, without changing the pitch. The tone can be rapidly adjusted in small steps this way. A lot of the expressiveness of the saxophone comes from the contrast between musical phrases, maybe complex, that can be achieved effortlessly and phrases, maybe simple, that give the impression of great effort.

Variation across the scale

The sustained tone of the saxophone varies from pure to exceptionally rich at higher breath pressure. Likewise there is a great variation in tone across the note range, providing for additional contrasts within a melody. The upper notes become increasingly difficult to play as the lip pressure needs to be raised without closing the reed. Similarly the lowest notes require slightly less lip and a carefully controlled amount of breath- not too much or too little. These notes are often deliberately split in contrast to playing the same notes cleanly.

Special blowing

Blowing the lower octave upwards an octave rather than using the octave key, produces an even more powerful, overdriven sound beloved of the hip. This sound is on the brink between resonance and chaos. Careful playing can give the impression of two notes separated by an octave. A less well known blowing technique originated by John Coltrane can create other intervals called 'multiphonics'.

2.1.2 The Pianoforte

In a sense the piano has fewer clearly identifiable properties than the saxophone. Its most obvious characteristic is polyphony thus enabling harmony. Over the last two centuries it has evolved into a much larger, heavier instrument. This is because more dynamic range was desired. The harder you hit a string, the thicker it must be to withstand the blow, the higher the string tension must be for same length. Hence a heavy metal frame is required to hold the strings. The penalty of a huge dynamic range is that the tone, especially in the bass has become quite muddy, owing to the thicker strings. When Beethoven wrote for the 'forte' he often used much closer chords in the bass than you would see composers write today.

The keys

The dynamic range is coupled to the subtle physical feel of the keys which help guide the performer as they attempt to deliver the precise energy at the precise time to each key. The key dynamics are physically appealing in their own right, and this can have psychological connections with the music itself. The result is that a very high resolution of dynamics and timing is possible and provides scope for much musical interest.

The sustain pedal

The only other parameters directly affecting the strings come from the pedals. Of these the sustain pedal is very important in allowing the piano to 'breathe': The dampers are released from all the strings so that any one may resonate sympathetically to a degree with any other. When the sustain pedal is not used the keys currently pressed may still resonate sympathetically enriching the mixed sound in a subtle, unpredictable way. Thus the harmonic possibilities of polyphony are augmented.

Digital pianos

Sympathetic resonance is the most difficult aspect to emulate in a digital piano: the pianist can play something twice the same and hear slight differences that he is barely conscious of. It is almost as if this changeability could be called a timbre. The digital pianist may become bored without really being aware that the digital piano is not successfully emulating a real piano.

2.1.3 Modern instruments

The electric guitar

The electric guitar was invented early in the century. There have been many variations, but basically it consists of a guitar with steel strings and electromagnetic pickups, possibly without a resonating cavity. An integral part of the instrument is the processing, amplification and output on speakers of the pickup signal. The success of the instrument comes from the proven physical control interface, the 'natural' root of the final sound combined with the great variety of processing and amplification that has been found musically useful. In particular the valve distortion effect, originally part of the amplification process, is very well suited to the signal. The reason for this seems to be that the signal is harmonically quite pure, but has just enough 'quirks' to become very interesting when distortion is applied: Many instrument sounds become 'messy' when distorted with valves.

The culture of the electric guitar compared to classical instruments

The great variety in electric guitars and their sounds contrasts with the degree of uniformity to which classical instruments have developed. This is because bands seek to play their own music with their own sounds using very few instruments, whereas classical instruments belong to an academic tradition in which standardisation is an important part of creating order. Even so, within the modern music culture there is a balance between respect for 'classic' electric guitar sounds and the new sounds. Just as important are the various styles of playing which are unique to the electric guitar. The same considerations apply to the bass electric guitar, but to a lesser degree.

Early analog instruments

Various early electric analogue instruments such as the Theremin, controlled by hand movement, and the Ondes Martenot, by keyboard were used by composers for their unusual, simplified tones; for instance in Messiaen's Turangalila Symphony. A spate of other electric keyboards were marketed from the early parts of the century. The Hammond Organ in its original form is an ingenious device. Each key has an electromagnetic-mechanical oscillator. Additionally draw-bars can be used to control the presence of harmonics in a note. However the harmonics are obtained by using oscillators of other keys and since the keyboard was even-tempered the harmonics are not exact. This lends the Hammond its character. Its modern form is still very popular. Engineering compromise, due to the cost of oscillators, has been turned to advantage.

Analog instruments in the 70's

The explosion in analogue keyboards used by the pop industry since the 70's has had a continuing, if sporadic, influence. Electronic integration had allowed more complex functions than were possible in the days of the Theremin: The keyboard controllers were augmented with panels of dials and switches linked to filters, low frequency oscillators, pulse width modulators etc. Maybe it wasn't the original intention, but the dials created many performance possibilities in the hands of people such as Brian Eno. Effects such as portamento transform the feel of a traditional keyboard. There was interest in windcontroller-synthesisers: An instrument called 'The Lyricon' was praised for its expressiveness. The use of processing such as plate reverberation, flanging and echo were used as part of a performance keyboard just as for an electric guitar. Sometimes in recordings it is difficult to distinguish the use of processing as part of a performance from something which is applied afterwards, and therefore miss the value of processing in performance.

The MIDI era

The keyboard has continued to be the universal control device for electronic instruments. MIDI is really based around a keyboard architecture. There is some processing of MIDI data on some keyboards. For example arpeggiation, one finger chording. These effects can be useful but also easily become clichéd. The irony of MIDI, 'Musical Instrument Digital Interface' is that it has led to the emphasis on performance being reduced. This is due to the sequencer. Some people do use the sequencer in a performance context, for instance the Utah Saints, but this is far from main stream.

The sampler

The sampler is a superb tool for composition but it takes imagination to turn it into a performance tool, something which is left to the user: You cannot just pick a sampling keyboard up and start playing. To a great extent the utility in playing a performance patch on a sampler comes from the novelty of the recorded sounds, more than the subtlety of expression with which these sounds can be controlled. As such, the patches can easily be overused.

The latest trend in synthesisers : control

There has been a trend over the last two years to revitalise the synthesiser market with keyboards that have better claim to being called performance instruments, both in the physical quality of controlling devices and more importantly the quality of the sound and its response to control.

The Yamaha VL1

The Yamaha VL1 is based on the waveguide technology developed at CCRMA by J.O.Smith primarily. See Smith (1992) for an introduction to waveguide synthesis. Waveguide synthesis is an efficient scheme for modelling acoustic instruments in which the wave motion is primarily in one dimension. All harmonic instruments are of this form, as harmonics are a product of a one dimensional wave equation. The art to waveguide synthesis is the incorporation of the control signals in to the model. Precise information on this is not generally available. The WX7 or WX11 windcontrollers are used, in addition to the modulation wheels, foot pedal and keyboard. The instrument is duophonic and very impressive. As well as delivering convincing imitations of real acoustic instruments, it can be used to generate completely abstract ones. Listen to the tape recording for examples. Yamaha plan to release a 16 note polyphonic version soon. It is worth mentioning the build quality and style of the instrument: It is much more expensive, and closer to the aesthetics of classical instruments than the conventional keyboard synthesiser.

The Korg Wavedrum

Yamaha have licensed their waveguide patents to Korg who released the 'Wavedrum'. This features a very high quality electronic drum pad, with an array of sensors under the skin for impact, pressure and scratch. Actually the scratch sensor is a microphone and so the resulting sound is not necessarily pure synthesis, but an interesting hybrid instead. A built in synthesiser unit can deal with several kinds of synthesis including digital simulation of analog synthesis. A two dimensional lattice structure of waveguides is used to approximate the wave equation on a drum skin. Again the instrument is superbly well-built.

The Korg Prophecy

The Prophecy has only just been released during the writing of this report, and has already been heralded as a classic instrument by some of the music press. It incorporates waveguide synthesis, but its real strength is in the control section. The physical controls include a bank of knobs, two modulation wheels, keyboard and a unique pressure pad, similar to those found on some new laptop computers. The pad can sense position and pressure. It also rotates about one axis. All the controllers can be mapped in a flexible, though not dynamic way, onto the various parts of the synthesiser and effects section. The effects section is comprehensive and includes an unusual harmonic emphasis algorithm.

The Novation Bass Station

The recent swell in demand for analog synthesisers, resulted in some manufacturers producing new digital hybrids with MIDI specifications. The leader is the Novation Bass Station. It has two digital oscillators, LFO, PWM, analog filters and a large bank of knobs each of which doubles as a MIDI controller. So it can be a performance instrument, and has the advantage that a performance can be recorded on a sequencer and later manipulated.

The Clavia Nord Lead Synth

Digital oscillators offer stability, but they are no good for simulating the quirks of the original analog synths. For this Clavia brought out a synthesiser physically modelling the analog circuitry of the original synthesisers.. This reintroduces a subtle temporal-complexity in the control which does not exist with Bass Station.

Other windcontroller-synthesiser instruments

AKAI have been producing a successful range of complex wind controllers with dedicated synthesisers, including analog synthesisers. At the high end of the market, some rather elaborate windcontrollers are produced: The Synthophone is a Yamaha acoustic alto saxophone with sensors and a MIDI interface. See McMahon (1995) for a review of other windcontrollers.

Modern performance trends

Many groups today are looking towards methods of hi-tech performance in which the electric guitar does not predominate. In electro-acoustic music there is a desire to involve performers more. One of the interesting problems here is that the notional instruments of electro acoustic music do not correspond well to conventional instruments.

2.2 What is a musical performance instrument?

To avoid restricting later considerations a musical performance instrument is defined in this report to be a physical device with human physical input and sound as output. The sound is monitored by the human(s). Additionally there maybe non-sound physical output such as touch pressure or light, which may also be monitored. For instance the vibration of a reed on a lip or the resistance of a key to movement. Input may also come from other non human sources. The sound output maybe be spatialised in some way.

Causality

The sound output at a given time has a non-zero statistical correlation to the human input up until that time (and zero afterwards!) Loosely speaking the player effects the sound. This model also covers an ensemble of players together, sound interaction occurs between players as well as their individual instruments. Figure 1 illustrates the human-instrument system for one person.

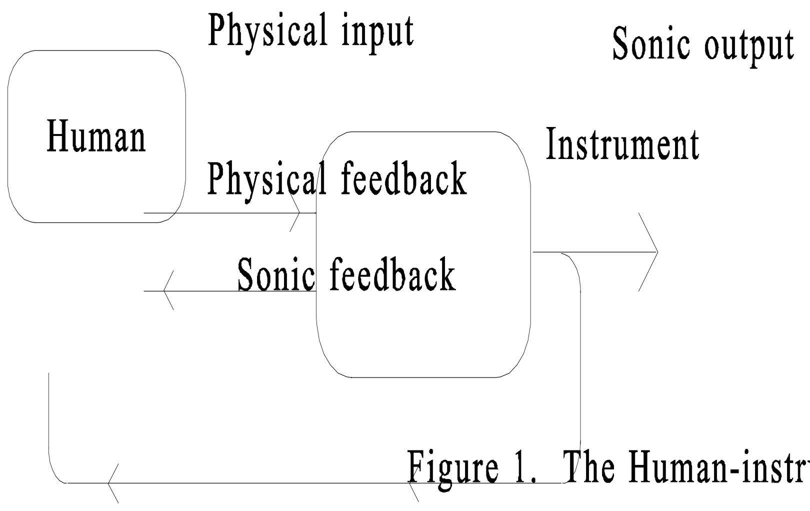


Figure 1. The Human-instrument system

2.2 Design criteria

What are the likely factors of a good instrument, one that players enjoy playing for its own sake and that writers are inspired to write for? What exactly is the instrument going to be used for? It is not necessarily the case that an instrument should be both a good solo instrument and a group instrument. A simple analog synthesiser can be very dull to play without other music. Ultimately fashion is a large factor in the success of an instrument. A new instrument which takes after an established instrument but has some novelties may be more popular than a very original instrument. On the other hand wacky oddballs may enjoy a brief popularity in some kinds of music. The following list identifies some characteristics of an instrument that are of general importance regardless of its intended use:

1. Physical appeal. It is a tremendous phycological boost to play something which looks beautiful and/or is well designed. Especially if it has unique qualities that make it individual even among instruments of its own type, as often with electric guitars. A lot of modern MIDI equipment falls down badly here: Every copy of a given model is exactly identical. Also, there often exist many different models by different manufacturers that are very similar.

