Inspired by Computer Technology

Miyuki ITO

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

Columbia University
2003

© 2003

Miyuki Ito
All Rights Reserved

ABSTRACT

Inspired by Computer Technology

Miyuki Ito

This dissertation analyzes the piece, *Réminicence d'un ancien esprit* both from the aesthetic point of view and from the technical point of view. It is comprised of four sections. The first section introduces a musical idea which is notated on the score. It also discusses some influence from Japanese aesthetics and traditional culture. The second part presents real time signal processings, using Max/Msp. The third section presents a process of how soundfiles are transformed by computer. The concluding section discusses a future possibility of how to create music with technology.

Table of Contents

LISTS OF MUSIC EXAMPLES AND TECHNICAL EXAMPLES iii - vi

ACKNOWLEDGEMENTS vii

SECTION 1. "MUSICAL IDEA"

Presentation of the Work 1

Text and the Japanese Language 2

Extended Vocal Techniques 5

Vocal Register 10

Percussion Usage 10

Japanese Sensibility 11

SECTION 2. "REAL TIME SIGNAL PROCESSING"

Technical Schemes 15

Max/Msp 19

SECTION 3. "ELECTRONICS"

Configuration 57

Pre-recorded Materials 60

Electnonics Contents 63

SECTION 4. "CONCLUSION" 82

Appendix

1. Software used for “Réminiscence d’un ancien esprit” 84

2. Score of “Réminiscence d’un ancien esprit” 86

Lists of musical exaples and technical examples

Example 1: intonation diagram 3

Example 2: section1, between events 16 and 17 6

Example 3a: section1, event 6 6

Exmaple 3b: section2, between events 24 and 26 6

Example 4: section1, event 21 7

Example 4: continued: section2, event 35 8

Example 5: section2, events 27,28 and 29 8

Example 6: section2, events 48 and 49 8

Example 7: section2, page 6 in the score 9

TECHNICAL SCHEME 1: speaker placement 17

TECHNICAL SCHEME 2 18

Example 9: *p INITS* 20

Example 8: Concertpatch 21

Example 10: *p pedal* 20

Example 11: *p score* 22

Example 12: table of the effects and pcontrol 23

Example 13: *p osc* 24

Example14: DSP status 24

Example 15: *p effect* 26

Example 16: event 4, event 4 delayed 1100 and event 6 26

Example 17: *p event* 1~79 27 - 32

Example 16: *p reverbs1 & 2* 33

Example 17a: *p transp/delay* 34

Example 17b 35

Example 17c: *p midi--->freq* 35

Example 18a: filtered noise ring modulation 36

Example 18b: three different filtered noises 37

Example 19: section 2, event 47 and section 3, event 58 37

Example 20: section3, event 59 39

Example 21a: *p vocoder* 39

Example 21b: *p bank* 39

Example 21c: *p env* 40

Example 21d: *p bank2* at event 6 40

Example 21d: *p bank2* 40

Example 22a: *p straing* 42

Example 22b 43

Example 22c: *CordeParam* 43

Example 22d: *MaterialPara* 43

Example 22e: section1, event 21 44

Example 23: *p filter* 44

Example 24a: *p fft-xover* 45

Example 24b: *pfft~ xover~* 45

Example 25a: section2, events 32 & 51 and sefction3, event 53 46

Example 25b: *p harm2* 47

Example 26a: *p xsynthx* 48

Example 26b: matrix for section3, event 53 48

Example 26c: *p cross-pup* 48

Example 27a: *p filtrage* 49

Example 27b: *buffer~* 49

Example 27c: *pfft~ filtragefft~* 50

Example 27d: filt1 50

Example 27d: filt2 and filt3 50

Example 28: *p pan1,2 control* 51

Example 29: 6 speakers setup 52

Example 30: summary for each event 53 - 56

Example 31: *p soundsplayer* 57

Example 32: section1, event 2 58

Example 33 59

Example 34: SF1 63

Example 35a: text1 64

Example 35b: Chaos/OpenMusic 65

Example 35c: bpf window 65

Example 35c: continued: pitch material 66

Example 35d: bpf window 66

Example 35e: bpf window and pitch material used for the second half of the contour of the text1 66

Example 35f: *samplor~* with Max/Msp 67

Example 35g: SF2 67

Example 36: SF4 68

Example 37: SF5 and SF6 69

Example 38a: formant filter parameters - [i] [u] [i] (SF7) 69

Example 38b: SF7 70

Example 39: SF8 and SF9 70

Example 40: Table of formant values for vowel sounds 71 - 72

Example 41: SF10 73

Example 42: SF12 74

Example 43: SF13 74

Example 44a: SFs21 and 22 76

Example 44b: Modalys 76 - 78

Example 45a: inscription1/Melodyne 79

Example 45b: SF25, text1(inscription1) 79

Example 46: SF27, text2 (inscription2) 79

Example 47: SF28, text3 (inscription3) 80

Example 48: SF24, jtb1 80

Example 48: SF26, jtb2 80

Example 49:

Inscription in Japanese, used in the electronics, section3 81

Acknowledgements

I wish to express my sincere gratitude to my professors, Jonathan Kramer, Tristan Murail, and technical assistants at IRCAM, Jean Lochard, Benjamin Thigpen and Sébastien Naves for their advice and encouragement throughout this project.

I would like to express my special appreciation to IRCAM without which I would not have come to the completion of the work, "Réminiscence d'un ancien esprit". I believe that technology helps our creativity.

SECTION 1

"MUSICAL IDEA"

Presentation of the Work

"Réminiscence d'un ancien esprit" was composed at IRCAM (Institute for research and coordination of acoustics / music, Pompidou Center, Paris, France) in 2002. It is scored for soprano solo and electronics with a small percussion set-up including 3 stones, rain stick, 2 Japanese temple bowls, and 3 Japanese wind bells. It was premiered at IRCAM on October 17th, 2002 during the Résonances Festival. It was also the first interactive piece of mine with computer.

The initial idea of this piece came from my trip to Nara, Japan. After visiting the Yakushiji-temple in Nara, I was inspired by two inscribed stones called the Buddha's Footprint Stones, originally called *Bussokuseki* in Japanese. These stones receive little attention compared to the three famous bronze Buddhist statues also at the temple. The Footprint Stones, however, are registered as a national treasure of Japan. They were completed in the eighth century. Inscribed on one of the stones are twenty anonymous poems in the Footprint Stone style of 5-7-7-7 syllabic form, three of which I chose to use as the text for my music. On the other stone are inscribed a footprint and three inscriptions.

For the electronic part of the last section, I used the inscription which

speaks of the origins of the footprints and how they can help people to go heaven. I created a sound like a Buddhist priest chanting using a modified soprano singer's voice and artificial sounds created by a formant filter. Using eight speakers I attempted to recreate the atmosphere of a Buddhist temple. I will discuss this in detail, later.

For this piece, I explored different features of the Japanese language, including phonetics, vowels, formants, and pitch accents. I am particularly interested in the way sounds are produced in Japanese to create a unique kind of music.

Text and the Japanese Language

As mentioned above, the three texts were chosen from the Buddha's Footprint Stone Poems. Here is the original text in Japanese and English translation.

1.

miato tukuru	The ringing of the rock
ishino hibikiha	on which the holy footprints are cut
ameni itari	reaches to heaven
tsuchisahe yusure	and even earth resounds:
chichi hahaga tameni	For father and mother,
morobitono tameni	for all men

2.

misoji amari	The footprints trod by
futatsuno katachi	him who is complete with
yasokusato	the thirty and
sodareru hitono	two marks and
fumishi ato dokoro	the eighty signs:
marenimo arukamo	How rare indeed there are.

3.

ohmiatowo	For him who comes to see
minikuru hitono	the divine footprints,
inishikata	even the sins of a thousand worlds,
chiyono tsumisahe	ago in the past
horobutozo iu	disappear - so it is said:
nozokutoao kiku	He takes them away - thus have we heard

* English translation taken from Miller, Roy Andrew. *The Foot Prints of the Buddha* (American Oriental Series vol.58)

Example 1: intonation diagram

(1) mi/a to tu/ku\ru
i/shi no hi/bi ki\ha
a\me ni i/ta\ri
tsu/chi sa he yu/su re
chi chi ha ha ga ta/me\ni
mo/ro bi to no ta/me\ni

(2) mi/so ji a\ma ri
fu/ta tsu no ka/ta chi
ya/so ku sa to
so/da re ru hi/to no
fu\mi shi a to do ko ro
ma/re ni\mo a\ru ka mo

(3) oh mi\a to wo
mi\ni ku\ru hi/to no
i/ni shi ka ta
chi\yo no tsu\mi sa he
ho/ro bu to zo i/u
no/zo\ku to zo ki/ku

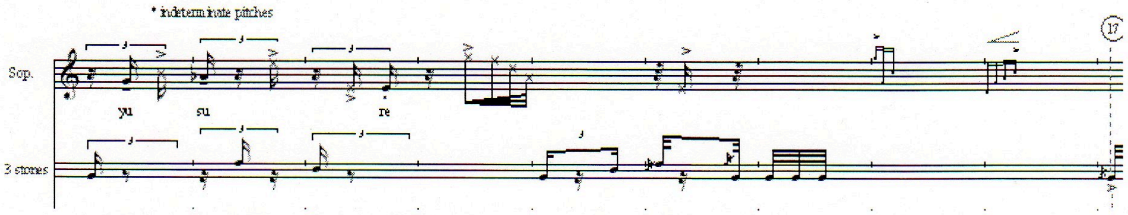
Since Japanese has pitch accents, if you change a pitch, a meaning might be changed. Even though there is some flexibility, I believe we should generally follow the rule of intonation because I respect the language itself. Since I am from Nagoya, where the dialect spoken is different from that in Tokyo, I checked the intonation of Japanese accent in a dictionary to make sure I was following the accents of standard Japanese. I always make an intonation diagram as in Example 1, to make a plan for my music. However, I do not have a strict system for pitch setting for the text. I am trying to find a natural way to create a melodic line which follows the text.

Another important feature is a simple consonant-vowel sound structure. There are sixty-five basic tones in Japanese, consisting of a consonant plus vowel structure. Vowels drawn from only the five sounds of a, i, u, e, o. Because of this structure, only vowel sounds can be extended. In a way, Japanese creates some kinds of melody with short or extended vowel sounds, with meaning-significant differences between them, and a space called "ma" between words. This is unlike English which creates rhythm using stress accent. In traditional Japanese theater or music, these features are used very well in a sophisticated vocal technique. Especially, melismas for vowel sounds are an important technique. Otherwise, due to the simplicity of the sound structure, it might be uninteresting. I develop this vowel-oriented structure in my music.

ExtendedVocalTechniques

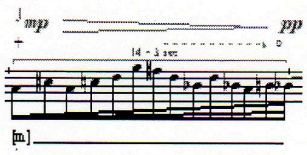
Tongue clicks () are used as a special effect. This sound merges with that of the stones being struck together in the first section. The clicks are notated as indeterminate pitches. Reverberation is added to the first tongue click in the first section in real time, with Max/Msp, providing a very metallic effect. It is reminiscent of the sound of stones which has two effects using Max/Msp filtering and *sträng* objects (see page 30) in real time at event 21(see Example 22e) because both effects make the sound of stones metallic. However, this effect sounds very soft and enables us to

listen without an amplifier, particularly with percussion and electronic sounds. The other example occurs after event 16. Here, the sounds of voice, tongue click and stones are merged with each other to create an interesting timbre and rhythm (see Example 2).

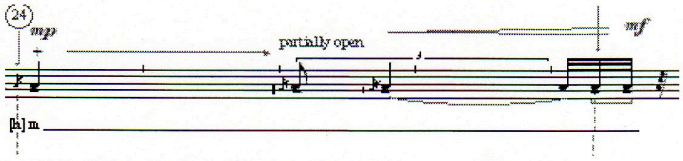


Example 2: section1, between events 16 and 17

Three mouth positions are specified such as open mouth (◦), partially open mouth, and closed mouth (+). In the beginning of the first and second sections, the position changes gradually from closed to open mouth. This gives subtle timbre changes, as Examples 3a and b shows.

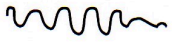


Example 3a: section1, event 6

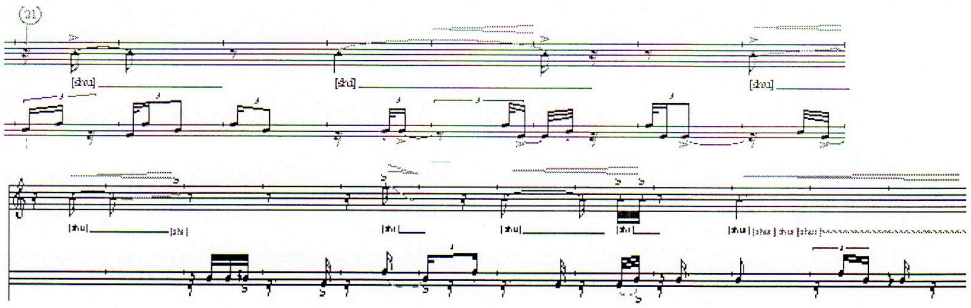


Exmaple 3b: section2, between events 24 and 26

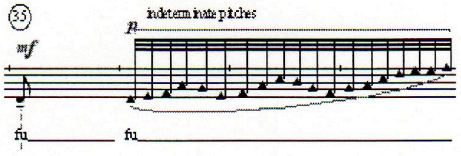
As the pitch changes, I use the three gestures of tremolo, trill and vibrato. In fact, tremolo is repeated pitches. The control of tremolo depends on the

register and on the singer. It seems difficult to control the exact number of tremolos in the high register for a soprano singer. I indicated an approximate number of tremolos in the high register after I discussed it with my singer. The trills also depend on the register and on the singer. I use an interval up to minor third. Its effect is just like instrumental trills. I use this symbol  for vibrato. It signifies the speed and amplitude of vibrato. I also often use this gesture for instruments in order to add subtle timbre.

Voiceless sounds ( ) are used without pitch, following Japanese pronunciation. I also notate this effect with indeterminate pitches (see Example 4). It causes a breathy sound and works well with a real-time treatment on computer.