The remaining points divide mainly into two types, those concerning the control process and those about the kind of sounds that can generated.

2. The control is improved by physical feedback to the player not only of the sound but also of, for example, touch on a keyboard, lip vibration from a reed.

3. Some precise timing control is necessary to execute rhythm. For instance the keys on a wind instrument provide precise timing control as well as pitch, but the timing control of the breath is much less precise.

4. Resolution. Fine changes in the input effect aspects of the sound output in subtle and repeatable ways. Really good players exploit the resolution, but this doesn't prevent lesser players from using,

and learning on the instrument.

5. Associating different controls with different aspects of the sound helps to make the control process intelligible to the player, and optimise use of different human muscle actions. However, the effect of different controls is often mixed to a small extent, as discussed earlier with saxophone: breath pressure effects pitch as well as volume. This can add interest to the instrument without making it incomprehensible and unlearnable. Again, good players exploit their knowledge of the subtle complexities of the instrument.

6. The sound responds dynamically to the input. The instrument can be modelled as a clocked state machine, with input from the physical controls. Acoustic instruments naturally behave in this way because the laws of physics are governed by differential equations. A simple example of the behaviour of a dynamic state instrument, would be the sound changing continuously without the input changing.

7. Unpredictability. The sound is difficult to reproduce exactly in every way. The pitching and timing may be very close, but other qualities may differ to different degrees. This maybe because the instrument is very sensitive to control in some aspect, or that additional random factors apply. For instance the crackly breath noise from a saxophone is definitely very random, where as the tonal variations in repeated piano chords are the result of sensitivity to initial amplitude and timing of each key.

8. Variety of contrasting sounds. For instance the large tonal variation of the oboe across its range, coupled with vibrato; a muted blues guitar rhythm line contrasted with a clear ringing, vibrato melody; a combination of "weak" and "strong" sounds. Subtly varied sounds are useful too. This ties in with point 4 above.

The emphasis should be on the response of the sound to the raw control input rather than the physical qualities of the control device. This is because once a musician has mastered the instrument, playing it takes his/her thought processes into an entirely abstract plane. Of course non-sound feedback of some kind is preferable: A little may go a long way to aiding the musician.

Chapter 3

Design Investigation

3.1 Scope of investigation

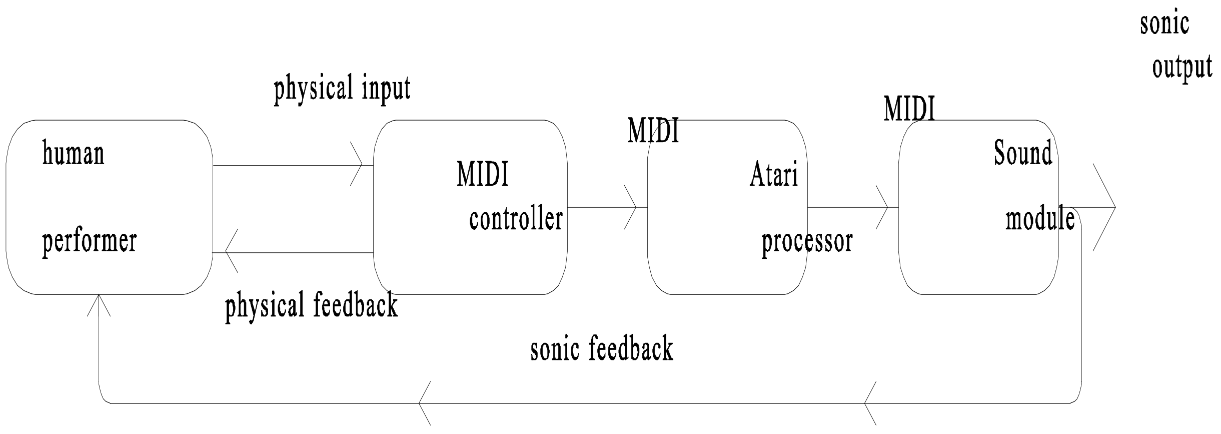
Following the remarks in the introduction, the following four classes of design are considered:

1. Using just MIDI equipment.

Here the aim is to use MIDI equipment in a novel way, or at least gain a better insight into the limitations. The performance controllers used are the Ensoniq EPS poly aftertouch keyboard and the WX7 windcontroller. These are not very good as physical controllers compared to a saxophone or a piano but they will serve the purposes of this project.

2. MIDI equipment + Atari ST processing of MIDI.

The system is illustrated in figure 2 below. The instrument has been split into a control processing stage driving a synthesiser. The aim is to enrich the control response of MIDI instruments and consider the wider possibilities of instrument design created.



p
e
r

Figure 2. Use of an Atari computer to process MIDI control signals

3. MIDI controllers + real time Csound running on an SGI Indy

Csound has been used for some time strictly as a compositional environment, despite its use of the term 'instrument'. The language is convenient for trying out synthesis methods, but awkward for implementing control processing. The idea is to try out things impossible with MIDI sound modules.

4. Sequenced spatial and delay processing using an SGI Indy

This is a look at the broader meaning of the term 'performance instrument', in which the instrument's behaviour changes radically over time.

The recordings

Note that the tape recording contains demonstrations of nearly all the instruments discussed below. The reader will find it helpful to listen to these in conjunction with the appropriate text.

3.2 MIDI equipment only

3.2.1 Use of the Ensoniq EPS in a live electro-acoustic piece

The piece "All strung up about nothing" was conceived for the electro-acoustic idiom using MIDI technology, in particular the Ensoniq EPS sampling performance workstation. The two most important qualities of the EPS which are used for the piece are:

1. The keyboard has self-calibrating poly aftertouch. Each key generates separate MIDI controller information depending on how hard the key is pressed. Self-calibration means that the keyboard keeps a fairly even response across the key range as it grows older. Poly aftertouch is quite rare even on expensive keyboards: Ensoniq were keen to produce a performance device.

2. Each controller can be assigned to a wide range of musical parameters.

The tape first contains excerpts from the piece which were each produced with a particular EPS configuration:

1. The original sample is of piano strings being strummed with the sustain pedal down. A note-on triggers this sample to play randomly forwards or backwards. The pressure on each key controls a pitch offset to that key. Slight variations of pitch create a powerful effect.

2. The original sample is the attack section from a bowed viola fifth. This has been looped and gated with a constant frequency amplitude envelope. The EPS output has been processed with a phasing effect. Staggering notes by small amounts creates the impression of an arpeggio. This makes a simple, yet interesting performance feature.

3. A coin dragged in circles on a wooden surface has been looped. Poly aftertouch is used to pan each key separately. This allows the player to move different sounds in different directions simultaneously.

4. The sound of a piano lid lifting has been looped. Filter cut off is controlled with aftertouch and pitch with a modulation wheel. A modulation wheel is used in preference to the default pitch wheel because it does not have centring springs. The noisy rhythmic nature of loop gives more an impression of increasing speed than pitch as the wheel is turned.

3.2.2 The Yamaha WX7 windcontroller

Here are the main characteristics of the WX7, before considering in detail its use:

1. The WX7 is based on the saxophone. The mouthpiece has a single plastic 'reed' on the underside. The reed does not vibrate. There is a right hand thumb rocker.

2. The fingering is similar to the saxophone, although the function of some extending keys has been changed to facilitate rapid playing. Extra octave keys greatly extend the range of the instrument.

3. A special button exists for sending program change messages in conjunction with the octave

keys.

4. A special 'hold' button exists for playing two part harmony, either with constant interval or fixed pedal note.

5. The WX7 can send two independent MIDI control messages determined by the breath flow and the reed control. In a real saxophone the breath pressure is the main determinant of volume. A problem occurs when clamping the WX7 reed tightly: The breath pressure stays high but the flow falls., and so the volume falls in an unnatural manner. Of course it is easy to get around such a difficulty with some computer processing of the MIDI signals. The thumb rocker merely adds to the reed control value, and as such is fairly redundant.

6. The MIDI message sends are coordinated as follows (This knowledge was gained by observing midibytes directly with an Atari ST): No messages are sent until the breath flow reaches a threshold. A volume message is sent at the lowest level first, followed by a note-on message whose velocity is determined by the initial rate of change of volume. Further volume messages are sent as the breath flow changes. If the fingering changes while the breath is held, a new note-on message is sent then a note-off for the old message.

7. A bank of dip switches and miniature pots can be readily changed with the aid of the small attached screwdriver. These are used for adjusting the response of the control signals to physical input, and setting which MIDI control numbers are sent. Use of computer processing makes these controls unnecessary.

3.2.3 WX7 windcontroller with a Korg M1, limitations

What can be done using a windcontroller with a MIDI synthesiser? A patch on the M1 was created for a wind instrument as follows: The amplitude envelope of a flute preset was changed to very sharp attack, level sustain and very fast release. Volume messages from the WX7 now alone determine the sound output volume. A background breath noise can be added by coupling the above voice with another similar voice whose sensitivity to volume control is turned off. The reed control can be directed straight towards pitchbend. Filtering is an alternative, or even both at the same time. The results are certainly useful but not compelling. It is difficult to put any temporal complexity into the instrument. Envelopes can be used but they are fixed, and are essentially just ways of creating different static sounds.

3.3 MIDI equipment with an Atari ST and Lattice C

Although much MIDI equipment quite programmable, much of this is through system-exclusive messages. These are typically 12 bytes long for a single parameter compared to 3 for a control change or note message. With continuous dynamics in mind the preference in the following designs is to use short messages, and make best use of the limited bandwidth available with MIDI. Another important factor is that on many synthesisers a note that has started is not effected by some of the more interesting sys-ex messages. For example the operator levels on the DX machines.

3.3.1 Arpeggiation instrument using the WX7 / Korg T3

Motivation

This is quite an odd instrument to start with. Arpeggiation is an often used device in MIDI keyboards. The idea is to provide performance control of a particular variant of arpeggiation in which the notes are calculated in modulo arithmetic.

Description of the instrument

The reed controls the speed of arpeggiation; closed is fast. Breath controls volume. The note value determines the increment between successive notes of the arpeggiation. The notes are divided into two sections: When a key is pressed in the lower three octaves, arpeggiation continues from the current note by the new increment. In the upper three octaves the next note is reset to the value of the pressed key. So the key pressed has a dual effect. The velocity of the note is also accented slightly.

Notes on the code, weird.c

The main loop is a free running, so that time is shared between processing any new MIDI and outputting the next note. 'count' is used as a simple means of controlling the amount of time between successive notes starting, whilst guaranteeing the MIDI buffer does not overflow. The note length is a function of the reed control value 'PB', found using a predefined table 'delay'. The table was set to give a comfortable 'feel' to the instrument: The constant range allows single notes to be played indefinitely, for a loose reed. As the reed tightens, the delay reduces gradually at first then faster, so that more control is possible at the slower more rhythmical speeds.

process.midi

This sets 'PB' from reed control, looks at note messages and breath control. The notes are divided into two sections : upper 3 octaves and lower 3. In the lower section the new note jump is calculated:

jump = data -73 +SPAN

73 ensures that the no hands WX7 position does not arpeggiate. **SPAN** prevents a negative value for **jump** so that the modulo operator, %, works correctly in **process_note**. If the lower section is being played, the current playing note, **pitch,** is reset with an accent.

The volume is calculated using reed and breath control:

data2*=(1+PB/64)

This compensates for reduced flow when the reed is closed, although pressure is being maintained. Its a rough function, but effective.

process_note
LOWER defines the first note of the range of output, **SPAN** is the number of notes in the range. **pitch %= SPAN** finds the remainder of pitch when divided by **SPAN,** and hence is used to wrap around the range.

3.3.2 A dynamic granular instrument using WX7 / SY55

Motivation

A technique used by the author for playing a MIDI synthesiser with high polyphony (~28), is to hold the sustain pedal down and shake the keys of a weighted keyboard preferably. As the keys played change, notes are stolen by the new notes from the old. One chord merges to another. This is particularly good for enlivening a simple string sound with an obvious loop. **granny.c** uses this idea on channel 2, where notes can overlap at the same pitch, then merge onto the next note.

Another idea in granny.c is that of using a poisson process to triggers note-ons and control the rate of the process with player controls. An example of a poisson process are the times of clicks heard from a geiger counter. Depending on the rate, a poisson process can be used to generate a texture, by triggering short sounds - 'grains', or an uncertain 'mixture' by triggering longer sounds. Using a poisson process circumvents the need to send continuous control information, which can be too much for MIDI, and creates a 'changing' quality in the sound discussed in chapter 2.

Finally there is the idea of having an element to the sound which is dynamically controlled contrasting with directly controlled sounds.

Description of instrument

There are two sounds; a low rumbling sound whose volume does note respond immediately to breath change, and a high slightly unsteady sound which responds fast, but leaves a trail of previous notes behind it. As a note is blown for longer the high sound becomes richer sounding.

Notes on the code, granny.c

Observe the notes on the SY55 user patch in the code header. It is important that these are correctly set to reproduce the instrument. Most important is that the sounds have no sustain, and therefore will finish without note-off. 'Note reserve' determines the richness of the high sound on channel2. The code should be fairly clear. **main_note** is the last note actually sounded, whereas **note** is the last key played.

process_midi

When a new note is received, then up to five of the notes played at the last pitch will be killed on channel 2, if that many were started. This prevents too much confusion, but allows notes to build up on one pitch. Breath control drives volume on channel2, and increments the dynamic velocity variable vel1 used by channel1 in **grains_out.**

grains_out

Two poisson processes are implemented here. The **step** function is for convenience, and anticipates further work with other similar functions. **step** goes from low at low input values to high at high input values, in a step shape: Channel1 sounds are more likely with increasing **vel1,** channel2 with increasing breath control.

dynamics

The dynamics of **vel1** are very simple but effective. **vel1** 'leaks' at a constant rate and breath control 'tops' **vel1** up. Obviously the upper and lower levels have to be limited. On reflection it would be an interesting to make the 'leak' a function of **vel1;** then a steady breath value would eventually lead to a steady **vel1** value other than zero or maximum.

3.3.3 A dynamic additive synthesis instrument using 2 KAWAI K1s

Motivation

The K1 conveniently has 13 sine wave harmonics as presets. There have been many keyboards produced with drawbars for controlling harmonics. Here the idea is to use the lower keys on a keyboard to interact dynamically with the harmonic levels, whilst the upper keys play notes consisting of these harmonics.

Instrument description

One of the k1's is used as the controlling keyboard. The lowest octave starting at C is used to effect the harmonic levels: Hitting a key hard makes the corresponding harmonic level rise faster. Repeated hitting adds to the 'velocity'. The velocity is being leaked so eventually the harmonic level returns to zero. The upper keys form a monophonic keyboard which plays notes consisting of the mixed harmonics.

Notes on the code, additive.c

The principals are very similar to the previous programs, with slight reorganisation. The dynamics are more complex, as the harmonic level 'velocity', **hvel**, is a dynamic variable aswell as the harmonic level itself, **hlevel.** It is very important for the perceived unity of a tone that the harmonics do not change relative to one another to fast.

3.4 MIDI controllers with an SGI Indy running Csound

The use of Csound in realtime does not appear to be well documented. The unexperienced reader is referred to the Appendix for an introduction to some general techniques which are important.

3.4.1 A bird-like instrument using the WX7

The three programs **birdy1.orc, birdy2.orc, birdy3.orc,** were produced in a very experimental manner. The key idea is to apply a damped resonant filter to the control signals. The second two programs are just different combinations of two resonant filters.

It is important to note that only sine waves are used. All the real processing is done at the control rate.

Instrument description

birdy1.orc

Sudden changes in breath cause a 'ripple' on the pitch output. Closing the reed raises the resonant frequency from 0 to just sub audio. With an open reed the pitch can be controlled by breath. With a closed reed the pitch can only be controlled by the keys.

birdy2.orc

This is similar to **birdy1.orc,** except 2 sine tones mix. For a closed reed they are a fifth apart. Their interval for an open reed is variable and difficult to control exactly. This provides some interest.

birdy3.orc

Similar to **birdy2.orc,** except that the tones are in unison when the reed closes, and one tone ripples much less.

Notes on the code, birdy1.orc birdy2.orc birdy3.orc

The orchestras are divided into the MIDI collection instrument and the control processing and synthesis instrument. This is so that a continuous, unbroken, sine tone can be generated despite the midi instrument switching on and off. The smoothed breath control, **gkv,** is resonantly filtered by applying it as the driving force to a damped simple harmonic oscillator. Global variables **gkx, gky** are used to integrate the differential equations. The input is subtracted from the 'position', **gkx,** to generate a pitch offset for the sine tone. The resonant frequency is changed by controlling the integration time increment, **gkdt,** from the reed control.

3.4.2 A whistle-like instrument using the WX7

This instrument arises from the observation that filtered noise produces a very natural tone. A minimal amount of control is required to create a convincing whistle sound.

Instrument description

At low breath the sound is noisy and rough. As breath is increased, the pitch rises a little and the tone becomes more focused. Switches between notes slightly overlap. At higher breath still a sequence of harmonics are mixed in.

Notes on the code, whistle.orc

The noise filter is fourth order achieved using to **'reson's.** Removing one of the resons gives a very breathy sound. A little dynamic variation is applied across the scale: High notes respond faster and have less overlap than low notes. The overlap is deliberate here in contrast with birdy1.orc. It is achieved using **linenr** which extends an instrument duration beyond the note-off.

3.4.3 A brass-like instrument using the WX7

Waveshaping is a very efficient method of harmonically distorting a signal, and is therefore worth investigating for realtime.

Instrument description

The breath controls volume and timbre off the sound, which becomes brighter at higher volumes. The reed effects pitch. The lower keys have sluggish response compared to the upper keys. There is a slight 'attack' when switch between notes. An interesting effect occurs when the breath increases sharply: Instead of perceiving a steady change in timbre, the sound takes on a kind of steady 'transition timbre'.

Notes on the code, wave.orc

The orchestra is structured into midi collector and synthesiser as for **birdy.** **gbuzz** is used for its rich harmonic content, which interacts more when waveshaped. The values used were found by experimentation. Some other options are commented out in the code. Two copies of the wave synthesiser play together, but each is controlled by slightly different dynamics. This helps to enrich the sound and add dynamic interest.

3.4.4 A conga drum using a keyboard

This is not an attempt to make a new instrument, but a way of showing how a keyboard can be used in an unusual way.

Instrument description

On the left of the keyboard a low conga sound plays, on the right a high one. In between there is a gradual cross over in sound. Hitting a key and releasing a key quickly results in a resonant conga sound. While the key is held the sound becomes progressively damped. If using aftertouch, pressing on the key further increases the damping.

Notes on the code, conga.orc

The 2 conga samples are loaded into tables initially. Fairly elaborate use is made of **linenr** to regulate the damping. The first **linenr** forms a variable to gate the damping process. Ideally the attack and decay times should be zero, but **linenr** ceases to work then. The second **linenr** extends the instrument life to the sample length. This is a little inefficient because the note may damped to zero volume prematurely, which is important if many short notes need to be played.

This simple routine provides a kind of control which is not available on synthesiser keyboards.

3.4.5 A filter bank using a poly aftertouch keyboard

This is more of a concept-instrument. It is inspired by the filtered harmonies of a waterfall in 'Riverrun' by Barry Truax. The idea is to control this process using a poly aftertouch keyboard.

Instrument description

With no keys pressed the player hears a natural sound; a stream running or wind blowing. Pressing a key down progressively harder, results in a whistling pitch rising from the background sound. Up to three notes can be played at once using the SGI Indy. Subtle control is required so that the sound does not become too prominent.

Notes on the code, natural.orc

The code is very straight forward. Each key press uses a single 2nd order filter. The original signal is mixed in to provide a better spectral balance.

3.5 Sequenced spacial sound processing on the SGI Indy in C

The following programs are not closely related to the main body of work above, but they do hint at future possibilities for performance instruments.

3.5.1 A four-speaker delay

This is simply a delay with feedback in which the four speakers take their signals from four equally spaced taps on the delay line. The interest of spatialisation added to the echo effect indicates the possibilities here.

3.5.2 An implementation of 'Solo' by Stockhausen

Solo, by Stockhausen, is a performance piece for a single instrumentalist with microphone. The microphone signal is processed with a stereo tape delay line. The delay and feedback connections are altered by two manual operators according to a written score. The stereo signal can be switched into a set of four speakers placed in the corner of the performance space. The switches and volume levels are adjusted by a second 'performer'. **solo.c** is an implemention of this piece for the SGI Indy. The input comes directly from the microphone socket, and the output goes directly to the 4 channel audio output.

Chapter 4

Ideas For Future Work

Dynamic control opens the door to many possibilities. The dynamics considered in the project designs were very simple, the most complex being the driven simple-harmonic-oscillator. It is possible more-complex dynamics could be of value, in particular mathematically-chaotic dynamics. Note that the total dynamic system includes the performer, who is difficult to quantify. This is why a simple driven oscillator works well. Some further ideas are listed:

1. Dynamically relating a set of harmonics.

2. Applying dynamics to pitch offsets of harmonics.

3. Performance use of granular synthesis could be taken much further using Csound than with MIDI. The 'randh' command can be used as a random number generator.

4. System exclusive has not been used in the project designs, for the reasons given in 3.3 There is plenty of scope for their use especially with granular techniques as this involves less data flow than continuous control.

5. A performance-instrument design language
Csound is awkward for control processing, but transparent for synthesis building. It would be possible to process MIDI signals in C before passing to Csound. This could be done externally with an Atari or, better, by running two linked processes on the Indy. Better still would be a single language combining orchestra design with C programming.

The project has been involved exclusively in the development of the control processes rather than the development of audio rate code. Waveguide synthesis is an example of successful new audio rate synthesis. It is hoped that many more possibilities exist in the audio rate domain.

Another important domain unexplored by this project, is the physical design of performance instruments. However some suggestions can be made for improving the design of the WX7 windcontroller:

1. The reed should have a lip position sensor on the underside. This can be achieved easily using a contact resistance sensor or pressure sensor.

2. The thumb wheel should have its own control number and not be linked to the reed as this is wasteful.

3. There should be a mode of MIDI transmission in which each key behaves like the key of a normal MIDI keyboard. With computer processing of the MIDI signals this would allow the sort of control found on a saxophone, where the fingering may be adjusted the tone not the pitch.