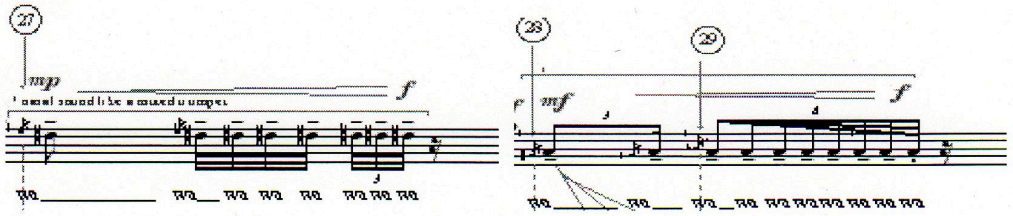


Example 4: section1, event 21

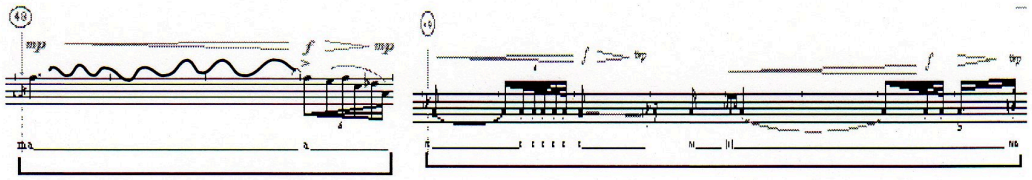


Example 4: continued: section2, event 35

A nasal sound, like a trumpet mute, is indicated in the score by the phonemes “wa” and “uo”. This effect is used especially in the second section (see Example 5). I also add a filtering effect in real time to reinforce this effect. Furthermore, at events 48 and 49, only the filtering effect is used in real time to produce a sound like a muted trumpet (see Example 6). I will discuss this in Section 3.



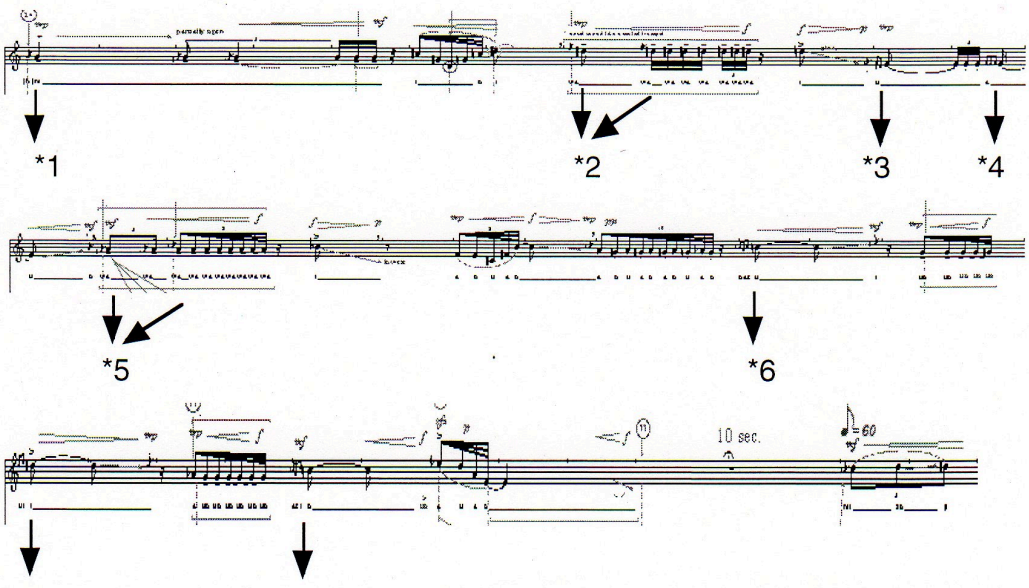
Example 5: section2, events 27,28 and 29



Example 6: section2, eventsß 48 and 49

In addition, letters with parenthesis represent phonemes, except on page 6 (see Example 7). Until the poem starts at the end of page 6, only phonemes are used. They are vowels which come from a consonant-

vowel sound structure. Consonants are excluded from the text on page 6. For example, mi-so-ji in the beginning of the second poem is transformed to i-o-i. Without consonants, the whole text of the second poem becomes i-o-i a-a-i, u-a-u-o a-a-i, a-o-u-a-o, o-a-e-u i-o-o, u-i-i a-o-o-o-o, a-e-i-o a-u-a-o (Example 7). When the same vowels continue, I change the timbre to sound like a muted trumpet such as wa-wa-wa-wa..., and uo-uo-uo-uo....which I mentioned in Example 5. Especially on this page, I tried subtle color changes with vowels, shown on the below.

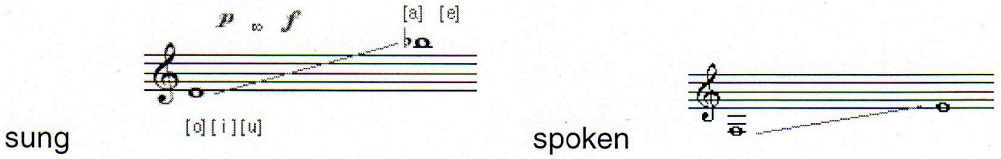


Example 7: section2, page 6 in the score

Grace notes should be sung on the beat. There are three kinds. The first and most often used grace note has the same word or vowel as the next pitch. It is a gesture similar to a melisma as a subtle timbre change. It is

shown at *2, *3, *4, and *5 in Example 7. The other two have different phonemes from the next pitch. One has a delicate accent effect, as at *1 in Example 7. The other one is a quick phoneme change, as at *6, *7 and *8 in Example 7. In fact, the grace note, one of the important gestures in my music, is also important in traditional Japanese music.

Vocal Register



This is the vocal register of my singer. As a soprano, she has quite a wide range. She could sing from middle D to Bb with a dynamic between p and f. Her high register can be produced with phonemes [a] and [e]. The low register can be with [o], [i] and [u].

Percussion Usage

In section 1, I use three stones (high, middle, low) and one stone to strike and a rain stick. I tried stones for my piece, "Towards the Universe" for soprano and piano, the third section of which is without piano. This effect

was successful in that piece. Because at that time, a singer just needed to play stones, it was possible for her to hold stones with her hands. However, since this time she needs to play a rain stick and press a finger switch, her hands should not be occupied holding stones. I fixed three stones on the table with a cushion, because leaving the stones directly on the table will make the sound muted and not good at all. Despite the cushion, it was difficult to find a nice sound. Because the text is obviously related to stones and I was curious about the combination of the sounds of stones and the tongue click, I use stones in section 1. In section 3, there are two Japanese temple bowls pitched approximately to C# and D and three Japanese wind bells (high, middle, low). Triangle beaters are used to rub the temple bowls and strike bells. A leather stick is used for a normal striking on the bowls from event 63. We needed to be careful with our use of the microphone for the percussion, because it easily detects small noises and causes feedback. I will talk about this technical point in the next section.

Japanese Sensibility

Traditional Japanese aesthetics is important in my music. Since I left Japan in 1995, I have become more deeply attached to Japanese culture. A recent main focus in my music is a gradual and subtle color change, using Japanese concepts of time.

Traditional Japanese aesthetics is deeply rooted in naturalism. The Japanese used to try to imitate and unify nature intuitively in all types of arts, including theater and music. In Japanese traditional paintings, these aesthetic values produce a randomly distorted formula. The result is a composition with artistically well-arranged blanks or spaces not however mathematically arranged in any precise order. Even though the structure of many of these works is not obviously symmetrical and balanced, recent analysis using fractal geometry has revealed structures closely related to those found in nature. Because I have always been very interested in this cultural aesthetics, this discovery made a strong impression on me. I tried this idea in my pieces, "Fading Beauty" for six players (2000) and "Of the Flowing River the Flood Ever Changeth..." for a large orchestra (2002), using fractal processing by computer. I also use chaos processing in the electronic part of the piece "Réminiscence d'un ancient esprit", which I explain later in Section 3.

Silence is also important in my recent pieces. My idea of silence comes from the reverberation of the Japanese temple bell and also from Japanese aesthetics called 'ma'. The Japanese temple bell is rung the second time only after the reverberation is fading out, unlike western bells which are generally rung regularly without waiting for the reverberation. The Japanese temple bell in particular has a rich resonance because of complicated inharmonic spectra. The Japanese idea is to be immersed in

the reverberations. I think of this spatial atmosphere as a colored silence in my music.

There is no one word in English to translate the Japanese 'ma'. 'Ma' refers to an interval in time and space. Physically, it is a silence and nothingness. Aesthetically, it means 'meaningful nothingness' in terms of space and time. It is an aesthetics of elimination. The Japanese say a good 'ma' defines the quality of works among Japanese arts, music, theater, dance, architecture, gardening, calligraphy etc. In these works, 'ma' is used as more than a conjunction. For example, it can be used to describe a dramatic pause in spoken lines in theater. In music 'ma' works as a big breath together with other musicians without a conductor and score, and in painting as a big empty space. The whole of the work is enhanced if it is a good 'ma'. With regard to music, 'ma' is felt mutually within the time frame of each player. However, through listening to one another carefully, musicians create a global beat through interaction. The question is evidently timing: when, how long, and how 'ma' is applied. 'Ma' is obiously a very important concept in understanding Japanese culture. This concept has religious, psychological and physical significance for every Japanese, extending into every aspect of Japanese life. I am always concerned about using 'ma' in my piece. 'Ma' is not something which is created or composed artificially but a thing born naturally in the human imagination. Since there is no system for using 'ma', I try to make it work in a musically

intuitive manner. I believe that it should be perceived naturally as beauty.

My use of eight speakers is important with regard to 'ma' in this piece. Although I do not have a specific rule for speaker setting, I wanted to create a balanced space with music, considering a sense of Japanese 'ma'.

SECTION 2

"REAL TIME SIGNAL PROCESSING"

Technical Schemes

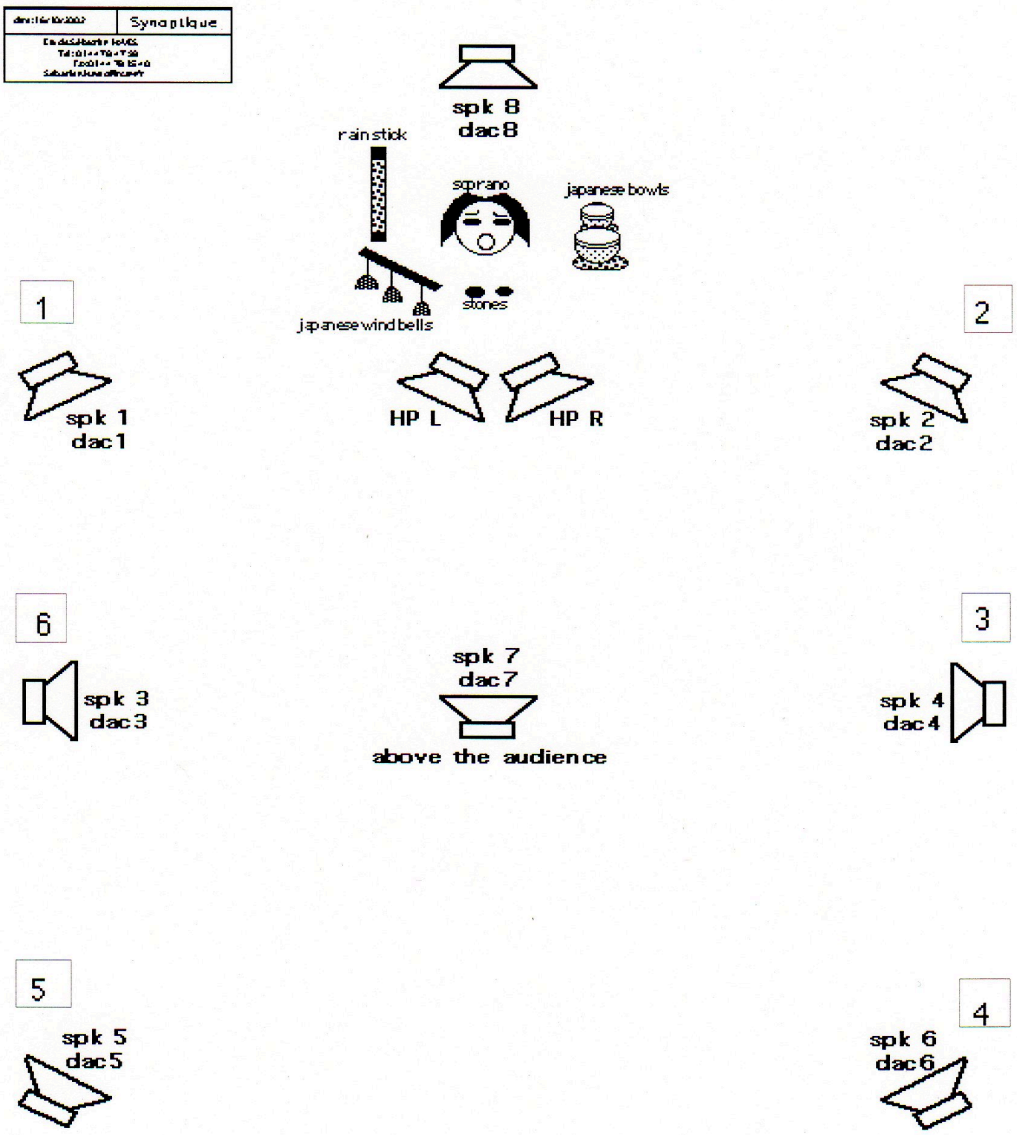
Technical Scheme 1 (next page) is a speaker placement particular to the concert hall, IRCAM, espace de projection. Spk 1-8 are used for electronic sounds. Spk 7 is attached to the ceiling and spk 8 on the stage just behind the singer. Spk 1-6 are used for real time effects with Max/Msp. Since the Max/Msp *spat~* object, which is a real-time spatial sound processing software system (see page 38), works in a circular direction, I numbered speakers in this way. In Technical Scheme 1, I enclosed them in a box.

Next, I am going to explain Technical Scheme 2 in detail. I use four microphones, C414, E609, and two KM140s. C414, TLII AKG are used for voice and the overall instruments. One of the KM104 Neumann microphones detects the rain stick and Japanese bells. The other one does the same for the Japanese temple bowls. Both microphones are cardioid with a directional pickup pattern that is most sensitive to sounds coming from the front and sides, and both use static technology. E609, Sennheiser is used for the stones. It is a hypercardioid microphone, which is a variation of the cardioid pickup pattern. It is very selective in terms of directionality to prevent feedback, specifically focusing on the stones. Only when I have a *sträng* effect (see page 30) in real time for the stones, the

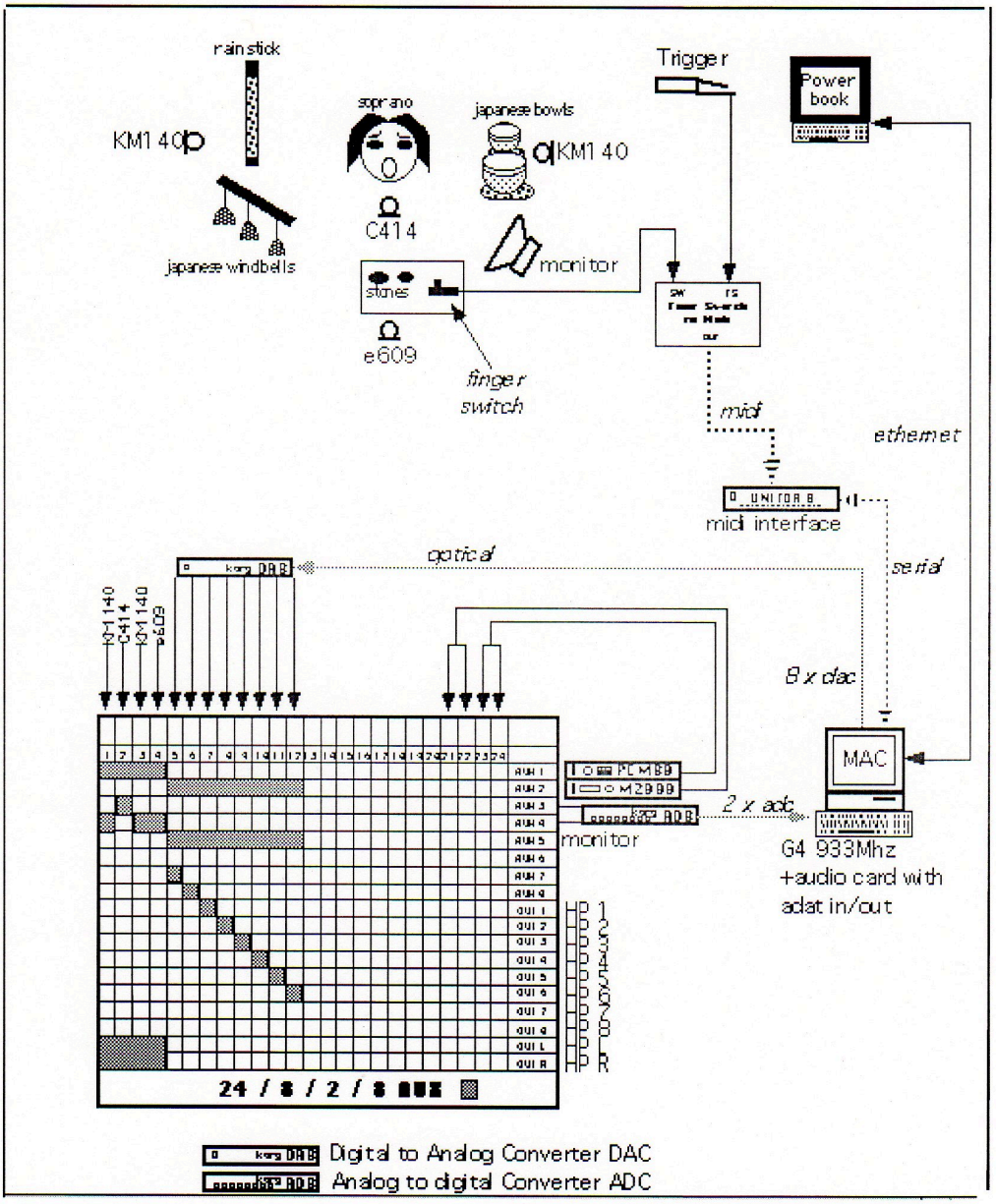
volume of the microphone must be turned down and the channels of the two KM140's are muted to avoid noise and feedback. A sound technician at IRCAM who set up the concert told me that this setting is very specific to the concert hall acoustics and the level of amplification is linked to the proximity of the speakers to the microphones. I have also controlled the volume balance with Max/Msp.

A monitor is a stage speaker designed to send sound to the musician directly. To trigger electronic sounds and real time effects, the singer uses a finger switch for sections 1 and 2, and a foot pedal for section 3. A Power Book on stage is used as a stopwatch for the singer to follow. It starts working with event 52 in the beginning of section 3 and continues until the end of the piece. The PCM80 is a lexicon reverb used for voice and percussion. The signals of different microphones with different levels are sent by using auxiliary outputs. Then outputs of the reverberations are sent back to the mixing board as standard sound sources. These settings are quite delicate. During the rehearsals and the concert, the professional technician at Ircam controlled them perfectly.

TECHNICAL SCHEME 1: speaker placement



TECHNICAL SCHEME 2



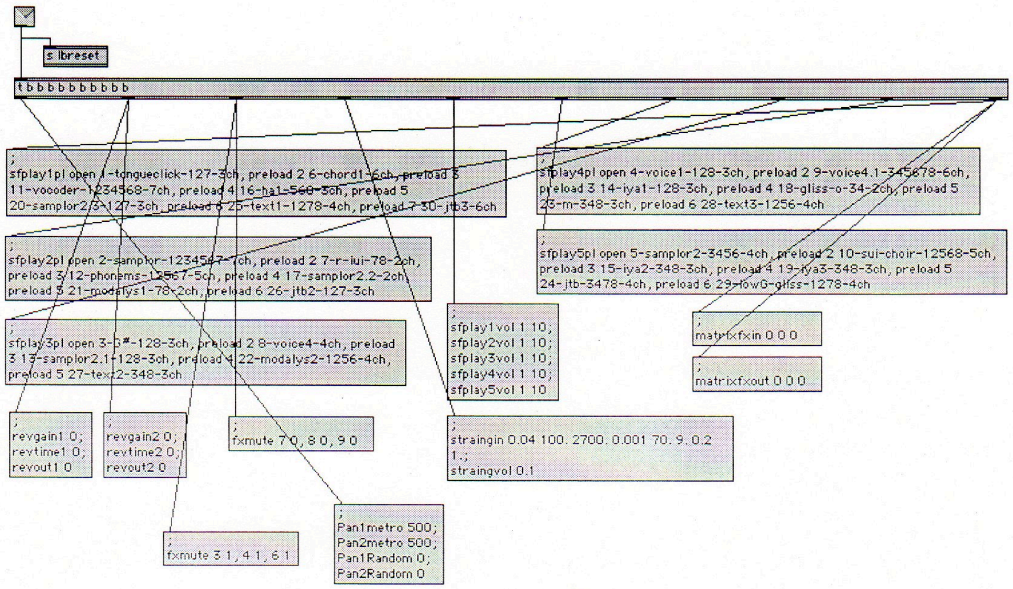
Max/Msp

Max/Msp, distributed by the American company, Opcode, and originally developed at IRCAM, is used for creating real time computer programming for live music. Max integrated the library Msp and added the functions of analysis, synthesis and treatment of sound in real time. It allows composers to program the interaction between musicians and computer.

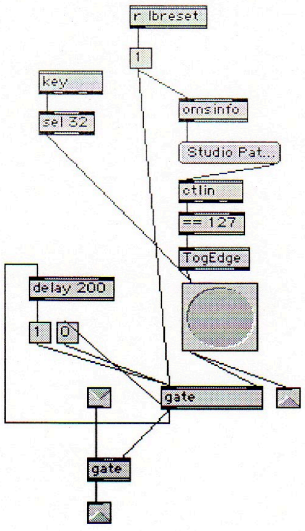
Next, I will explain a real time effect created in Max/Msp. Example 8 (next page) is a Concertpatch which was created by Max/Msp. It consists of three parts. The first one (numbered 1) makes everything run. The second one (numbered 2) operates the real time signal processing. The third one (numbered 3) triggers the electronics.

Example 9 shows a subpatch called *p INITS* within the Concertpatch. A lower case letter, '*p*" represents a patcher which is a subpatch within a patch. When you open the Concertpatch, this subpatch initializes automatically. Here, it consists of soundfiles, *pcontrol* for section 1, the first parameter of *sträng* effect and a reset of *reverb* and *matrix*. I will explain about soundfiles in Section 3. I will also talk about other parameters individually later in this section.

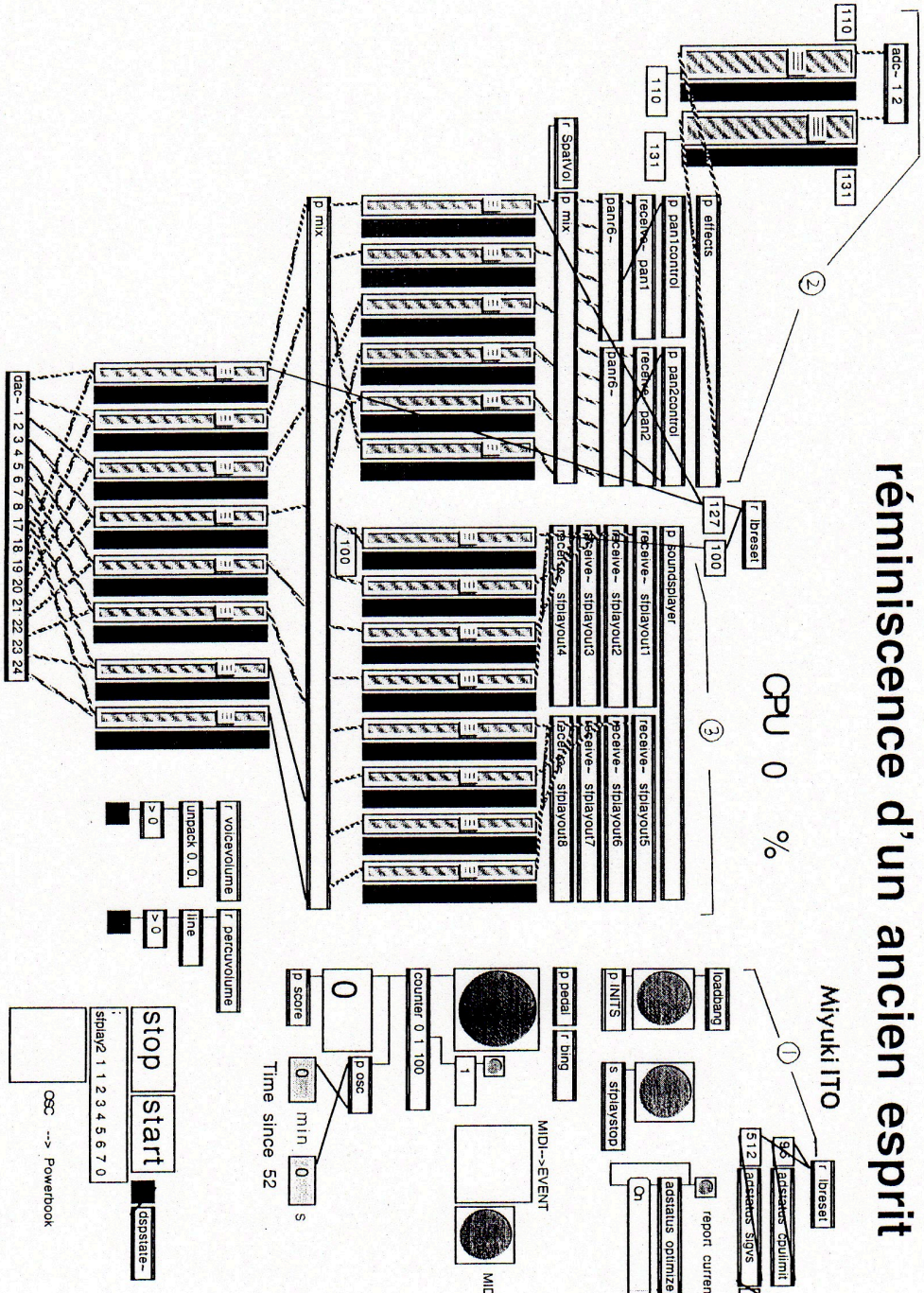
Example 9: *p INITS*



Example 10: *p pedal*

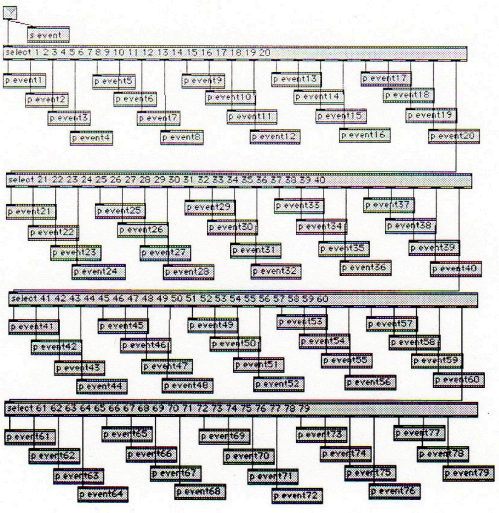


Example 8: Concertpatch



Example 10 is a subpatch of *p pedal* and Example 11 is subpatch of *p score*. There are 79 events. I also colored *p event*s 24, 52, 54, 56, 58, 60 with red in the *p score* because inside of them, there are parameters of *pcontrols*, which open and close subwindows within patchers to save CPU utilization. If I click the event after the colored one without clicking it, the effect does not work and the computer will crash because the effect is still muted. In each section, effects I do not use are muted. For example, in section 1, *harm2, xsynthx,* and *filtrage* are muted. The other muted effects are in section 2, *vocoder*, *sträng, fft-xover, xsynthx* and in section 3, *fft-xover* (see Example 12). A subpatch of each event contains parameters for effects and soundfile. I will explain about them in detail later. In addition, event 22 was triggered by the composer because it was difficult for the singer to do it while she was singing. Event 69 was triggered automatically after event 68, for the same reason.