Chapter 5

Conclusion

The process of design by experimentation exhibited in this project has demonstrated the validity of some of the initial hopes: Dynamic control processing is very worthwhile and should be further investigated. 'Changing sounds' can be generated in simple ways such as in **granny.c** The answer to the question posed in the introduction 'Can temporal complexity of control, used with a static sound make a successful instrument?' is decidedly in the affirmative, with results from **birdy1.c** . Regarding MIDI equipement: It is less than transparent to use in any but the most straight forward way, and highly machine dependent. However, with perseverence MIDI synthesisers can be a valuable tool in performance instrument design. The situation may become more favourable if the new 'Zippy' standard becomes widely adopted.

Overall, the emphasis has been on combining many different elements of design to produce a successful instrument rather than pinning hopes on a single grand idea. In this sense performance instrument design resembles 'composition in possibility'.

Bibliography

Baker GL, Gollub JP

1990 CUP

"Chaotic Dynamics"

Donnington R

1970 Methuen

"The Instruments Of Music"

Experimental Musical Instruments

gopher://echonyc.com/11/Music/MO/EMI

Jenkins JC

1983 Royal Scottish Museum

"Survey Of Non-European Instruments"

Krishnaswarmy S

1971 Crescendo

"Musical Instruments Of India"

Remmant M

1978 Batsford

"Musical Instruments Of The West"

Sawyer D

1977 CUP

"Making Unorthrodox Musical Instruments"

Smith JO

1992 Computer Music Journal Vol 16 No 4

"Physical Modelling Using Digital Waveguides"

Appendix A

Contents of the tape recording

Use of MIDI equipment alone

"All Strung Up About Nothing" excerpts:

1. Piano strum with poly aftertouch controlling pitch.

2. Viola fifths with constant frequency amplitude envelope.

3. Coin drag with poly aftertouch controlling panning.

4. Lid sound with wheel controlling loop speed and aftertouch controlling filter cut off.

"All Strung Up About Nothing" complete.

Use of MIDI equipment with Atari ST processing

1. Demo of program **weird.c** used with preset sound A22 on a Korg T3, controlled using a Yamaha WX7.

2. Demo of program granny.c used with a special user multi patch on an SY55, controlled using a Yamaha WX7.

Use of MIDI controllers with Csound running on an SGI Indy

1. Demo of Csound orchestra **birdy1.orc** using a WX7.

2. Demo of **birdy2.orc**

3. Demo of **birdy3.orc**

4. Demo of **whistle.orc** using a WX7. This is a filtered noise instrument. First you hear it quietely, then louder with harmonics.

5. Demo of **wave.orc** using a WX7. This instrument uses wave-shaping.

6. Demo of **conga.orc** using a keyboard controller. Holding a key down rapidely dampens the drum sound.

7. Demo of **natural.orc** using a polyphonic aftertouch keyboard (the Ensoniq EPS).   Pressing a k

   Wind
   Sea
   Stream

Commercial synthesisers

1. The Yamaha VL1; a duophonic waveguide synthesis instrument. 3 instruments are recorded. A

   Shakahachi

   Oboe

   Abstract (2 part harmony)

2. The Korg Wavedrum;' a waveguide synthesis drum. 4 presets are recorded:

   Raindrum

   Sawari A

   Syn Tone

   Scratch

3. The Korg Prophecy;

   Roland 101 and 303 style bass

   Sync sweep

   Metallic JX

   Virtual saxophone

   Analogue guitar

   Electric bass

Appendix B

A primer on the use of Csound in realtime

First an explanation of the special command line used:

```
csound -dm0 -o devaudio -M /dev/ttyd2 -b512 -B512 midi5.orc midi.sco
```

-dm0 turns off as much text output as possible, to prevent glitches.

-o devaudio directs the sound output to computer audio output.

-M /dev/tty2d2 collects MIDI information from serial port 2

-b 512 the input buffer size is kept small to improve response times.
Of course this means audio glitching is more likely, and so is a compromise.

-B 512 the output buffer.

midi.sco is a normal score file with at least one instrument that is switched on at time 0 and lasts for atleast the duration of the performance. This can be an empty instrument. It is possible to mix performance instruments with sequenced instruments, or even sequence aspects of a performance instrument..

midi.orc is the performance instrument 'engine'. Instrument numbers 1..16 are hard-wired to be triggered by note-on messages received on MIDI channels 1..16. Normally an instrument call terminates when a note-off message is received with the same note value that started it. If several note-ons are received on the same note and channel, they form a stack which is cleared by several note-offs. linenr is useful for extending the life of one of the instruments. Within instruments 1..16, various MIDI parameters can be assigned to control variables and initial variables. The current Csound manual contains the full details. Sadly, it does not yet appear fully implemented for receiving control messages, at least for the SGI Indy. However, controller 7 can be read with **chpress;** mono and poly aftertouch with **aftouch;** and pitch-bend with **pchbend.**

Control rates

The MIDI control signals only have a resolution of 128, so before applying to control dynamics or synthesis, they must be filtered otherwise glitches will appear in the audio output. There is not really a wholly satisfactory way of achieving this easily. Depending on the application good results can be achieved with **port.** This can be used to do dynamics processing as well if long time constants are used. One draw back of **port** is that it resets its initial value each time an instrument starts using it. This can be circumvented by emulating **port** with an expression using global variables:

gkout = gkout + (gkin - gkout) / giconstant

Global variables

These are useful generally for tying several Csound instruments together into a group-instrument, and providing continuity in the audio output. The latter is especially important if dynamic processing of the control input is required.

Appendix C

A complete listing of the code

Index

weird.c

```
/**   weird.c                                    **/
/**   A windcontroller-arpeggiator instrument     **/
/**   Use the WX7                                **/
/**   Dylan Menzies-Gow August 95                **/


#include <mus_libd.h>

#define  K qwerty_input()
#define  LOWER   20
#define  SPAN   70

void fill_table(int *);
void process_midi(void);
void process_note(void);

int delay[128];    /* Table for calculating note length from reed control */
int jump=0;           /* Current jump between successive notes */
int PB=64;         /* Current reed control value */
int pitch=73;      /* Initial pitch = hands off pitch */
int count=0;       /* Counter used for giving length to each note */
int vel=0;         /* Velocity of last MIDI note-on received */
int key;           /* Current key being pressed */

void main( void )
{
    fill_table(delay);
     clear_midi_buffer();
     prog_change(1,5);          /*  Windchimes on the Korg T3 */
     control_change(7,1,0);     /*  Reset volume to zero */

     while( !K )
     {
          process_midi();          /* Receive and process and MIDI input */

          count++;
          if (count > delay[PB])
               process_note();    /* Calculate next note and send MIDI */
     }
}


void process_midi(void)
{
     int type, data1, data2, channel;

     type = get_midi_event(&data1, &channel, &data2);

     switch( type )
     {
          case PITCH_BEND :       /* Reed control */
          {
               PB = data2;
               break;
          }
          case NOTE_EVENT : if (data2>0)
          {
               vel = data2/2;
               key = data1;
               if (data2>0) jump = data1-73+128;
/* 73 -> no hands gives zero jump */

/* 128 -> an aid to modulo arithmetic later */

/* ( % does not work with negative numbers */
               if (data1>=0x4a)   /* Reset the current note value */
               {
               midi_note(pitch,1,0);
                  pitch = data1-12;
                  midi_note(pitch,1,90+vel);  /* Give accent to 'special' note */
               }
               break;
          }
          case CONTROL    : if (data1==7)
          {
               data2*=3;
               data2*=(1+PB/64);  /* Harder to blow down norrower gap */
                                  /* -this compensates reduced flow */
               if (data2 > 127) data2=127;  /* Limit volume within MIDI range */
               if (data2 < 5) data2=0;
               control_change(7,1,data2);
          }
     }
}


void process_note()
{
     count=0;     /* Reset timer */

     if ( jump != 128 )
     {
          midi_note(pitch,1,0); /* Kill last note */

          pitch -= LOWER;      /* Calculate next note by adding 'jump' modulo */
          pitch += jump;       /* 'SPAN' with offset 'LOWER' */
          pitch %= SPAN;
          pitch += LOWER;
          midi_note(pitch,1,64+vel); /* Start new note */
     };
}


void fill_table( int *table)
{
     int i,j;
     for(i=0; i<128; i++)
     {
          j = i;
          if (j<64) table[i] = 1000000
          else table[i] = 1000/(j-63);
     }
}
```

granny.c

```
/**   granny.c                                    **/
/**   Dynamic-granular-windcontroller instrument  **/
/**   Use the WX7                                 **/
/**   This program was initally written for a     **/
/**   multi patch on a Yamaha SY55. The key       **/
/**   features of the patch are :                 **/
/**   Channel 1 : a short attack and decay        **/
/**   envelope on a breathy sound         **/
/**   Note reserve = 16                           **/
/**   Channel 2 : a much longer attack and decay  **/
/**   on a 'digital' sound.                       **/
/**   Note reserve = 7                            **/
/**   Dylan Menzies-Gow, August 95                **/


#include <mus_libd.h>

void dynamics(void);
void grains_out(void);
void process_midi(void);
int bump(int);
int step(int);

int BC;                    /* Latest breath control value */
int PB;                    /* Latest reed control value */
int note              /* Latest key pressed */
int main_note = 0;    /* The last note actually played */

float vel1=0;         /* The velocity on channel 1, a dynamic variable */


void main(void)
{
     int wait;

     control_change(7, 1, 127); /* Volume on channel 1 stays constant at maximum */

     while(!qwerty_input())
     {
          process_midi();         /* Receive any MIDI and process */

          grains_out();           /* Calculate if sound shall be output, then output */

          dynamics();                /* Process the dynamic variable(s) */

          for(wait=1; wait<5000; wait++);  /* Pause */
     }
}
```

```
void process_midi(void)
{
      int event, data1, data2, channel, i;

      event = get_midi_event(&data1,&channel,&data2);

      if (event==NOTE_EVENT)
      {
             if (data2>0) note=data1;  /* Note on */
             else
             {
                  for(i=0; i<5; i++) midi_note(main_note,2,0);
                  main_note=0;          /* Note off */
      /* 5 of the last notes to start, at the same pitch, on channel 2 are killed */
             }
      }

      else if (event==CONTROL && data1==7)
      {
            BC=data2;
            vel1 += data2 * 0.05    /* Dynamic velocity is effected by breath here *?/
            if (vel1 > 127) vel1 = 127; /* Upper velocity limit */
            control_change(7, 2, BC);  /* Set volume on channel 2 */
                                       /* This is not dynamic */
      }

      else if (event==PITCH_BEND)
            {
                  PB=data2;
                  pitch_bend(2, PB/2);  /* Reed control of pitch on channel 2 */
            }
}


void grains_out(void)
{
      /* Notes are started according to a pseudo-poisson process */
      /* ( like geiger counter clicks ) */

      if ( rand()<(int)(step(vel1) ) )/* Note start more likely if vel1 is bigger */
            midi_note(40, 1,vel1);        /* Use dynamic variable vel1 */

      if ( rand()<(int)(step(BC)*0.02) )
      {
            midi_note(note, 2,127);
                  /* Velocity is fixed, but volume is controlled in process_midi */
            main_note = note;          /* Register that a note has been started */
                                       /* on the current key value */
      }
}

int bump(int x)    /* A convenient function for use by grains_out */
{
     if (x>0 && x<64) return(x*512);
     else if (x>=64 && x<=127) return((127-x)*512);
     else return(0);
}

int step(int x)
{
     if (x>0 && x<64) return(x*512);
     else if (x>=64) return(64*512);
     else return(0);
}


void dynamics(void)
{
     vel1 -= .2;     /* vel1 'leaks' away */
     if (vel1 < 0) vel1 = 0;  /* Limit velocity to MIDI range */
}
```

additive.c

```
/**   additive.c                                        **/
/**   Dynamic-additive-synthesis-keyboard instrument    **/
/**   Intended use with 2 K1s both in combination mode  **/
/**   with harmonic presets 1..13 set to channels 1..13 **/
/**   Dylan Menzies-Gow, August 95                      **/


#include <mus_libd.h>

#define RESET 61   /* The reset key */
#define PITCH_BASE 36      /* First note of the pitch control section */
#define LEVEL_BASE 48      /* First note of the level control section */
#define PLAY_BASE 62       /* First note of the playing section */
#define PAUSE 2000
#define HDECEL 4   /* Upper limit of deacceleration to levels */
#define HVELMIN -60        /* Minimum rate of change of levels */
#define HNUM 13      /* Number of harmonics used */
#define HMAX 3200    /* Upper limit of levels */

void set_harms( void );
void process_midi( void );
void adjust_harms( void );

int hlevel[16], hvel[16];

int count =0;

int key_decode[] = {1,0,2,0,3,4,0,5,0,6,0,7,8};

void main(void)
{
     int wait;

     set_harms();    /* Initial dynamic state of levels */

     while(!qwerty_input())
     {
          process_midi();      /* Process any MIDI messages */

          adjust_harms();      /* Dynamically adjust harmonic levels */

          for(wait=0; wait<PAUSE; wait++);

     }
}

void process_midi(void)
{
     int event,i;
```

```
          int data1, data2, channel;

          event = get_midi_event(&data1,&channel,&data2);

          if (event == NOTE_EVENT)
          {
                if (data1 == RESET) set_harms();

                else if (data1 >= PLAY_BASE)          /* Play-section */
                {
                      for(i=1; i<=HNUM; i++)  /* Echo note info across the harmonics */
                        midi_note(data1-24,i,data2);
                }

                else if (data1 >= LEVEL_BASE)
                {
                      i = key_decode[ data1-LEVEL_BASE ];        /* Improve ergonomics */
                      if (i > 0)
                        hvel[i-1] += data2*5;   /* Increase the velocity in proportion */
                                                      /* to note velocity */
                }

          }
    }
```

```
void adjust_harms( void )
{
     int temp, i;

     for(i=0; i<HNUM; i++)   /* Adjust each harmonic in turn.. */
     {
          temp = (int)hlevel[i];
          hlevel[i] += hvel[i];   /* Apply rate of change */
          if (hlevel[i] > HMAX) hlevel[i] = HMAX;
          if (hlevel[i] < 0) hlevel[i] = 0;

          hvel[i] -= HDECEL;      /* Apply acceleration */
          if (hvel[i] < HVELMIN) hvel[i] = HVELMIN;


          if (temp != (int)hlevel[i])
          control_change(7, i+1, hlevel[i]>>8);

     }
}



void set_harms( void )
{
     int i;
     for(i=1; i<=HNUM; i++)     /* Set initial harmonic levels */
     {
          control_change(7,i,0);
          hlevel[i-1] = 0;
          hvel[i-1] = 0;
     }

}
```

mus_libd.h

```
                     /********************************/
                     /* GENERAL PURPOSE MIDI LIBRARY */
                     /********************************/
                     /*      A.D. Hunt  16/09/94     */
                     /*  Changes D.Gow  27/7/95      */
                     /*                              */
                     /********************************/

/* Include Standard Headers */

#include <stdio.h>
#include <stdlib.h>
#include <gemlib.h>
#include <osbind.h>
#include <math.h>



#define TRUE  1
#define FALSE 0

#define WAIT 0
#define IMMEDIATE 1

#define NOTHING    0
#define NOTE_EVENT 1
#define PITCH_BEND 2
#define CONTROL    3
#define POLY_PRESS 4

#define NEXT_MIDI (unsigned char)Bconin(3)&0xFF

/* FUNCTION DECLARATIONS */

int  random(int, int);
void midi_note(int, int, int);
int get_midi_note(int *, int *, int *);
int get_midi_event(int *, int *, int *);
unsigned char get_next_midi(int);
void clear_text_buffer(void);
short qwerty_input(void);
char input_char(void);
void Move_cur(char, char);
long timer(void);
void pause(int);
void main(void);

void pitch_bend(char, char);
void control_change(char, char, char);
void prog_change(char, char);

            /*********************************************/
            /* THE SOURCE CODE FOR THE LIBRARY FUNCTIONS */
            /*********************************************/


/* RANDOM NUMBER GENERATOR */

int
random(minm, maxm)
   int minm, maxm;
{
   static short first_time = TRUE;
   unsigned short value;
   int range, rand_val;

   if (first_time) {
      srand( timer() );
      first_time = FALSE;
   }

   value = rand();
   range = maxm - minm + 1;
   rand_val = (((long)value * range) / 32768) + minm;

   return (rand_val);
}
```

```
/* MIDI NOTE PLAYER */

void
midi_note( pitch, channel, velocity )
    int pitch, channel, velocity;
{
  unsigned char midiword[3];
  midiword[0] = 0x90 + (unsigned char)channel - 1;
  midiword[1] = (unsigned char)pitch;
  midiword[2] = (unsigned char)velocity;
  Midiws(2, midiword );
}

void
pitch_bend( channel, bend )
    char channel, bend;
```

```
{
  unsigned char midiword[3];
  midiword[0] = 0xE0 + (unsigned char)channel - 1;
  midiword[1] = 0;
  midiword[2] = (unsigned char)bend;
  Midiws(2, midiword );
}


void
control_change( num, channel, val )
    char num, channel, val;
{
  unsigned char midiword[3];
  midiword[0] = 0xB0 + (unsigned char)channel - 1;
  midiword[1] = (unsigned char)num;
  midiword[2] = (unsigned char)val;
  Midiws(2, midiword );
}


void
prog_change( channel, prog )
    char channel, prog;
{
  unsigned char midiword[2];
  midiword[0] = 0xC0 + (unsigned char)channel - 1;
  midiword[1] = (unsigned char)prog;
  Midiws(2, midiword );
}



/* FETCH NEXT MIDI NOTE EVENT */

int
get_midi_note(pitch, channel, velocity)
         int *pitch,*channel,*velocity;
{
   static unsigned char last_status = 0;
   unsigned char midibyte;
   int note_received = 0;
   int more_to_get = 1;

   while(more_to_get) {

      if(Bconstat(3)) {             /* If there's MIDI present */

         midibyte = get_next_midi(IMMEDIATE); /* Fetch the byte */

         if (midibyte != 0) {  /* If there's still a valid byte there */
            if(midibyte >= 128) {      /* It's a STATUS byte */
```

```
                last_status = midibyte;
                if((midibyte&0xE0) == 0x80) {   /* Note Event */
                   note_received = 1;
                   *channel = midibyte&0x0F;
                   *pitch = get_next_midi(WAIT);
                   *velocity = get_next_midi(WAIT);
                }
             }
             else {                     /* This is a DATA BYTE */
                if((last_status&0xE0) == 0x80) {   /* Last Status was Note Event */
                   note_received = 1;
                   *channel = last_status&0x0F;
                   *pitch = midibyte;
                   *velocity = get_next_midi(WAIT);
                }
             }
          }

          if (note_received) {  /* Turn note OFF's into velocity 0 */
             more_to_get = 0; /* Stop now - we've got a note event */
              *channel += 1;  /* Set to USER channels (ie 1 - 16) */
             if ((last_status&0xF0) == 0x80) {
                *velocity = 0;
             }
          }
       }
       else more_to_get = 0;  /* Stop now, as there's no more MIDI */
    }
    return(note_received);
 }



 int
 get_midi_event(data1, channel, data2)
          int *data1,*channel,*data2;
 {
    static unsigned char last_status = 0;
    unsigned char midibyte;
    int type_received = NOTHING;
    int more_to_get = 1;

    while(more_to_get) {

       if(Bconstat(3)) {             /* If there's MIDI present */

          midibyte = get_next_midi(IMMEDIATE); /* Fetch the byte */

          if (midibyte != 0) {  /* If there's still a valid byte there */
             if(midibyte >= 128) {      /* It's a STATUS byte */
                last_status = midibyte;
                if((midibyte&0xE0) == 0x80) {   /* Note Event */
```

```
         type_received = NOTE_EVENT;
         *channel = midibyte&0x0F;
         *data1 = get_next_midi(WAIT);
         *data2 = get_next_midi(WAIT);
      }
      else if((midibyte&0xF0) == 0xE0) {   /* Pitch Bend */
         type_received = PITCH_BEND;
         *channel = midibyte&0x0F;
         *data1 = get_next_midi(WAIT);
         *data2 = get_next_midi(WAIT);
      }
   }
   else {                     /* This is a DATA BYTE */
      if((last_status&0xE0) == 0x80) {   /* Last Status was Note Event */
         type_received = NOTE_EVENT;
         *channel = last_status&0x0F;
         *data1 = midibyte;
         *data2 = get_next_midi(WAIT);
      }
      else if((last_status&0xF0) == 0xE0) {   /* Last Status was PB */
         type_received = PITCH_BEND;
         *channel = last_status&0x0F;
         *data1 = midibyte;
         *data2 = get_next_midi(WAIT);
      }
      else if((last_status&0xF0) == 0xB0) {
         type_received = CONTROL;
         *channel = last_status&0x0F;
         *data1 = midibyte;
         *data2 = get_next_midi(WAIT);
      }
      else if((last_status&0xF0) == 0xA0) {
         type_received = POLY_PRESS;
         *channel = last_status&0x0F;
         *data1 = midibyte;
         *data2 = get_next_midi(WAIT);
      }

   }
}
```

```
           if (type_received == NOTE_EVENT) {  /* Turn note OFF's into velocity 0 */
            if ((last_status&0xF0) == 0x80) {
               *data2 = 0;
            }
         }
         if (type_received != NOTHING) {
            more_to_get = 0; /* Stop now - we've got a note event */
            *channel += 1;  /* Set to USER channels (ie 1 - 16) */
         }
      }
      else more_to_get = 0;  /* Stop now, as there's no more MIDI */
   }
   return(type_received);
}




unsigned char
get_next_midi(type)
          int type;  /* Return immediately, or wait  ?? */
{
unsigned char inmidi;
   do {
      if(type == IMMEDIATE) {
         if(Bconstat(3)) {   /* Only get a byte if it's there */
            inmidi = NEXT_MIDI;   /* collect next MIDI byte */
         }
         else {   /* No more MIDI in buffer */
            inmidi = 0;
         }
      }
      else {
         inmidi = NEXT_MIDI;  /* WAIT for next byte */
      }
   } while ((inmidi & 0xF8) == 0xF8); /* REAL-TIME */

   return (inmidi);
}




void        /* Clears keyboard buffer of any previously typed characters */
clear_text_buffer()
{
   while(Bconstat(2))Bconin(2);
}

short             /* Returns true if there is anything in keyboard buffer */
qwerty_input()
{
   return((short)Bconstat(2));
}


char       /* Inputs a character from qwerty keybd- presuming one's there */
input_char(void)
{
   return((char)Bconin(2));
}

void                       /* Moves cursor to a specific screen location */
Move_cur(char line,char column)
{
   Bconout(2,27);  /* ESCAPE */
   Bconout(2,'Y');
   Bconout(2,32+line);
   Bconout(2,32+column);
}


long
timer()
{
   register long *save_ssp = (long *)Super(0L);
   register long time_value = *(long *)0x4ba;
   Super(save_ssp);
   return time_value;
}


void
pause(value)
  int value;
{
   long target = timer() + value;
   while(timer() < target) {
      /* Do nothing now - but add your own stuff here if necessary */
   }
}
```

midi.c

```
/**   midi.c                                         **/
/**   A program for monitoring decoded MIDI messages  **/


#include <mus_libd.h>

void main(void)
{
     int type, data1, data2, channel;
     int q=0, i=0;

     char name[5][10] = { NOTHING, NOTE_EVENT, PITCH_BEND, CONTROL };

     clear_midi_buffer();
     while( !q )
     {
          while ( (type = get_midi_event(&data1, &channel, &data2)) == NOTHING
                          && !(q=qwerty_input()) );
          printf("\ntype/channel/data1/data2/i = %d,%d,%d,%d,%d"
          ,type,channel,data1,data2,i);
          i++;
     }
}
```

midib.c

```
/**   midib.c                                            **/
/**   Prints individual bytes received by the MIDI port  **/


#include <mus_libd.h>

void main(void)
{
     int byte;

     while( !qwerty_input() )
     {
          byte = get_next_midi(WAIT);
          printf("%x\n",byte);
     }
}
```