Example 11: *p score*



Example 12: table of the effects and pcontrol

```
Effects
 0     reverb 1
 1     reverb 2
 2     transp/delay
 3     vocoder
 4     straing
 5     filter
 6     fft-xover
 7     harm2
 8     xsynthx
 9     filtrage

matrixfxin                    matrixfxout
0     voice                   0     pan 1
1     percussion              1     pan 2
2     inlet 1                 2     inlet 1
3     inlet 2                 3     inlet 2

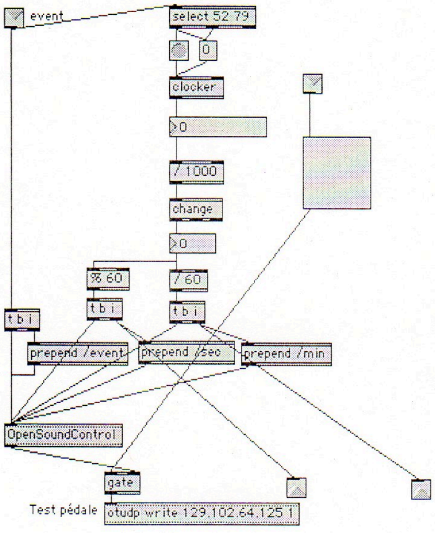
pcontrol - remote control of patcher window
      enable 1 - enable the patcher window for MIDI
      enable 0 - disable the patcher window for MIDI

Section 1   INITS  -fxmute 7 0, 8 0, 9 0
Section 2   Event24-fxmute 3 0, 4 0, 6 0, 8 0 fxmute 7 1, 9 1
Section 3   Event52-fxmute 6 0, 3 0, 4 0      fxmute 7 1, 9 1, 8 1
            Event54-fxmute 8 0
            Event56-                          fxmute 4 1
            Event58-fxmute 4 0                fxmute 3 1
            Event60-fxmute 3 0
```

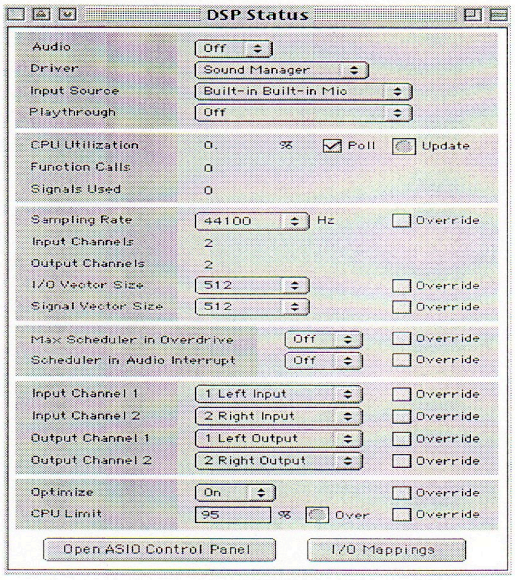
Example 13 shows a *p ocs.* A stopwatch connected to a computer on

stage starts working from events 52 till 79 in section 3. The singer needs to follow it, as I mentioned before. IP address for a computer should be changed when I use a different computer. Furthermore, to reduce CPU utilization, an optimization setting should be on, a vector size of 512 and a CPU limit of 95% on the DSP status. (see Examples 8 and 14)

Example 13: *p osc*



Example14: DSP status

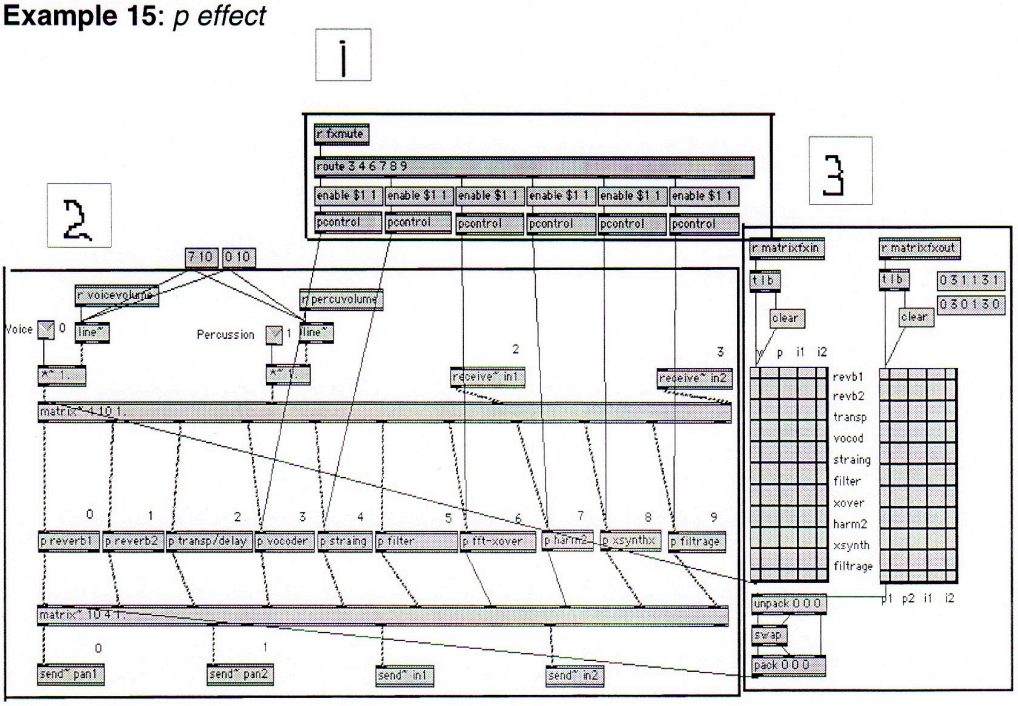


Next, I would like to talk about Concertpatch number 2 in Example 8. A *adc~* object output audio signal from logical channel pairs 1/2. The output of 2 channels is sent to *gain~* object which send them to *p effect* (see Example 15). The *adc~* 1 is for a voice and *adc~ 2* for percussion. The volume size is set up for them on the Concertpatch.

Example 15 is a subpatch of *p effect*. The top of this patch consists of *pcontrol*-related objects to reduce CPU utilization as I mention above. The six effects, *vocoder, sträng, fft-xover, harm2, xsynthx, and filtrage,* numbered 3, 4, 6, 7, 8 and 9 in the patch, are muted by *pcontrol* objects when they are not used in each section, respectively.

Below the *pcontrol*-related objects, there are ten subpatches of effects for a voice and percussion. On the right side of *p effect* , there is the *matrixctrl* object which is a matrix switch control. It generates messages for the *matrix~* object. The messages are written like (input #, output #, gain #) in each *p event* under the score file (Example 11). Example 17 shows all parameters for each event. The number for each effect in Example 12 is the number of inputs. The number of *matrixfxin* and *matrixfxout* are the number of outputs. The effect works when the gain is 1. It stops when the gain is 0. For example, event 4 has an infinite reverberation until event 8. I send all messages between event 4 and 8 of *matrixfxout* in event 4. If you add a message each time, the previous message will be turned off. When the effect continues, I send all messages until the event where the effect is stopped. I sometimes have the effect with an added effect like event 6 in ex. 16. It uses both *pan1* and *pan2*. The voice sends to *vocoder* and to *pan1*, using *Spatline1*. It also sends to *vocoder* and to *inlet 1* and adds the *reverb2* and then sends to *pan2*, using *Spatline2*.

Example 15: *p effect*

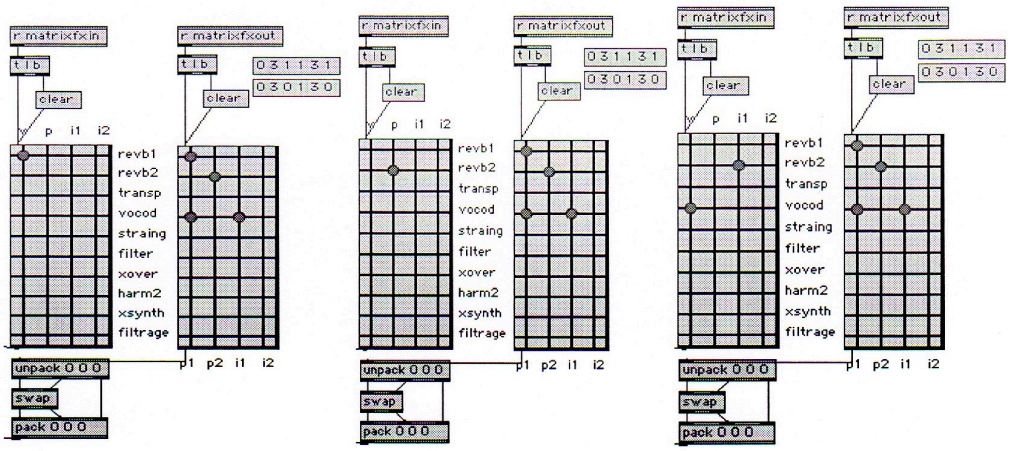


Example 16:

event 4 : event 4 delayed 1100 : event 6

matrixfxin 0 0 1 matrixfxin 1 1 1 matrixfin 0 3 1 2 1 1

matrixfiout 0 0 1 1 1 1 0 3 1 2 3 1



Example 17: *p event* 1~79

Section 1

Event1 ; voicevolume 2 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 ; revgain1 92; revtime1 102; revout1 106 ; Spatline1 source Az 0 1 0
delay 3000 - ; voicevolume 0 10
Event2 ; sfplay1vol 1 ; sfplay1 1 1 2 7 0 0 0 0 0
Event3 delay 2600 - ; sfplay2 1 1 2 3 4 5 6 7 0
delay 3500 - ; voicevolume 2 10 ; matrixfxin 0 2 1 ; matrixfxout 0 2 1; **2fshift 1** ; PshifyDelay1 600 ; **PshifyFreq1 -152.659302** ; PshiftDelay2 900 ; **PshiftFreq2 292.670471** ; PshtFeedBack 0.7 ; Spatline1 source Az 180 1 0 1; Spatline1 source Az 90 0 0 2000
delay 4200 - ; voicevolume 2, 0 10
Event4; voicevolume 2 10 ; **matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 0 3 1 2 3 1** ; revgain1 92; revtime1 127; revout1 100 ; Spatline1 source Az 0 1 0
delay 100 - ; voicevolume 0 10
delay 1100 - ; percuvolume 2 10 ; **matrixin 1 1 1** ; revgain1 110; revtime1 117; revout1 115 ; Pan2Random 1 ; Pan2metro 700
delay 1000 - ; percuvolume 0 100
Event5 ; Pan2Random 0
Event6; voicevolume 3 10 ; **matrixfxin 0 3 1 2 1 1** ; Spatline2 source Az 180 1 0 1000 ; Spatline2 source Az 0 1 0 4000; Spatline1 source Az 0 1 0 ; **VocoderQ1 20 ; Vocoderamp1 20 ; Vocoderenvon 1 ; Vocoderpass 3614 ; Vocodersplrate 700 ; VocoderQ2 15 ; Vocoderamp2 14 ; VocoderSfplay 1 ;bandfreq 1000 3000 1500 4500 500 2000 5000 1000 0**; revgain2 104; revtime2 120; revout2 63
delay 3400 - ; voicevolume 0 10
Event7 delay 600 - ; sfplay3 1 1 2 8 0 0 0 0 0
Event8; sfplay4 1 1 2 8 0 0 0 0 0 ; revoutfadeout1 100, 0 300
delay 500 - revgain1 0; revtime1 0; revout1 0 ; **matrixout 0 0 0**
Event9 ; sfplay5 1 3 4 5 6 0 0 0 0
Event10 delay 1100 - ; sfplay1 2 1 2 3 4 5 6 0 0
Event11 ; voicevolume 2 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 1 2 1 ; revgain1 125; revtime1 122; revout1 125 ;Spatline1 source Az 180 1 0
delay 100 - ; voicevolume 0 10
Event12 delay 400 - ; voicevolume 0.8 10 ; matrixfxin 0 1 1 ; revgain2 95; revtime2 127; revout2 95 ; Spatline2 source Az 0 0 0
delay 1000 - ; voicevolume 0 10

Event13; voicevolume 1 10 ; matrixfxin 0 0 1 ; revgain1 100; **revtime1 127; revout1 100** ; Spatline1 source Az 0 1 0
delay 300 - ; voicevolume 0 10
delay 650 - ; voicevolume 4 10 ; matrixfxin 0 0 0 0 2 1 ; PshifyDelay1 500 ; PshifyFreq1 0 ; PshtFeedBack 0.85 ; 2shift 0 ; Pan2Random 1 ; Pan2metro 800
delay 5000 - ; voicevolume 0 10
Event14 delay 500 - ; voicevolume 1 10 ; matrixfxin 0 0 1 ; revgain1 92; **revtime1 127; revout1 106** ; Spatline1
source Az 180 1 0

delay 2200 - ; voicevolume 0 10

Event15; **revoutfadeout1 106, 0 300 ; revoutfadeout2 100, 0 300** ; Pan2Random 0
delay 500 - ; **matrixfxout 0 0 0 1 1 0** 1 2 0 ; **revgain1 0; revtime1 0; revout1 0 ; revgain2 0, revtime2 0,revout2 0**

Event16 ; sfplay3vol 0.4 1
delay 950 - sfplay3 2 1 2 3 4 0 0 0 0

Event17 ; sfplay4vol 0.6 1 ; sfplay4 2 3 4 5 6 7 8 0 0

Event18 delay 1000 - ; voicevolume 1.3 10 ; matrixfxin 0 2 1 0 0 1 ; matrixfxout 0 0 1 2 2 1 ; PshifyDelay1 2100 ; PshifyFreq1 0 ; PshtFeedBack 0.9 ; 2shift 1 ; PshifyDelay2 200 ; PshifyFreq2 0 ; revgain1 110; revtime1 115; revout1 106 ; Spatline1 source Az 0 1 0 1 ; Spatline1 source Az 180 1 0 5000
delay 6000 - ; voicevolume 0 10
delay 8000 - ; matrixfxout 0 2 0 ; PshtFeedBack 0

Event19 delay 1000 - ; percuvolume 1 10 ; matrixfxin 1 0 1 ; matrixfxout 0 0 1 ; revgain1 100; revtime1 100; revout1100; Spatline1 source Az 180 1 0 1 ; Spatline1 source Az 0 1 0 3000
delay 4000 - ; percuvolume 0 500

Event20 ; sfplay5vol 1.5 ; sfplay5 2 1 2 5 6 8 0 0 0

Event21; percuvolume 1 10
delay 100 - ; matrixfxin 1 4 1 1 5 1 2 0 1 ; matrixfxout 0 4 1 1 0 1 2 5 1 ; **Ftr1delay 500 ; Ftr1inputapm 0.85; Ftr1inputfedbk 0.63; Ftr2delay 2000 ; Ftr2inputapm 0.77 ; Ftr2inputfedbk 0.67 ; Ftr1vibspeed 0; Ftr1amp 300 ;**

Ftr1centfreq 7500 ; revgain1 90; revtime1 110; revout1 110 ; Pan1AZ 0, -450 20000 ; Pan2AZ 0, 450 20000
delay 27000 - ; percuvolume 0 10

Event22 ; voicevolume 0.6 10 ; matrixfxin 0 2 1 ; matrixfxout 0 2 1 2 6 1 0 0 1 1 0 1 ; PshifyDelay1 500; PshifyFreq1 181.543513 ; PshtFeedBack 0.9 ; 2shift 1 ; PshifyDelay2 700 ; PshifyFreq2 107.766602 ; Spatline1 source Az 90 1 0 1 ; Spatline1 source Az 180 1 0 7000
delay 1200 - ; voicevolume 0 10

Event23 ; voicevolume 0.6 10 ; matrixfxin 0 6 1 2 0 1 ; XoverSfplay 1 ; Xoverfreq 1 ; revgain1 95; revtime1 100; revout1 86; Pan1AZ 0, 360 7000 ; Pan2AZ 0, -360 7000
delay 7000 - ; voicevolume 0 10

Section 2

Event24 ; sfplay1 3 1 2 3 4 5 6 8 0 ; matrixfxout 0 0 0
delay 1000 - ; fxmute 3 0, 4 0, 6 0, 8 0
delay 2000 - ; fxmute 7 1, 9 1

Event25 ; voicevolume 4 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 ; revgain1 78; revtime1 115; revout1 86 ; Spatline1 source Az 0 1 0 1 ; Spatline1 source Az 0 0 0 10
delay 900 - ; voicevolume 0 10

Event26 ; voicevolume 4 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 ; revgain1 115; revtime1 120; revout1 100; Spatline1 source Az 0 1 0 1 ; Spatline1 source Az 180 1 0
delay 100 - ; voicevolume 0 10
delay 500 - ; voicevolume 1 10 ; matrixfxin 0 0 1 ; revgain2 101; revtime2 108; revout2 100 ; Spatline2 source Az 0 1 0
delay 1500 - ; voicevolume 0 10

Event27; voicevolume 0.8 10 ; matrixfxin 0 2 1 0 5 1 ; matrixfxout 0 2 1 0 5 1 ; revgain1 0; revtime1 0; revout1 0 ; revgain2 0 revtime2 0; revout2 0 ; PshifyDelay1 200; PshifyFreq1 0 ; PshtFeedBack 0.98 ; 2shift 0 ; **Ftr1vibspeed 7 ; Ftr1amp 180 ; Ftr1centfreq 2000** ; Pan1Random 1 ; Pan1metro 300

delay 4000 - ; voicevolume 0 10

Event28; voicevolume 2 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 1 0 1 2 2 1 ;PshifyDelay1 500;PshifyFreq1 83.25116; PshtFeedBack 0.9 ; **2shift 1** ; PshifyDelay2 400 ; PshifyFreq2 99.002808 ; **3shift 1**; PshifyDelay3 200 ; PshifyFreq3 -206.94928 ; revgain1 102; revtime1 115; revout1 104 ; Spatline2 source Az 0 1 0 ; Pan1Random 0

delay 300 - ; voicevolume 0 10

Event29 ; sfplay2vol 0.8 1 ; sfplay2 3 1 2 5 6 7 0 0 0

Event30 ; voicevolume 3 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 ; revgain1 112; revtime1 116; revout1 115 ; Spatline1 source Az 0 0 0

delay 1600 - ; voicevolume 0 10

Event31 ; voicevolume 5 10 ; matrixfxin 0 0 1 0 2 1 ; matrixfxout 0 0 1 0 2 1 ; revgain1 102; revtime1 120; revout1 104 ; PshifyDelay1 400; PshifyFreq1 206.94928; PshtFeedBack 0.7 ; 2shift 0 ; 3shift 0 ; Pan1Random 0 ; Spatline1 source Az 180 1 0

delay 1500 - ; voicevolume 0 10

Event32delay 1100 - ; voicevolume 5 10 ; matrixfxin 0 7 1 2 0 1 ; matrixfxout 0 0 1 1 0 1 2 7 1 ; revgain1 110, revtime1 120; revout1 121 ; **harm2transp -150 ; harm2transpsmth 849 ; harmswindow 100 ;harm2windowsmth 3000 ;harm2delay 200 ; harm2delaysmth 100 ; harm2delayfreq 0.96 ; harm2delayamp 0.73 ; harm2feedbk 0.85 ; harm2lpasin 1 ; harm2cutofffreq 2000** ; Pan1AZ 0, 180 1500 ; Pan2AZ 0, -180 1500

delay 4000 - ; voicevolume 0 10

Event33 ; sfplay3vol 1 1 ; sfplay3 3 1 2 8 0 0 0 0 0

Event34 ; sfplay4vol 1 1 ; sfplay4 3 1 2 8 0 0 0 0 0

Event35 ; voicevolume 0.4 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 0 0 1 2 2 1 1 0 1 ; VocoderSfplay 1 ; PshiftDelay1 0 ; PshiftFreq1 2500 ; 2shift 0 ; 3shift 0 ; revgain1 115; revtime1 121; revout1 122 ; Pan1AZ 0, 180 1500 ; Pan2AZ 0, -180 1500

delay 4000 - ; voicevolume 0 10

Event36 delay 600 - ; voicevolume 4 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 ; revgain1 115; revtime1 118; revout1 115 ; Spatline1 source Az 0 0 0

delay 1500 - ; voicevolume 0 10

Event37 ; sfplay5 3 3 4 8 0 0 0 0 0

Event38 ; voicevolume 1 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 0 1 , revgain1 115; revtime1 118; revout1 115 ; Spatline1 source Az 90 1 0 1 ; Spatline1 source Az -90 1 0 2500 ; Spatline2 source Az -90 1 0 1; Spatline2 source Az 90 1 0 2500

delay 3000 - ; voicevolume 0 10

Event39 ; sfplay1 4 5 6 8 0 0 0 0 0

Event40 ; sfplay2 4 1 2 0 0 0 0 0 0 ; voicevolume 4 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 ; revgain1 110; revtime1 127; revout1 115 ; Spatline1 source Az 0 1 0

delay 2000 - ; voicevolume 0 10

Event41 ; revoutfadeout1 115, 0 300

delay 500 - ; matrixfxout 0 0 0 ; revgain1 0; revtime1 0; revout1 0

delay 2000 - ; voicevolume 1 10 ; matrixfxin 0 0 1 ; matrixfxout 1 0 1 0 1 1 0 2 1 ; revgain1 90; revtime1 127; revout1 85 ; Spatline2 source Az 180 1 0 1 ; Spatline2 source Az 0 1 0 3000

delay 2500 - ; sfplay4 4 3 4 0 0 0 0 0 0

delay 2800 - ; voicevolume 0 10

Event42 ; voicevolume 4 10 ; matrixfxin 0 1 1 0 2 1 ; PshifyDelay1 500; PshifyFreq1 -144.365234; PshtFeedBack 0.5 ; 2shift 1 ; PshifyDelay2 300 ; PshifyFreq2 182.253967 ; revgain2 110; revtime2 127; revout2 110 ; Spatline1 source Az 0 0 0 1

delay 1200 - ; voicevolume 0 10

Event43 ; sfplay5 4 5 6 8 0 0 0 0 0 ; sfplay5vol 0.5 1 ; revoutfadeout1 85, 0 600 ; revoutfadeout2 110, 0 600
delay 1000 - ; matrixfxout 1 0 0 0 2 0 0 1 0 0 2 0 ; revgain1 0; revtime1 0; revout1 0 ; revgain2 0; revtime2 0; revout2 0

Event44 ; voicevolume 4 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 0 0 1 2 2 1 ; PshifyDelay1 400; PshifyFreq1 -58.867432 ; PshtFeedBack 0.5 ; 2shift 1 ; PshifyDelay2 800 ; PshifyFreq2 -17.462219 ; revgain1 100; revtime1 112; revout1 115 ; Spatline1 source Az 0 0 0
delay 2000 - ; voicevolume 0 10

Event45 delay 1000 - ; sfplay1 5 1 2 7 0 0 0 0 0

Event46 ; voicevolume 5 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 0 1 ; revgain1 112 ; revtime1 100 ; revout1 116 ; Spatline1 source Az 135 1 0 10 ; Spatline1 source Az -45 1 0 2000 ; Spatline2 source Az -135 1 0 10; Spatline2 source Az 45 1 0 2000
delay 1000 - ; voicevolume 0 10

Event47 ; sfplay2 5 7 8 0 0 0 0 0 0 ; sfplay2vol0, 0.6 3000 ; voicevolume 6 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 0 0 1 2 2 1; Vocodersfplay 2 ; PshiftDelay1 0 ; PshiftFeedBack 0 ; PshiftFreq1 3500 ; 2fshift 0 ; reverbgain1 102 ; revtime1 117 ; revout1 110 ; Spatline1 source Az 0 0 0
delay 3000 - ; voicevolume 0 10

Event48 ; sfplay3 4 1 2 5 6 0 0 0 0 ; voicevolume 1.5 10 ; matrixfxin 0 5 1 2 0 1 ; matrixfxout 0 0 1 1 0 1 2 5 1; **Ftr1inputapm 0.65 ; Ftr1vibspeed 6.15 ; Ftr1amp 200 ; Ftr1centfreq 1200** ; revgain1 92 ; revtime1 102 ; revout1 106 ; Pan1AZ 90, 450 30000 ; Pan2AZ -90, -450 3000
delay 4500 - ; voicevolume 0 10

Event49 ; voicevolume 1.5 10 ; matrixfxin 0 5 1 ; matrixfxout 0 5 1 1 5 1 ; **Ftr1inputapm 0.3 ; Ftr1vibspeed 11 ; Ftr1amp 215; Ftr1centfreq 2000 ; Ftr1delay 100** ; Spatline1 source Az 90 1 0 ; Spatline2 source Az -90 1 0
delay 8500 - voicevolume 0 10

Event50 ; voicevolume 4 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 2 7 1 ; revgain1 92 ; revtime1 118 ; revout1 106; Spatline1 source Az 0 1 0
delay 1000 - ; voicevolume 0 10

Event51 ; voicevolume 7 10 ; matrixfxin 0 7 1 2 1 1 ; matrixfxout 1 1 1 2 7 1 ; revgain2 115 ; revtime2 122 ; revout2 115 ; Spatline2 source Az 180 1 0 1 ; Spatline2 source Az 0 0 0 2000 ; **harm2transp -300 ; harm2transpsmth 800; harm2window 50 ; harm2windoasmth 2000 ; harm2delay 300 ; harm2delaysmth 200 ; harm2delayfreq 0.96;harm2delayamp 0.8 ; harm2feedbk 0.85; harm2lpasin 1;harm2cutofffreq 2000**
delay 1000 - ; sfplay4 5 3 4 8 0 0 0 0 0 ; sfplay4vol 0.8 1000
delay 3000 - ; voicevolume 3, 0 10

Section 3

Event52 ; sfplay5 5 3 4 7 8 0 0 0 0 ; sfplay5vol 1 1 ,fxmute 6 0, 3 0, 4 0
delay 1000 - ; matrixfxout 0 0 0 ; fxmute 7 1, 9 1, 8 1
delay 14000 - ; sfplay1 6 1 2 7 8 0 0 0 0

Event53 ; voicevolume 1 10 ; matrixfxin 0 7 1 0 8 1 2 0 1 ; matrixfxout 0 7 1 0 0 1 2 8 1 ; revgain1 112 ; revtime1 106 ; revout1 105 ; **harm2transp -100 ; harm2transpsmth 800 ; harm2window 50 ; harm2windoasmth 3000 ; harm2delay 300 ; harm2delaysmth 200 ; harm2delayfreq 0.96 ; harm2delayamp 0.73 ; harm2feedbk 0.85 ; harm2lpasin 1 ; harm2cutofffreq 1200** ; XsynthSfplay 1 ; Spatline1 source Az 0 1 0 1 ; Spatline1 source Az 180 1 0 2000
delay 7000 - ; voicevolume 0 10 ---when the event54 trigers

Event54 ; voicevolume 7 10 ; matrixfxin 0 5 1 0 1 1 ; matrixfxout 1 1 1 1 5 1 0 0 1 2 2 1 ; Ftr1inputapm 0.6 ; Ftr1vibspeed 9 ; ftr1amp 150 ; Ftr1centfreq 1000 ; Ftr1delay 0 ; revgain2 100 ; revtime2 96 ; revout2 100 ; Spatline2 source Az 0 1 0 1 ; Spatline2 source Az 0 0 0 800
delay 1000 - ;fxmute 8 0
delay 3000 - ; voicevolume 0 10 ---when event55 trigers

Event55 ; voicevolume 3 10 ; matrixfxin 0 2 1 2 0 1 ; PshiftDelay1 100 ; PshiftFeedBack 0.4 ; PshiftFreq1 -14.896 ; 2fshift 1; PshiftDelay2 50 ; Pshiftfreq2 -29.367868 ; 3fshift 1 ; PshiftDelay3 30 ; PshiftFreq3 -43.42765 ; revgain1 100; revtime1 110 ; revout1 115; Spatline1 sorce Az 0 0 0
delay 4000 - voicevolume 0 10

Event56 ; percuvolume 0.5 10 ; **straingvol 0.1** ; matrixfxin 1 9 1 ; **matrixfxout 0 9 1 1 0 1 2 4 1** ; **straingin 0.07 100. 2650. 0.001 79. 9. 5. 1.** ; Spatline1 source Az 0 1 0 ; Spatline1 source Az 180 1 0 30000 ; **fxmute 4 1**
delay 40000 - ; percuvolume 0 10

Event57 ; voicevolume 1.5 10 ; matrixfxin 0 4 1 2 0 1 ; Pan2Random 1 ; Pan2metro 200 ; revgain1 100 ; revtime1 120 ; revout1 115
delay 4200 - ; voicevolume 0 10