Csound instruments

The following score, **midi.sco**, is used with all the orchestras.

```
f1 0 8192 10 1
f2 0 8192 10 4 4 3 3 3 2 2 2 2 1 1 0 1 0 0 1
f3 0 8192 3 -4 4 10 0 30 40 0 20
f4 0 8192 7 -1 2000 -1 4192 1 2000 1
f5 0 8192 10 0 0 1
f6 0 8192 10 1 0 0 1
f7 0 8192 10 0 0 0 0 0 1
f8 0 8192 10 1 0 1 0 1

f10 0 512 7 0 100 0 300 1 100 1
f11 0 512 7 0 190 0 20 1 20 0 190 0
f12 0 8192 7 1 250 1 250 0

f20 0 16384 1 "conga1.aiff" 0 0 0
f21 0 16384 1 "conga2.aiff" 0 0 0
f22 0 16384 1 "conga22.aiff" 0 0 0
f30 0 512 7 0 200 0 100 1 200 1

i100 0 3600
```

The command line

```
csound -dm0 -o devaudio -M/dev/ttyd2 -b512 -B512 midi5.orc midi.sco
```

birdy1.orc

```
;  birdy1.c
;  A windcontroller-sine tone instrument
;  A single sine tone is dynamically controlled,
;  using breath and reed control data.
;  Dylan Menzies-Gow, August 95

sr = 20000
kr = 5000
ksmps = 4
nchnls = 1

ga1   init  0
gkx   init  4   ; kv resonantly filtered
gky   init  0   ; An integration variable
gkdt  init  .004 ; Integration step, determines resonant frequency
gkd   init  .15  ; Damping, determines decay time


      instr 1  ; Collect midi data
gioct octmidi
gioct =     gioct + 1 ; increase upper range
gkdt  pchbend .2
gkdt  =     ( gkdt < 0 ? 0 : gkdt ) ; freeze the lower range
gkvol chpress 20000
gkvol =     ( gkvol<2000 ? 0 : gkvol )
      endin


      instr 100  ; Control processing and synthesiser
                 ; -separate from instr 1 to ensure continuity of output
gkv   port  gkvol, 0.01, 0

gkx   =     gkx + gky * gkdt; A driven, simple-harmonic-oscillator
gky   =     gky + (-gkx - gkd*gky + gkv ) * gkdt ; gkv forces.


kcps  =     cpsoct(gioct + (gkv-gkx)/10000) ; Abrupt breath changes cause a transient
kcpsl =     ( kcps > 100 ? kcps : 100 )   ; ripple to occur on the output pitch.
a1    oscil      gkv, kcps, 1
      out   a1
ga1   =     ga1 + a1
      endin



      instr 98   ; Facility for delay processing
a1    delay ga1, 0.5
ga1   =     a1 / 2
      out   a1
```

```
     endin


     instr 99
a1   reverb     ga1, 1.5
     out   a1
ga1  =     0
     endin
```

birdy2.orc

```
;  birdy2.orc
;  As for birdy1.orc but with two sine oscillators driven with
;  different control processors but with the same control data.
;  The quiescent pitches of the two sections are a fifth apart.
;  Dylan Menzies-Gow, August 95

sr = 20000
kr = 5000
ksmps = 4
nchnls = 1

ga1   init  0
gkx1  init  4   ; kv resonantly filtered
gky1  init  0   ; Integration variable
gkd1  init  .15  ; Damping -> decay time

gkx2  init  4   ; The two sections have the same set of these parameters.
gky2  init  0   ; but gky2, below, has a different equation to gky1.
gkd2  init  .15



      instr 1  ; Collect midi data
gioct octmidi
gioct =     gioct + 1 ; increase upper range
gkdt  pchbend .2
gkdt  =     ( gkdt < 0 ? 0 : gkdt ) ; freeze the lower range
gkvol chpress 10000
gkvol =     ( gkvol<1000 ? 0 : gkvol )
      endin



      instr 100  ; Control processing and synthesiser

gkv   port  gkvol, 0.01, 0

gkx1  =     gkx1 + gky1 * gkdt
gky1  =     gky1 + (-gkx1 - gkd1*gky1 + gkv ) * gkdt ; kvol forces.

gkx2  =     gkx2 + gky2 * gkdt
gky2  =     gky2 + (-gkx2*2 - gkd2*gky2 + gkv ) * gkdt ; kv forces.

kcps1 =     cpsoct(gioct + (gkv-gkx1)/10000)
kcps2 =     cpsoct(gioct + (gkv-gkx2)/10000)

a1    oscil      gkv, kcps1, 1
a2    oscil gkv, kcps2, 1
      out   a1+a2
```

```
ga1    =     ga1 + a1
       endin



       instr 98
a1     delay ga1, 0.5
ga1    =     a1 / 2
       out   a1
       endin



       instr 99
a1     reverb     ga1, 1.5
       out   a1
ga1    =     0
       endin
```

birdy3.orc

```
;  birdy3.orc
;  A variant on birdy2.c inwhich the sections have the same quiescent
;  pitch, but the damping factors are different
;  Dylan Menzies-Gow, August 95.

sr = 20000
kr = 5000
ksmps = 4
nchnls = 1

ga1   init  0
gkx1  init  4   ; kv resonantly filtered
gky1  init  0   ; integration variable
gkd1  init  .15  ; damping -> decay time

gkx2  init  4   ; kv resonantly filtered
gky2  init  0   ; integration variable
gkd2  init  .6   ; damping -> decay time



      instr 1  ; collect midi data
gioct octmidi
gioct =     gioct + 1 ; increase upper range
gkdt  pchbend .2
gkdt  =     ( gkdt < 0 ? 0 : gkdt ) ; freeze the lower range
gkvol chpress 10000
gkvol =     ( gkvol<1000 ? 0 : gkvol )
      endin



      instr 100  ; control processing and synthesiser

gkv   port  gkvol, 0.01, 0

;simple harmonic oscillator
gkx1  =     gkx1 + gky1 * gkdt
gky1  =     gky1 + (-gkx1 - gkd1*gky1 + gkv ) * gkdt ; kvol forces.

gkx2  =     gkx2 + gky2 * gkdt * 0.5
gky2  =     gky2 + (-gkx2 - gkd2*gky2 + gkv ) * gkdt *0.5  ; kv forces.

kcps1 =     cpsoct(gioct + (gkv-gkx1)/10000)
kcps2 =     cpsoct(gioct + (gkv-gkx2)/10000)

a1    oscil      gkv, kcps1, 1
a2    oscil gkv, kcps2, 1
      out   a1+a2
```

```
ga1     =       ga1 + a1
        endin



        instr 98
a1      delay   ga1, 0.5
ga1     =       a1 / 2
        out     a1
        endin



        instr 99
a1      reverb          ga1, 1.5
        out     a1
ga1     =       0
        endin
```

whistle.orc

```
;  whistle.orc
;  A windcontroller instrument.
;  Filtered noise with added sine tones for harmonics.
;  Increased breath tightens the filter.
;  Pitch is affected by the breath as well as keys and reed controls
;  Dylan Menzies-Gow, August 1995


sr = 15000
kr = 1000
ksmps = 15
nchnls = 1

ga1   init  0

gkv   init  0     ; filtered volume control
gkb   init  .2    ; filtered pitch bend


      instr 1
kbend pchbend    0.4
gkb   =     gkb + (kbend-gkb)/50
icps  cpsmidi
irise =     60/icps ; limit rise time : trills
                 ; less for higher notes

kvol  chpress 10000

gkv   =     gkv + (kvol-gkv)/10000*icps ; smooth out MIDI resolution
                  ; make higher notes respond quicker

kcps  =     icps*(.8+gkb)*(1-10/gkv)

as    rand  gkv, .45
kbw   =     5000/gkv

; one reson for noisy whistle:
a1    reson as, kcps, kbw
a1    reson a1, kcps, kbw

km1   tablei     gkv/20+60, 12  ; alter response of overtones here.
km2   tablei     gkv/50+60, 11
km3   tablei     gkv/50+40, 11
km4   tablei     gkv/50+20, 11
km5   tablei     gkv/50+10, 11
km6   tablei     gkv/50+00, 11

a3    oscil km2, kcps*3, 1
a4    oscil km3, kcps*4, 1
```

```
a5    oscil km4, kcps*5, 1
a6    oscil km5, kcps*6, 1
a7    oscil km6, kcps*7, 1


a1    balance a1, as
a1    =     a1*km1 + (a3+a4+a5+a6+a7)*2000

a1    linenr     a1, irise, 0.1, irise ; fade out note to prevent glitches
      out   a1
ga1   =     ga1 + a1
      endin



      instr 98
a1    delay ga1, 0.5
ga1   =     a1 / 2
      out   a1
      endin


      instr 99
a1    reverb     ga1, 1.5
      out   a1
ga1   =     0
      endin

      instr 100
      endin
```

wave.orc

```
;  wave.orc
;  A windcontroller instrument based on wave shaping.
;  Mixes two elements with different response times.


sr = 15000
kr = 1000
ksmps = 15
nchnls = 1

ga1   init  0

gkv   init  0
gkv2  init  0
gkb   init  .2
;gkp  init  0
;gkp2 init  0

;  This collects midi data on channel 1
;  possibly several notes at once.
      instr 1

;kcps cpsmidib
;kcps init  icps


kbend pchbend    0.4
gkb   =     gkb + (kbend-gkb)/50
icps  cpsmidi

gkcps =     icps*(.8+gkb)/4 ; 2 octaves down


gkvol chpress 10000

;gkp  =     gkp + (kcps-gkp)/200
;gkp2 =     gkp2 + (kcps-gkp2)/500
;gkp  =     kcps

;kvfluct   randi (1-gkv/10000)*0.8, 10, .12
;kvfluct = kvfluct + 1

      endin


; This is SYNTH CENTRAL

      instr 100
```

```
kdv   =     (gkvol-gkv)/4000*gkcps
kdv   =     ( kdv>0 ? kdv : kdv*2 ) ; quicker decay than attack
gkv   =     gkv +  kdv; smooth out MIDI resolution
gkv2  =     gkv2 + (gkvol-gkv2)/10000*gkcps  ; make higher notes respond quicker

;a11  gbuzz gkv, gkcps, 10, 2, 0.1, 1 ; sliding door
;a11  gbuzz gkv, gkcps, 1, 4, 0.5, 1 ; woody up top, helicoptor down below
;a11  oscil gkv, gkcps, 1 ; simple, hollow, elecroacoustic

a11   gbuzz gkv, gkcps, 100, 1, .5, 1 ; Brassy.
a1    =     a11

a22   gbuzz gkv2, gkcps, 100, 1, .5, 1
a2    =     a22

koffset     randi .2, 3, .12 ; Some random variation.
a1    tablei     a1/2*(1+koffset), 5, 0, 4096 ; waveshaping
a2    tablei     a2/2, 6, 0, 4096
a1    =     (a1+a2) * 7000
a1    balance a1, a11


      out   a1
ga1   =     ga1 + a1
gkvol =     0
      endin
```

conga.orc

```
;   conga.orc
;   A keyboard conga instrument.
;   Dylan Menzies-Gow, August 95

sr = 32000
kr = 1000
ksmps = 32
nchnls = 1

ga1   init  0


      instr 1
idec1 =     0.00  ; normal rate of attenuation
idec2 =     0.02  ; extra rate of attenuation when touching the drum
khp   init  10000
klevel      init  1
ivel  veloc
imax  =     .3  ; max length of drum sound

koct  octmidib
kcps  cpsmidib

kdec2 aftouch 0.3  ; additional attenuation caused by pressure

;kv   =     kv + (kat-kv)/1000
kgate linenr     idec2+kdec2, .001, .001, .01
            ; generate a variable indicating a note-off
kdummy      linenr  1,0,imax,1
            ; extend instr life long enough for sound to complete

;khp  =     khp * (1-kgate) ; During note-on khp decays
klevel      =     klevel * (1-idec1-kgate)

;klevel     expseg  1, 1, .01

;a1   randi 200, kcps*(1+kat)*4
;a1   gbuzz 200, 400, 100, 1, .5, 1
;a1   oscil 200, 400, 1

a1    loscil     200, 400, 20, 200, 0,0,16000  ; 20 = "conga1.aiff"
a2    loscil     200, 400, 21, 200, 0,0,16000  ; 21 = "conga2.aiff"
a3    loscil  200, 400, 22, 200, 0,0,16000  ; 22 = "conga22.aiff"

kmix1 tablei     koct * 25, 30
a2    =     a3 * kmix1 + a2 * (1-kmix1)

kmix2 tablei     koct * 30, 30
a1    =     a2 * kmix2 + a1 * (1-kmix2)
```

```
;a1   reson a1, kcps, kcps

a1    =     a1 * klevel * ivel
      out   a1

ga1   =     ga1 + a1
      endin

      instr 98
a1    delay ga1, 0.5
ga1   =     a1 / 2
      out   a1
      endin


      instr 99
a1    reverb     ga1, 1.5
      out   a1
ga1   =     0
      endin

      instr 100
      endin
```

natural.orc

```
;   natural.orc
;   A poly aftertouch keyboard instrument
;   Filtering a natural sound to create pitch/harmony
;   Dylan Menzies-Gow, August 95

sr = 32000
kr = 4000
ksmps = 8
nchnls = 1


ga1   init  0
gkb   init  0


      instr 1

kbend pchbend    0.4
gkb   =     gkb + (kbend-gkb)/200
icps  cpsmidi
icps  =     icps*4  ; 2 octs up

gkcps =     icps*(.8+gkb)


kchpr aftouch .002  ; for EPS.
;kchpr      chpress .001    ; for WX7


a1    reson ga1, gkcps, gkcps/200
amix  =     a1*kchpr+ga1
            ; mix in original sound to maintain power across the spectrum
ga1   balance    amix, ga1
      endin



      instr 100

      out   ga1
ga1,a2      soundin "stream.aiff"
; ga1 in    ; 'in' can be used to read sound directly from the
                 ; sound port if implemented.
      endin
```

delay4.c

```
/*****************************************************************************
******************************************************************************

     Description:  Tool for balancing a 4 speaker array by ear.
                   Quad echo, using one delay line with taps,
                   mono input from microphone.

     Author:       Dylan Menzies-Gow, JOShUA Interactive.

     Date:      17/5/95

     Comments:  Have fun.

******************************************************************************
*****************************************************************************/


#include <audio.h>
#define SAMPLE_RATE 44100
#define DELAY 40000

void main(void)
{

ALport port_address_out;   /* Pointer audio port for SGI Indigo */
ALport port_address_in;   /* Pointer audio port for SGI Indigo */
ALconfig config_in, config_out;

long buf[] = {  AL_CHANNEL_MODE, AL_4CHANNEL,
           AL_INPUT_SOURCE, AL_INPUT_MIC,
           AL_INPUT_RATE, SAMPLE_RATE,
           AL_OUTPUT_RATE, AL_RATE_INPUTRATE,
           AL_MIC_MODE, AL_MONO,
           AL_SPEAKER_MUTE_CTL, AL_SPEAKER_MUTE_ON,
           AL_LEFT_INPUT_ATTEN, 0,
           AL_RIGHT_INPUT_ATTEN, 0 };

short delay_line[DELAY]; /* = (short *)calloc( DELAY*4, sizeof(short)); */
long count1, count2, count3, count4;
                  short samples[4], S;

config_in=ALnewconfig();
config_out=ALnewconfig();
ALsetwidth(config_in, AL_SAMPLE_16);
ALsetwidth(config_out, AL_SAMPLE_16);
ALsetchannels(config_in, 2);
ALsetchannels(config_out, 4);
ALsetparams(AL_DEFAULT_DEVICE, buf, 6);
ALsetqueuesize(config_in, 4000);
ALsetqueuesize(config_out, 4000);
port_address_in = ALopenport("input","r",config_in);
port_address_out = ALopenport("output","w",config_out);

for(count1=0; count1<DELAY; count1++)
  delay_line[count1] = 0;

count1=0;
count2=DELAY/4;
count3=DELAY/2;
count4=DELAY*3/4;
while(1)
{

ALreadsamps(port_address_in, samples, 2);
delay_line[count1++] = (samples[0] += delay_line[count1]>>1);
samples[1] = delay_line[count2++];
samples[2] = delay_line[count3++];
samples[3] = delay_line[count4++];

if (count1>=DELAY) count1=0;
if (count2>=DELAY) count2=0;
if (count3>=DELAY) count3=0;
if (count4>=DELAY) count4=0;

ALwritesamps(port_address_out, samples, 4);

}

}
```

solo.c

```
/*****************************************************************************
*****************************************************************************

     An implementation of the delay line for SOLO by Stockhausen.
     Feedback, microphone levels and timing are all sequenced.
     Performance output on  channels 1,2. Click track on channel 3,4.
     Set number of clicks per period, for each section in char clicks[].

     (c) Dylan Menzies-Gow, June 95.

*****************************************************************************
*****************************************************************************/


#include <audio.h>
#include <stdio.h>
#define SAMPLE_RATE 44100
#define SPEED 1
#define MAX_DELAY 2231460
#define SECTIONS 6

void main(void)
{

ALport port_address_out;   /* Pointer audio port for SGI Indigo */
ALport port_address_in;   /* Pointer audio port for SGI Indigo */
ALconfig config; /* Temporary varialbe to set SGI audio parameters */
long buf[] = {  AL_CHANNEL_MODE, AL_4CHANNEL,
          AL_INPUT_SOURCE, AL_INPUT_MIC,
          AL_INPUT_RATE, SAMPLE_RATE,
          AL_OUTPUT_RATE, AL_RATE_INPUTRATE,
          AL_MIC_MODE, AL_MONO };

short delay_line1[MAX_DELAY];
short delay_line2[MAX_DELAY];

char mic_level1_table[] = { -1,  /* extra beat */
                3, -1, -1, 2, -1, -1, 0, 2, 3, -1, -1,
                4, 2, 3, 0, 3, 2, 3, -1,
                4, -1, -1, 3, -1, -1, -1,
                5, -1, -1, 4, -1, -1,
                -1, 3, -1, 2, -1, 3, -1, -1, -1,
                2, -1, 2, -1, -1, 1 -1, 2, -1, -1
                };

char mic_level2_table[] = { -1,
                -1, 2, 1, -1, 0,1, -1, -1, 1, 2, 1,
                3, 0, 2, 2, 2, 0, 0, -1,
                1, 0, 2, -1, 3, 2, 1,
                -1, 4, 3, -1, -1, -1,
                2, 1, 1, 1, 1, -1, 1, 2, -1,
                -1, 2, -1, -1, 3, -1, 2, -1, 2, -1
                };

char feedback_level1_table[] = { -1,
                -1, 2, 0, 1, 1, -1, -1, 1, 1, 0, 1,
                -1, 1, 1, 0, 0, 1, 2, 1,
                -1, 3, -1, -1, 2, 0, 2,
                -1, 3, -1, -1, 2, -1,
                -1, -1, 1, 1, -1, -1, 1, 2, -1,
                -1, 0, 1, 1, 0, 1, -1, -1, 2, -1
                };

char feedback_level2_table[] = { -1,
                -1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1,
                -1, 0, 1, 1, 1, 0, 1, -1,
                -1, 0, 1, -1, -1, 1, 1,
                -1, -1, 2, -1, -1, -1,
                -1, 0, 0, 0, 1, 0, 0, 1, 0,
                -1, -1, 1, -1, -1, -1, -1, 1, 2, -1
                };
char clicks[] = { 4, 4, 4, 4, 4, 4 };

char periods[] = { 12, 8, 7, 6, 9, 10 }; /* NB extra period at start for sync */

float delay_times[] = { 6, 14.2, 19, 25.3, 10.6, 8 };

long delay_sizes[6];

long click_counts[6];

short del_samp[4], mic_samp[4];

long count;
long delay_pos, delay_size, click_pos, click_count;
char mic_level1, mic_level2, feedback_level1, feedback_level2;
char period, section, period_total;

     /* Open up SGI port for audio output */
config=ALnewconfig();/* Default structure for audio configuration */
ALsetwidth(config, AL_SAMPLE_16);          /* 16-bit samples */
ALsetchannels(config, 4);
ALsetparams(AL_DEFAULT_DEVICE, buf, 10);
ALsetqueuesize(config, 10000);
port_address_in = ALopenport("input","r",config);  /* Open SGI audio port */
port_address_out = ALopenport("output","w",config);  /* Open SGI audio port */

for(count=0; count<MAX_DELAY; count++)
{
  delay_line1[count] = 0;
  delay_line2[count] = 0;
```

```
};

for(section=0; section<6; section++)
{
  delay_sizes[section] = delay_times[section]*SAMPLE_RATE/SPEED;
  click_counts[section] = delay_sizes[section] / clicks[section];
}

section=0;
period_total=0;
do  /* new section */
{
     period = 0;
     delay_size = delay_sizes[section];
     click_count = click_counts[section];
/*
     printf("\nnew_section\n");
     printf("%d\n", delay_size);
*/

     do  /* new period */
     {
/*
          printf("new_period\n");
*/
          delay_pos = 0;
          click_pos = 0;
          mic_level1 = mic_level1_table[period_total];
          mic_level2 = mic_level2_table[period_total];
          feedback_level1 = feedback_level1_table[period_total];
          feedback_level2 = feedback_level2_table[period_total];
/*
printf("mic_level1 = %d\n",mic_level1);
*/

          do  /* new sample */
          {
               if (click_pos == 0)
               {
                  /* Make a click on channels 3,4 */

                  del_samp[0] = del_samp[1]
                  = ( delay_pos == 0 ? 0x8000 : 0x4000 );

                  ALwritesamps(port_address_out, del_samp, 4);
                  ALwritesamps(port_address_out, del_samp, 4);
                  ALwritesamps(port_address_out, del_samp, 4);

                  /* higher pitch on leading beat.. */
                  if (delay_pos != 0)
                  {
                  ALwritesamps(port_address_out, del_samp, 4);
```

```
                    ALwritesamps(port_address_out, del_samp, 4);
                    ALwritesamps(port_address_out, del_samp, 4);
                    };
                    del_samp[0] = del_samp[1] = 0;
                    ALwritesamps(port_address_out, del_samp, 4);
                    click_pos = click_count;
                  }
                  del_samp[2] = delay_line1[delay_pos];
                  del_samp[3] = delay_line2[delay_pos];
                  ALwritesamps(port_address_out, del_samp, 4);

                  while (ALgetfilled(port_address_in)==0);

                  ALreadsamps(port_address_in, mic_samp, 4);

                  delay_line1[delay_pos] =
              ((feedback_level1== -1) ? 0 : (del_samp[2]>>feedback_level1))
          +   ((mic_level1== -1) ?  0 : mic_samp[2]>>mic_level1);

                  delay_line2[delay_pos] =
              ((feedback_level2== -1) ? 0 : (del_samp[3]>>feedback_level2))
           +  ((mic_level2== -1) ?  0 : mic_samp[3]>>mic_level2);

              delay_pos++;
              click_pos--;
              }
              while(delay_pos < delay_size);
          period++;
          period_total++;
          }
          while(period < periods[section]);
      section++;
      }
      while(section < SECTIONS);

      }
```

The Duck Family Tree

[From Per Starback:]

How the ducks are related is an old much-debated topic in duckdom, and it has been discussed at some length in the disney-comics list as well. This file does not contain The Answers to those questions, but just information on one interesting source of such information, namely a duck family tree that Carl Barks made in the early fifties for his own reference. It is published in Carl Barks Library, Set VI, p. 476, and I won't try to redraw it here, but the information in it is:

Old "Scotty" McDuck had the following children:
  Matilda McDuck who married Goosetave Gander,
  Scrooge McDuck,
  Hortense McDuck.