Event58 ; voicevolume 2 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 0 0 1 1 0 1 2 2 1 ; VocoderSfplay 3 ; PshiftDelay1 0; Pshiftfreq1 0 ; PshifyFeedBack 0 ; 2shift 0 ; 3shift 0 ; revgain1 115 ; revtime1 121 ; revout1 122 ; Pan1 AZ 90, -90 1000 ; Pan2Random 0 ; Pan2AZ -90, 90 1000
delay 1000 - ; fxmute 4 0
delay 2000 - ; fxmute 3 1
delay 3000 - ; voicevolume 0 10

Event59; percuvolume 12 10 ; matrixfxin 1 3 1 2 1 1 ; matrixfxout 0 1 1 1 1 1 2 3 1 ; **VocoderQ1 20 ; Vocoderamp1 10; Vocoderenvon 1 ; Vocoderlpas 3614 ; Vocodersplrate 700 ; VocoderQ2 10 ; Vocoderamp2 10 ; Vocodersfplay 1 ; bankfreq 0 3000 1500 4500 5000 2000 500 1000 0** ; revgain2 104 ; revtime2 120 ; revout2 115 ; Pan1AZ 45, 180 10000 ; Pan2AZ -45, -180 10000
delay 10000 - ; percuvolume 0 10

Event60 ; voicevolume 1 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 ; revgain1 104 ; revtime1 127 ; revout1 115 ; Spatline1 source Az 0 0 0
delay 300 - ; voicevolume 0 10
delay 1000 - ; fxmute 3 0

Event61 ; sfplay2 6 1 2 7 0 0 0 0 ; sfplay2vol0, 0.4 500 ; voicevolume 2 10 ; matrixfxin 0 1 1 ; revgain2 104 ; revtime2 127; revout2 115 ; Spatline2 source Az 0 0 0
delay 200 - ; voicevolume 0 10

Event62 ; sfplay3 5 3 4 8 0 0 0 0 0 ; revoutfadeout1 115, 0 1500 ; revoutfadeout2 115, 0 1500
delay 2000 - ; matrixfxout 0 0 0 1 1 0 ; revgain1 0 ; revtime1 0 ; revout1 0 ; revgain2 0 ; revtime2 0 ; revout2 0

Event63 ; percuvolume 1.5 10 ; matrixfxin 1 0 1 ; matrixfxout 0 0 1 1 1 1 2 2 1 ; revgain1 85 ; revtime1 103 ; revout1 73 ; Spatline1 source Az 0 1 0 1 ; Spatline1 source Az 180 1 0 15000
delay 25000 - ; percuvolume 0 10

Event64 ; percuvolume 4 10 ; matrixfin 1 2 1 2 1 1 ; PshiftDelay1 150 ; PshiftFeedBack 0.5 ; PshiftFreq1 -32.313927 ; 2fshift 0 ; 3fshift 0 ; revgain2 100 ; revtime2 115 ; revout2 105 ; Spatline2 source Az 0 0 0
delay 3000 - ; percuvolume 0 10

Event65 ; voicevolume 6 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 2 2 1 ; revgain1 115 ; revtime1 120 ; revout1 115; Spatline1 source Az 0 0 0
delay 500 - ; voicevolume 0 10

Event66 ; voicevolume 2 10 ; matrixfxin 0 2 1 2 1 1 ; PshiftDelay1 600 ; PshiftFeedBack 0.4 ; PshiftFreq1 0 ; 2fshift 1; PshiftDelay2 400 ; PshiftFreq2 157.152771 ; 3fshift 1 ; PshiftDelay3 600 ; PshiftFreq3 -400.437073 ; revgain2 115; revtime2 90 ; revout2 100 ; Spatline2 source Az 0 1 0
delay 20000 - ; voicevolume 0 10

Event67 ; voicevolume 10 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 0 0 1 2 2 1 1 1 1 ; PshiftDelay1 2500 ; PshiftFeedBack 0.2 ; PshiftFreq1 -114.174797 ; 2fshift 0 ; 3fshift 0 ; revgain1 110 ; revtime1 110 ; revout1 115; Spatline1 source Az 180 1 0
delay 50000 - ; voicevolume 0 10

Event68 ; voicevolume 2 10 ; matrixfxin 0 1 1 ; revgain2 115 ; revtime2 115 ; revout2 115 ; Spatline2 source Az 0 0 0
delay 2500 - ; voicevolume 0 10 ----to trip 69 automatically

Event69 ; sfplay4 6 1 2 5 6 0 0 0 0

Event70 ; voicevolume 4 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 0 0 1 1 0 1 2 2 1 ; VocoderSfplay 4 ; PshiftDelay1 400 ; PshiftFeedBack 0.1 ; PshiftFreq1 0 ; 2fshift 0 ; 3fshift 0 ; revgain1 125 ; revtime 120 ; revout1 125 ; Pan1AZ 90, 180 2000 ; Pan2AZ -90, -180 2000
delay 3500 - ; voicevolume 0 10

Event71 ; sfplay5 6 1 2 7 8 0 0 0 0 ; sfplay7vol 0.4 1

Event72 ; voicevolume 1 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 ; revgain1 85 ; revtime1 127 ; revout1 85 ; Spatline1 source Az 180 1 0 1 ; Spatline1 source Az 0 1 0 15000
delay 3000 - ; voicevolume 0 10

Event73 ; voicevolume 1.5 10 ; matrixfxin 0 1 1 ; revgain2 90 ; revtime2 127 ; revout2 85 ; Spatline1 source Az 0 0 0
delay 100 - ; voicevolume 0 10
delay 5000 - ; sfplay1 7 1 2 3 4 5 6 0 0

Event74 ; revoutfadeout1 85; 0 500 ; revoutfadeout2 85; 0 500
delay 500 - matrixfxout 0 0 0 1 1 0 ; revgain1 0 ; revtime1 0 ; revout1 0 ; revgain2 0 ; revtime2 0 ; revout2 o

Event75 ; voicevolume 2 10 ; matrixfxin 0 2 1 2 0 1 ; matrixfxout 0 0 1 1 0 1 2 2 1 0 1 1 1 1 1 ; PshiftDelay1 700; PshiftFeedBack 0.1 ; PshiftFreq1 166.502319 ; 2fshift 1 ; PshiftDelay2 900 ; PshiftFreq2 62.228149 ; 3fshift 0; revgain1 115 ; revtime1 120 ; revout1 110 ; Pan1AZ 90, 0 3000 ; Pan2AZ -90, 0 3000
delay 3500 - ; voicevolume 0 10

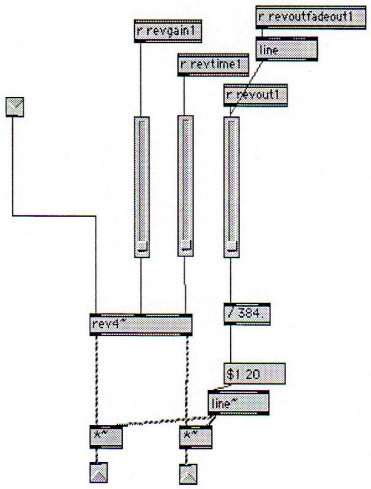
Event76 ; voicevolume 1 10 ; matrixfxin 0 1 1 ; revgain2 95 ; revtime2 120 ; revout2 85 ; Pan1AZ 180, 90 3000 ; Pan2AZ -180, -90 3000
delay 5000 - ; voicevolume 0 10

Event77 ; voicevolume 4 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 ; revgain1 125 ; revtime1 120 ; revout1 125; Spatline1 source Az 180 1 0
delay 700 - ; voicevolume 0 10

Event78 ; voicevolume 2 10 ; matrixfxin 0 0 1 ; matrixfxout 0 0 1 1 1 1 2 9 1 ; revgain1 115 ; revtime1 120 ; revout1 115; Spatline1 source Az 0 1 0
delay 1000 - ; percuvolume 1 10 ; matrixfxin 1 9 1 2 1 1 ; revgain2 100 ; revtime2 100 ; revout2 115 ; Spatline2 source Az 0 0 0 1 : Spatline2 source Az 180 1 0 4000
delay 2000 - ; voicevolume 0 10
delay 9000 - ; percuvolume 1, 0 300

Event79 ; percuvolume 1 10 ; matrixfxin 1 0 1 ; revgain1 110 ; revtime1 120 ; revout1 106 ; Spatline1 source Az 180 1 0
delay 8000 - ; percuvolume 0 10

Next, I would like to explain nine effects individually with examples; *reverb, trans/delay, vocoder, sträing, filter, fft-xover, harm2, xsynthx* and *filtrage* (see Example 12) and the *spat~* object to control the diffusion of sounds in real space.



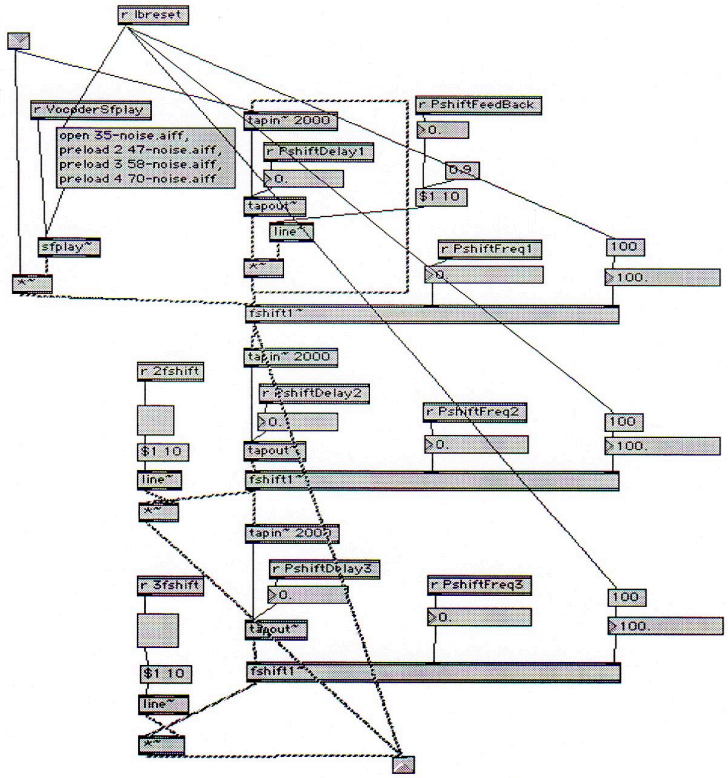
Example 16: *p reverbs1 & 2*

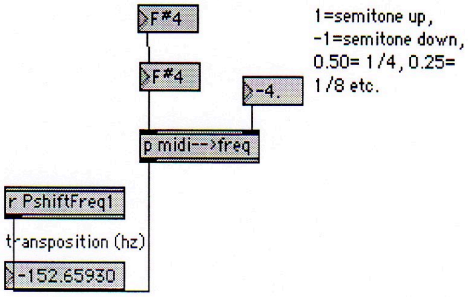
The *rev4~* object is to add a reverberation. Input (also called an inlet) parameters for *reverb gain (revgain)*, *reverb time (revtime)* and *output level (revout*)are normalized from 0 to 127.

There are two *reverb*s in my Concerpatch. When one of them is an infinite *reverb,* I use the other one at the same time. When *revtime* is 127, it becomes an infinite *reverb.* I sometimes control the volume of the end of the *reverb* with *revoutfadeout* especially, after the infinite *reverb.* Otherwise, if I stop it all of a sudden, its sound is sometimes distorted. For

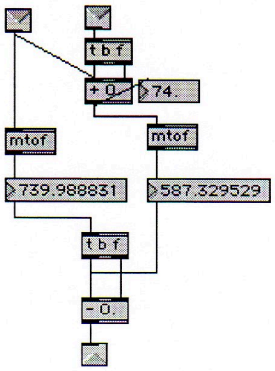
example, look at the events 13, 14 and 15 in Example 15. Here, both *reverb*s are used as infinite *reverb*s. At event 15, both are stopped with fadeout. The parameter of *revoutfadeout1* 106, 0 300 means that in 300 milliseconds, 106 of *revout* becomes 0. I also reset *matrixfxout* 0 0 0 1 1 0 and *revgain* 0, *revtime* 0, *revout* 0 to make sure to stop both *reverb*s. It is very important for me because I use the infinite *reverb* to make a chord. I made also a little change in a parameter of *reverb* each time to make a subtle timbre modification. In this piece, the *reverb* has a very important role. You can find a detail in *p score* of Example 15.

Example 17a: *p transp/delay*





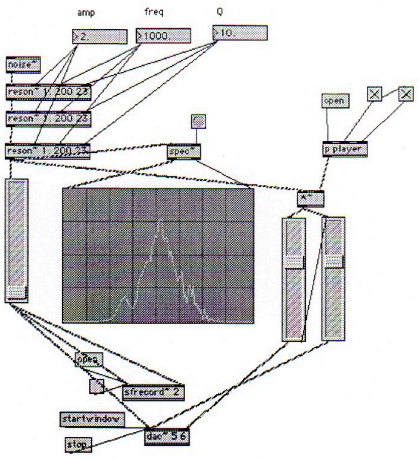
Example 17b



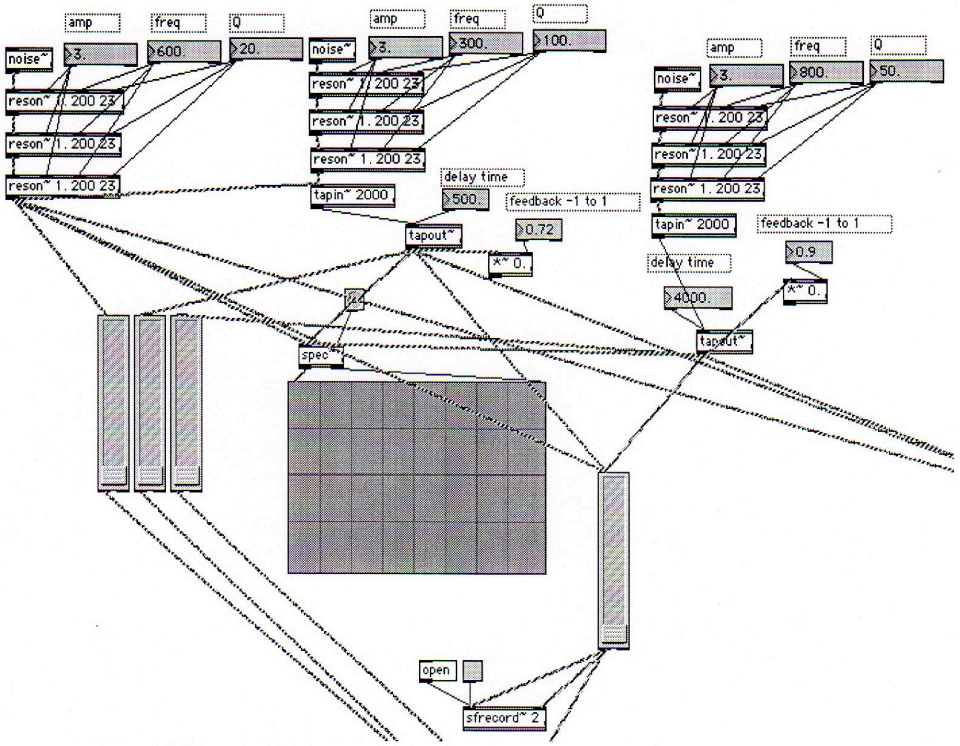
Example 17c: *p midi--->freq*

Example 17a is a subpatch of *transposition/delay*. The *fshift1~* object is a simple frequency shifter. In the left input, it sets a signal-in. In the middle input, it sets a reference frequency in hertz, which means a transposition. In the right input, it sets a portamento time in milliseconds. Because I wanted to have a specific pitch for a transposition each time in my music, I needed to convert the pitch from Midi to Hertz, using the object *mtof*. For example, Examples 17b and 17c have parameters for event 3. The original pitch from the voice is F#4-G4. I wanted to have D4- E4 and A4-B4 with delay (milliseconds). To have the first delayed transposition from F#4 to D4, which is 4 semitones down, using *mtof* to convert to hertz, -152.659302 hertz is used for *fshift1~*. When I have the second delayed transposition, as in event 3, *2fshift* should switch on, which is written with 1 in *p event* of Example 17. If it is 0, the *toggle* object is blank. Only event 28 has three transpositions with delay respectively. In this case, both *2fshift* and *3fshift* are on.