Grandma Duck had the following children:
  Quackmore Duck,
  Daphne, who married Luke the Goose.

Hortense McDuck and Quackmore Duck married and had Thelma Duck (the mother of Huey, Dewey and Louie) and Donald Duck.

Luke the Goose and Daphne had one son, Gladstone, who was orphaned when Daphne and Luke overate at a free-lunch picnic. Gladstone was then adopted by Matilda McDuck and Goosetave Gander!

Gus Goose was a nephew of Luke the Goose "making him a very distant `cousin' of Donald".

BYE BYE

An Investigation Into The Design Of Musical Performance Instruments

Dylan Menzies-Gow

Departments of Electronics and Music,
University of York, United Kingdom.

Abstract

The Yamaha VL1 has attracted much interest as the first generally available synthesiser to emulate the subtle dynamic response of acoustic instruments, and yet not be constrained to copy these instruments wholesale. While the VL1 is a powerful, state of the art machine, the possibility is explored here of enriching the control dynamics side of existing MIDI equipment by the computer processing of MIDI control signals with an Atari ST. The WX7 windcontroller and the polyphonic aftertouch keyboard are considered as controlling devices. This leads onto more general considerations of musical performance instruments. Csound running in real time on an SGI Indy equipped with a MIDI interface is used to explore techniques not accessible on MIDI synthesisers. Several useful examples are presented, and some ideas for future work which the author feels encouraged to undertake

1.Introduction

Many current electronic synthesisers being marketed as performance keyboards lack the control possibilities seen in acoustic instruments, and are often very similar to one another. The key to there usefulness is often just the variety and novelty of the samples they contain.

1.1 The use of effects processors to augment instruments

Effects processors serve to enrich the response of the instrument as well as changing its sound. A good example is delay: A complex, interesting and slightly unpredictable sound can be generated with a few notes. The control of the 'delay-instrument' is more complex, and interesting than without delay: The output depends significantly on the player's input sometime before. It is natural therefore to consider the general class of instruments in which the sound output at a given time depends on the history of input by the player. This shall be the main consideration in the designs described later. In retrospect, acoustic instruments exhibit 'temporal complexity' in the control of their sound, which certainly contributes to their musical value.

Unprocessed sounds from sample-playback keyboards have a very static quality: On repetition, exactly the same sound is output. Apply an effects processor and this is not true as many effects algorithms are time dependent and/or highly sensitive to initial conditions. The control may be uninteresting note on/off but the sound in itself is interesting. This changing quality is very apparent in real instruments like the piano, and is an important design consideration later. The question arises 'how far can temporal complexity alone be musically useful without using 'changing sounds?'

1.2 The Yamaha VL1

The Yamaha VL1 is the first generally available synthesiser to emulate the rich response of acoustic instruments, and the main inspiration for the investigation. It is one of the few today to take an integrated approach to being a musical performance instrument rather than a synthesiser with a keyboard attached. A synthesiser may be capable of producing sounds similar to the VL1 with much effort, but a performer can only become involved and produce good music if the whole instrument is good: the physical side and the response as well as the synthesiser.

The Yamaha WX7/WX11 windcontrollers are relatively simple and physically unappealing by comparison with a saxophone, yet they can be used to stunning effect with the VL1. This demonstrates the importance of the 'response feel' or temporal-complexity of the instrument over the 'physical feel', and hence provides some validation for the use of the WX7 in the following designs. In the VL1 the synthesis is tightly bound to the control response, because it is based on a waveguide model of real instruments: While the VL1 is admired for the 'new' instruments which can be constructed, its response characteristics are constrained to the waveguide model.

2. Original instrument designs

The designs have been constrained to specific hardware and software configurations as follows. Considerations of physical design have been left, although it is realised that these are very important.

1. MIDI equipment with MIDI processing by an Atari ST
The system is illustrated in figure 1. The instrument has been split into a control processing stage driving a synthesiser. The principal aim is to enrich the control reponse of MIDI instruments and consider the wider possibilties of instrument design created.

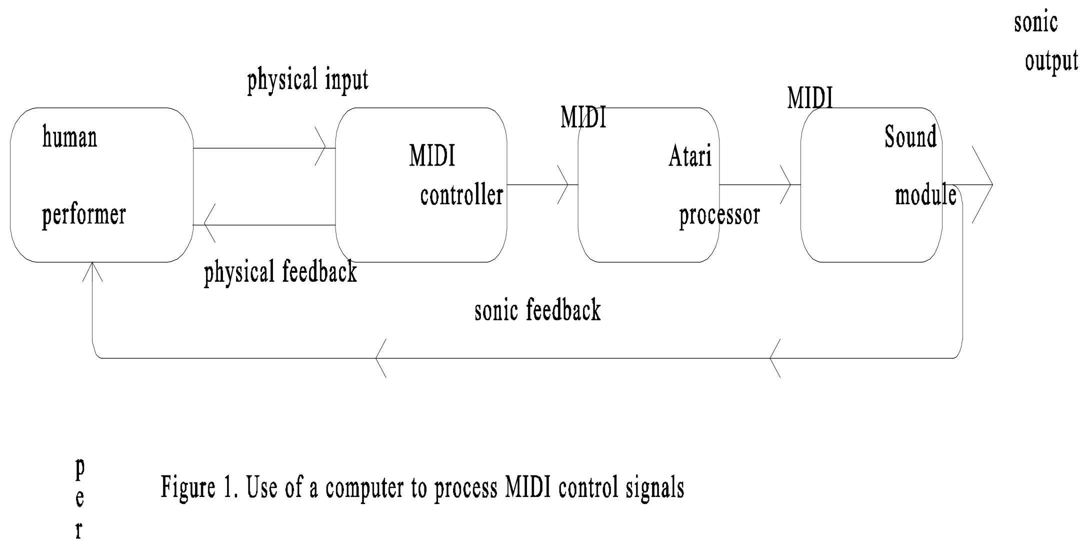


Figure 1. Use of a computer to process MIDI control signals

2. MIDI controllers + real time Csound running on an SGI Indy with MIDI interface.
Csound has been used for some time strictly as a compositional environment, despite its use of the term 'instrument'. The language is convenient for trying out synthesis methods, but awkward for implementing control processing. The idea is to try out designs impossible with MIDI sound modules.

<u>2.1 Summary of designs</u>

<u>MIDI equipment with an Atari ST and Lattice C</u>
'reed' and 'breath' refer to the controls of the Yamaha WX7.
The program names are in bold type.

1. Arpeggiation instrument using the WX7 / Korg T3, **weird.c**
The player controls the arpeggiation rate and speed using the key and reed controlss. Breath controls volume. 'Normal' playing can be executed by opening the reed.

2. Granular synthesis instrument using the WX7 / SY55, **granny.c**
Two sounds are generated: A low sound resonds to breath very slowly, and remains constant in pitch. It provides a kind of background aura. The sound is generated by triggering short notes with a poisson process. A high sound responds quickly to breath, but leaves a trail of notes as the keys change. The longer a note is held the richer the sound, because notes overlap.

3. Additive synthesis instrument using the EPS / K1, **additive.c**
The player dynamically controls the harmonic content of a note by hitting a control key for each harmonic.

<u>MIDI controllers with an SGI Indy running Csound</u>
1. A bird-like instrument using the WX7, **birdy1.c, birdy2.c, birdy3.c**
Sudden changes in breath cause a 'ripple' on the pitch output, due to the application of a resonant filter. Closing the reed raises the resonant frequency from 0 to just sub audio. With an open reed the pitch can be controlled by breath. With a closed reed the pitch can only be controlled by the keys. The result is a human bird song generator.

2. A whistle-like instrument using the WX7
A resonant filter is applied to white noise. Closing the reed tightens the filter and produces a tone. The pitch falls off at low breath, and consecutive notes overlap by increading amounts down the scale. This results in a very realistic whistle instrument.

3. A brass-like instrument using the WX7
The breath controls volume and timbre off the sound, which becomes brighter at higher volumes. The reed effects pitch. The lower keys have sluggish response compared to the upper keys. There is a slight 'attack' when switch between notes. An interesting effect occurs when the breath increases sharply: Instead of perceiving a steady change in timbre, the sound takes on a kind of steady
'transition timbre'.

4. A conga drum using a keyboard, **conga.orc**

This is not an attempt to make a new instrument, but a way of showing how a keyboard can be used in an unusual way. On the left of the keyboard a low conga sound plays, on the right a high one. In between there is a gradual cross over in sound. Hitting a key and releasing a key quickly results in a resonant conga sound. While the key is held the sound becomes progressively damped. If using aftertouch, pressing on the key further increases the damping.

5. A filter bank using a poly aftertouch keyboard, **natural.orc**

This is more of a concept-instrument. It is inspired by the filtered harmonies of a waterfall in 'Riverrun' by Barry Truax. With no keys pressed the player hears a natural sound; a stream running or wind blowing. Pressing a key down progressively harder, results in a whistling pitch rising from the background sound. Subtle control is required so that the sound does not become too prominent.

3. Conclusion

By a process of design by experimentation the investigation has demonstrated the validity of some of the initial hopes: Dynamic control opens the door to many possibilities. The dynamics considered in the project designs were very simple, the most complex being the driven simple-harmonic-oscillator. It is possible more-complex dynamics could be of value, in particular mathematically-chaotic dynamics. Note that the total dynamic system includes the performer, who is difficult to quantify. This is why a simple driven oscillator works well.

'Changing sounds' can be generated in simple ways such as in granny.c The answer to the question posed in the introduction 'Can temporal complexity of control, used with a static sound make a successful instrument?' is decidedly in the affirmative, with results from birdy1.c . Regarding MIDI equipment: It is less than transparent to use in any but the most straight forward way, and highly machine dependent. However, with perseverance MIDI synthesisers can be a valuable tool in performance instrument design. The situation may become more favourable if the new 'Zippy' standard becomes widely adopted.

Overall, the emphasis has been on combining many different elements of design to produce a successful instrument rather than pinning hopes on a single grand idea. In this sense, performance instrument design resembles 'composition in possibilities'.

References

Baker GL, Gollub JP
1990 CUP
"Chaotic Dynamics"

Donnington R
1970 Methuen
"The Instruments Of Music"

Experimental Musical Instruments
gopher://echonyc.com/11/Music/MO/EMI

Jenkins JC
1983 Royal Scottish Museum
"Survey Of Non-European Instruments"

Krishnaswarmy S
1971 Crescendo
"Musical Instruments Of India"

Remmant M
1978 Batsford
"Musical Instruments Of The West"

Sawyer D
1977 CUP
"Making Unorthrodox Musical Instruments"

Smith JO
1992 Computer Music Journal Vol 16 No 4
"Physical Modelling Using Digital Waveguides"