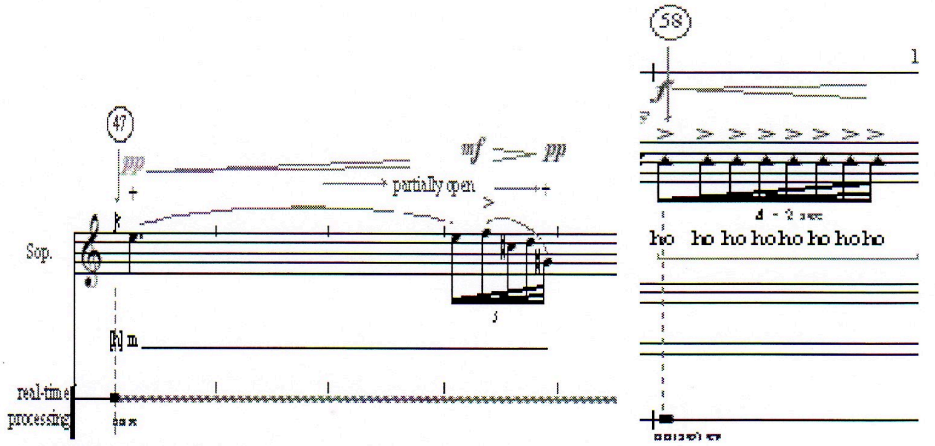
Example 17a is also used for ring modulation with filtered noise. The sounds of the filtered noise were recorded in advance to use in the Concertpatch. The soundfiles of 35, 47, 58, and 70 noise.aiff, in Example 17a, are created by the object *reson~,* which is a bandpass filter. (The numbers 35, 47, 58 and 70 refer to the event numbers.) 35 and 47 noise.aiff are generated by Example 18b, in which three different filtered noises are used. 58 and 70 noise.aiff are made by Example 18a. The signal of the object *noise~*, a white noise generator, is filtered with *reson~*. The left-middle input is set for the bandpass filter gain, the right-middle input for the bandpass filter center frequency in hertz and the right input for the bandpass filter Q. Q is a ratio of a bandpass filter's center frequency to its bandwidth. It means that the higher Q values indicate a narrower bandwidth. It should be between 0.01 and 500. I wanted to have windy and breathy sounds with voiceless Japanese phonemes, [fu] (see Example 4), [hm] and [ho] (see Example 19) at these events.



Example 18a: filtered noise ring modulation

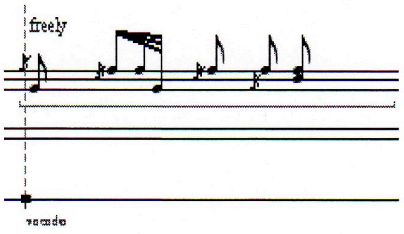


Example 18b: three different filtered noises

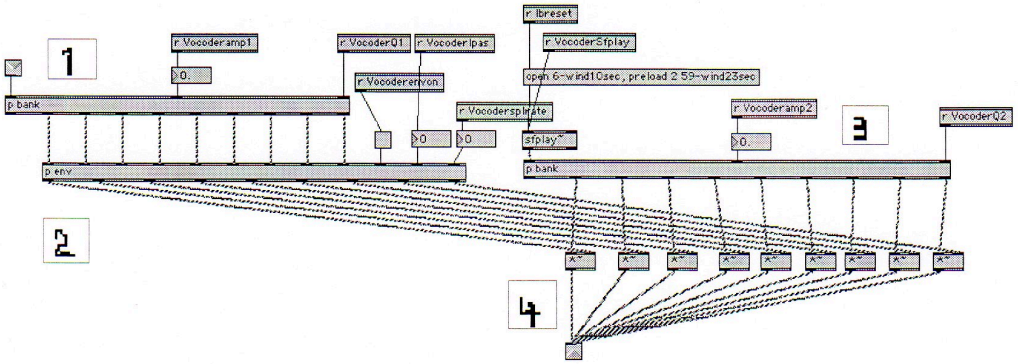


Example 19: section 2, event 47 and section 3, event 58

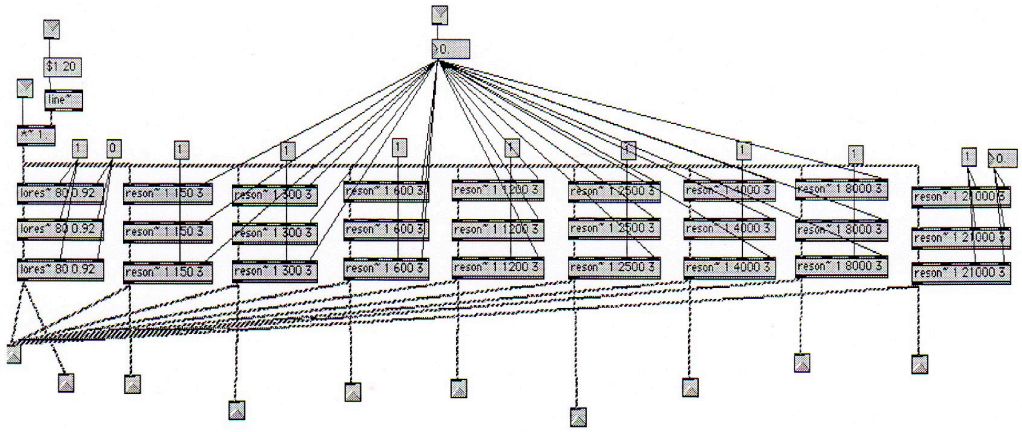
I use *vocoder* twice; at events 6 (Example 3a) for voice and 59 (Example 20) for Japanese wind bells. *Vocoder* stands for Voice Operated reCOrDER. It is a signal processor, considered a family of sound analysis/synthesis systems that imposes the amplitude envelopes of one input control signal upon a second input program signal. I would like to explain how it works in my Concertpatch (Example 21a).  at Example 21a in *p bank* (Example 21b) splits the sound-in into frequency bands. The object *lores~* is a resonant lowpass filter. The middle input is set for the lowpass filter cutoff frequency, the right input for the resonant factor between 0 (minimum resonance) and 1 (maximum resonance). The *reson~* object is explained already in Example 18.  at Example 21a in *p env* (see Example 21c) follows the envelope of each frequency bands.  at Example 21a in *p bank2* (see Example 21d) passes soundfiles I recorded in advance through the same bank of filters with different parameters. The sound of wind is used as a soundfile. The parameters I have in Example 21d are used for event 6.  at Example 21a uses the first sound envelopes to control the amplitude of the second sound's filter output. For example, at event 6, the envelopes of the vocal figure with [m] will be controlled by the amplitude of the filtered sound of wind. If you use a white noise-like sound for the second sound, this effect works quite well. All parameters for *vocoder* of events 6 and 59 are written in Example 17: *p event.*



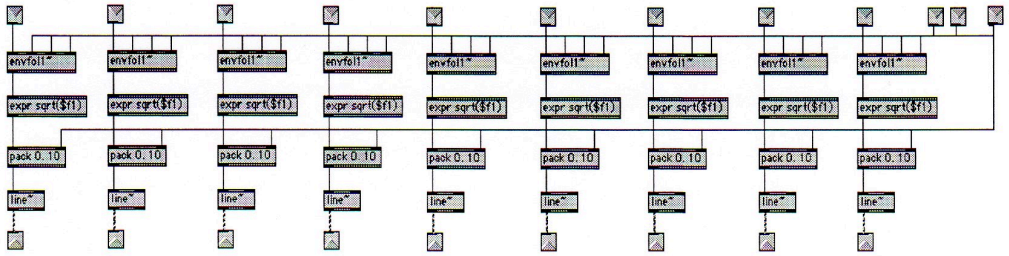
Example 20: section3, event 59



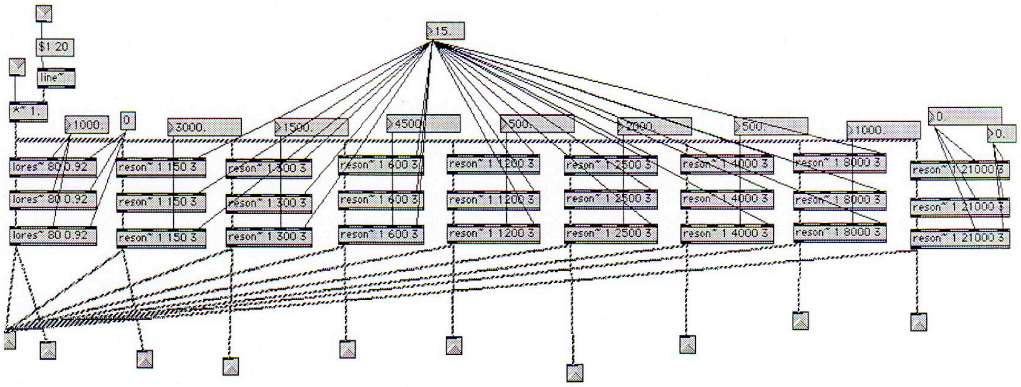
Example 21a: *p vocoder*



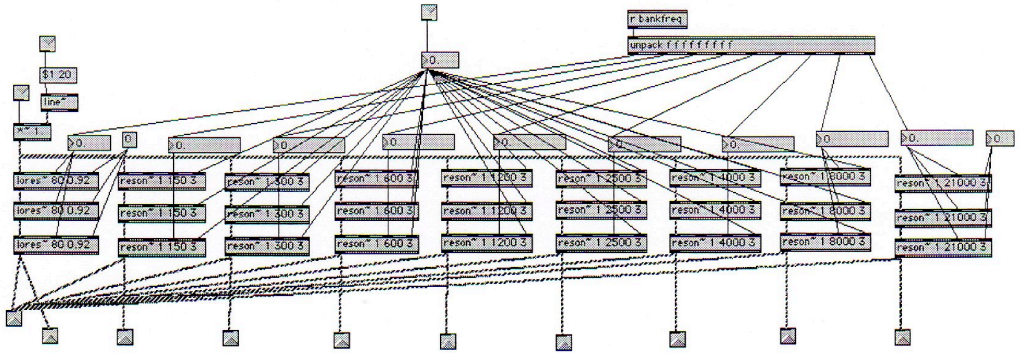
Example 21b*: p bank*



Example 21c: *p env*



Example 21d: *p bank2* at event



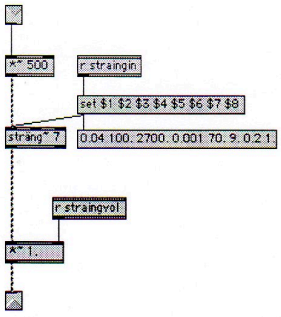
Example 21d: *p bank2*

The *sträng~* object at Example 22a is a creation from a Modalys mono-string with force connection. (We have to define how to connect different objects with Modalys. Principal connections include *adhere, bow, force, hole, pluck, position, reed, reed-free and strike.*) Modalys, one of the softwares developed at IRCAM, is used for creating virtual instruments based on physical properties such as strings, metal plates, tubes, membranes, plectra, reeds, and hammers. We construct an instrument and define how it will be played. I also use Modalys in the electronic part in this piece. I will talk about it more in Section 3.

I use *sträng~* twice, at events 21 for stones and 57 for a voice. The left input is to set seven message arguments: length (meters), tension (newtons), density (kg/m3), radius (millimeters), young, freq-loss, const-loss. These seven parameters and straing volume at event 21 are set in *p INITS* at Example 9. The middle and right inputs are for force-in and point-output locations (default number are 0.3 and 0.6). Example 22b is the patch how I found the parameters at event 21. There are seven materials: aluminium, silver (Argent), wood (Bois), chromium (Chrome), copper (Cuivre), quartz, and glass (Verre) inside a subpatch of *MaterialPara* (Example 22d). At event 21, I use aluminium and chrome and then at event 57, quartz and chromium. Example 22c is a subpatch of *CordeParam.*

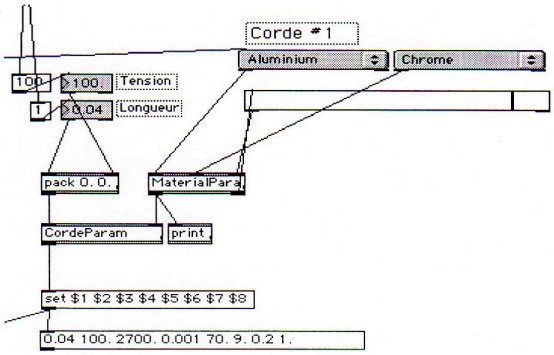
My first idea for event 21 (Example 22e) was to change the sound of stones gradually to metallic sound by using *sträng~.* However, according to a model of a mono-string, if the material is changed gradually, the pitch will be changed, making it difficult to control the pitch. My solution was using *sträng~* and *filtergraph~* in real time to create metallic sounds. Then both sounds are moved circularly in opposite directions, using *spat~* (page 38). (I will explain how speakers work later in this section.) These sounds are merged with breathy and white noise-like sounds of the voice. Finally these sounds reach to the sound of *suikinkutsu* (I will mention it in Section 3) in the electronic part. As I mentioned about the microphone control for *sträng~* on page 10, the volume should be really turned down to avoid distortion and feedback. I needed to spend time to find right volumes for the input and output both on the mixing board and on Max/Msp. This effect is very sensitive to the volume.

Event 57 is for voice. From a technical point of view, the parameters for *sträng* are set at event 56 because the effect of *filter* are still lasting and *p control* for *sträng* is off. You will find them in detail at Example 17.

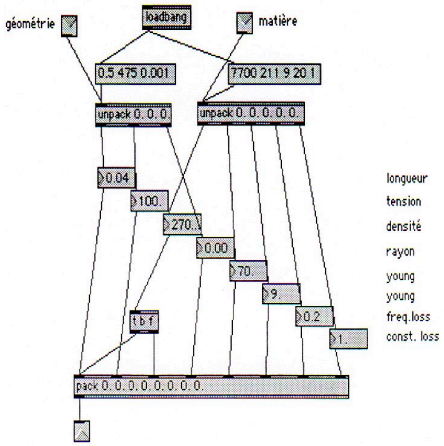


Example 22a: *p straing*

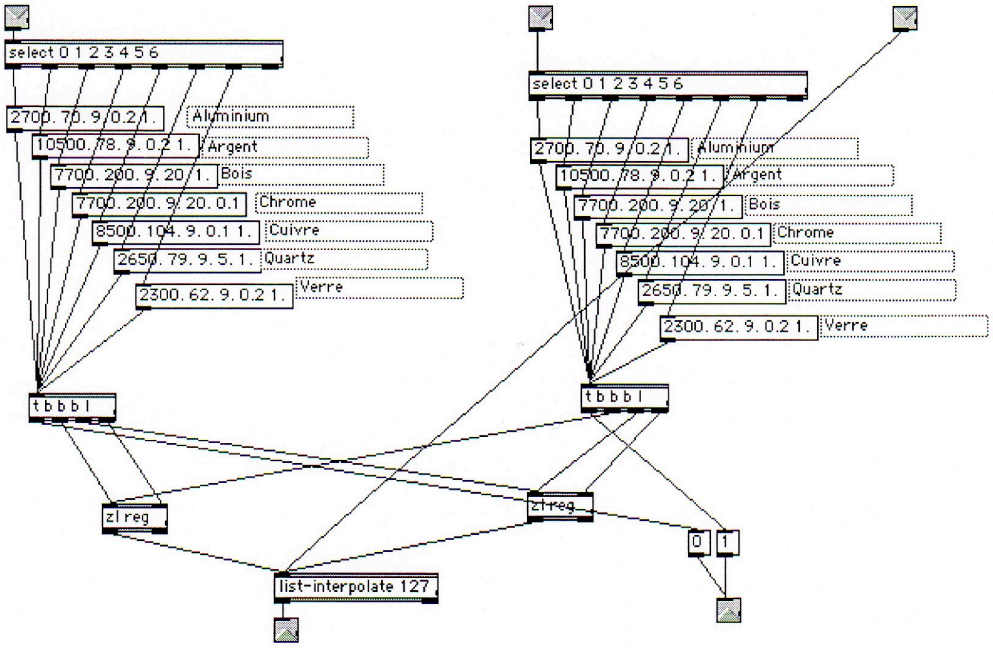
Exanple 22b

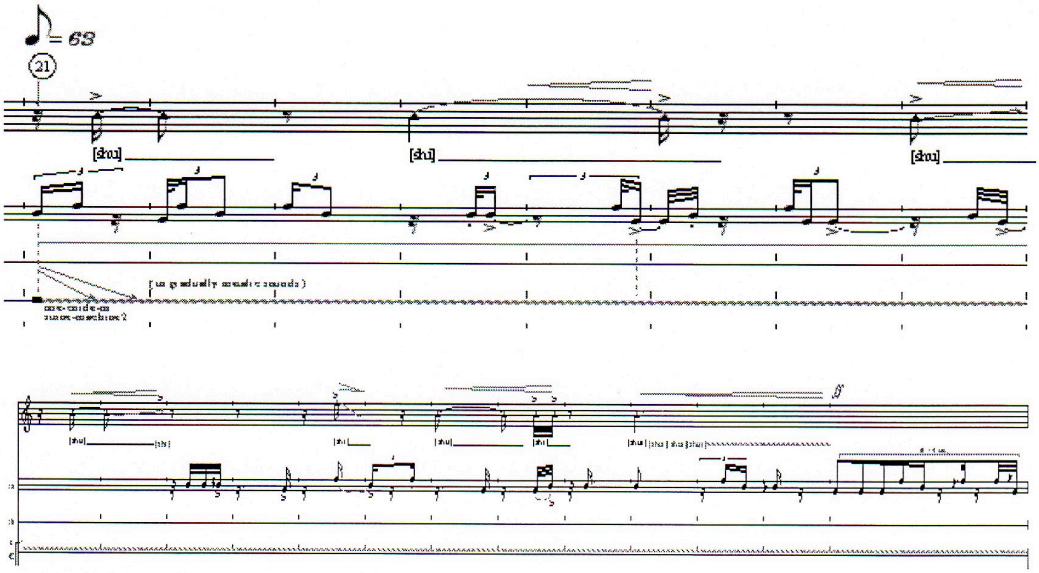


Example 22c: *CordeParam*

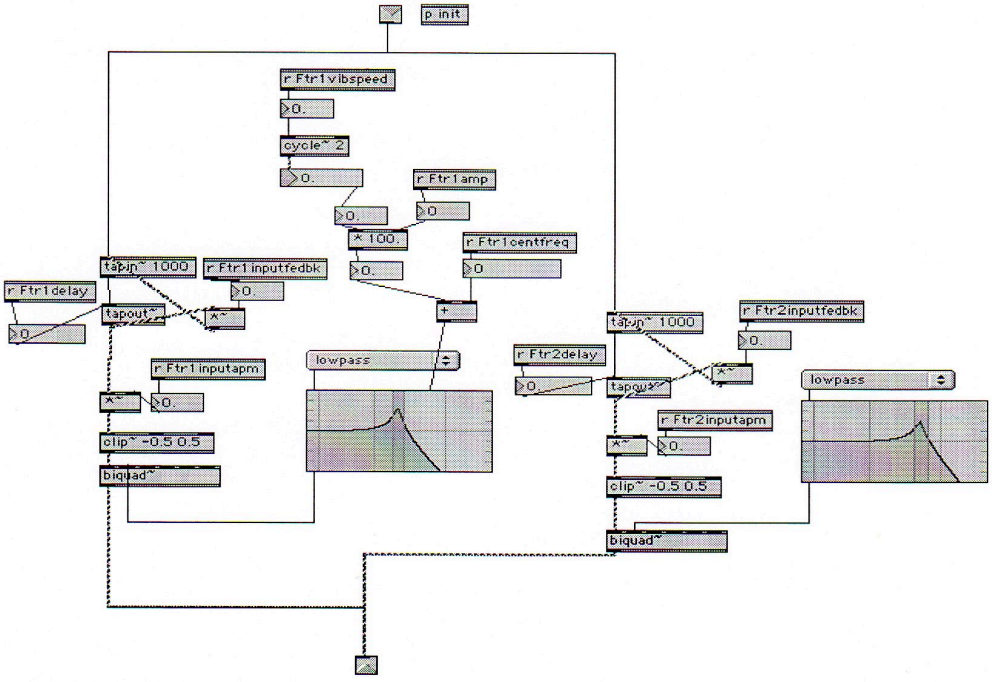


Example 22d: *MaterialPara*





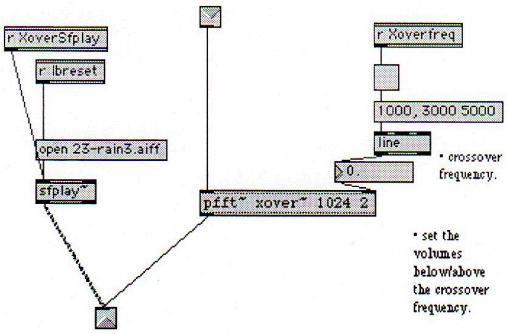
Example 22e: section1, event 21



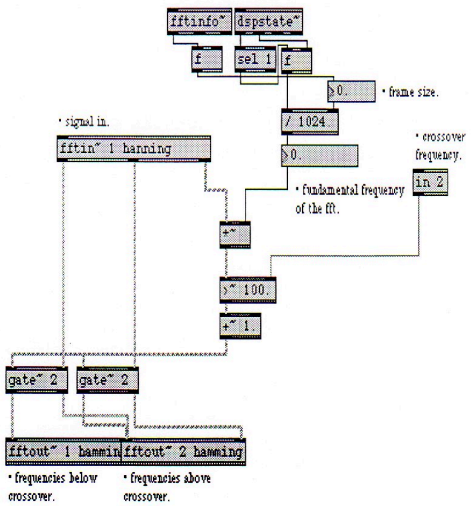
Example 23: *p filter*

Example 23 shows *p filter.* The signal is to be filtered with the *biquad~* object. The *filtergraph~* object, which is a graphical filter editor, saves its filter parameters with the patch. In the left input, there are six types of filters: lowpass, highpass, bandpass, bandstop, peaknotch, lowshelf, and highshelf. I use a lowpass filter for my music.

At event 21 (see Example 22e), as I mention above, *filter* is used for stones to create metallic sounds with *sträng~.* I use two *filtergraphs~* with delay (see Example 17: *p event*). On the other hand, at events 27, 48, 49 and 54, this effect is used to produce a sound like a muted trumpet. I use only the middle part of the patch - three parameters: center frequency, speed of vibration and amplitude at Example 23: *p filter.* You can see the vibration of center frequency on the *filtergraphs~.* It causes nasal sounds.

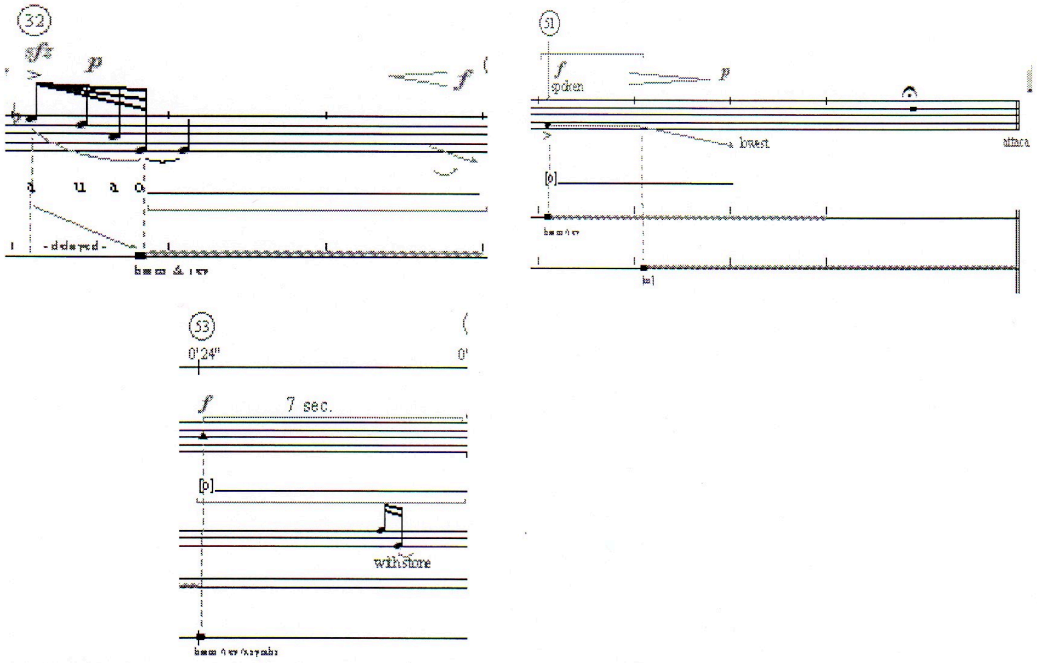


Example 24a: *p fft-xover*

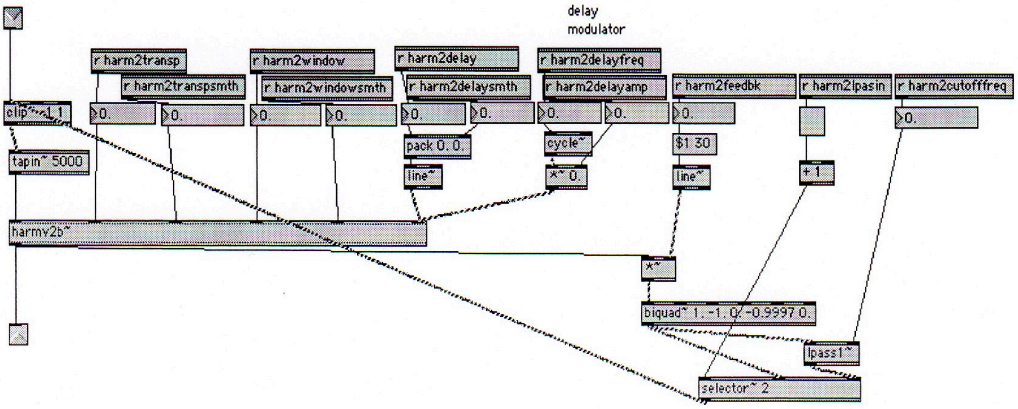


Example 24b: *pfft~ xover~*

The effect, *crossover* (see Example 24) is used only at event 23. The parameters for *Xoverfreq* (1000, 3000 5000) mean that the frequency changes from 1000 hertz to 3000 hertz in 5000 milliseconds (5 seconds). While it changes, the original vowel, [i] changes slightly. Here, it also works with a recorded soundfile of a rainstick and the live sound of rainstick with *reverb*. I wanted to create a bright shining sound for a high-pitched voice with a rainstick as the end of the first section, after the sound of stones and electronics become metallic in color.

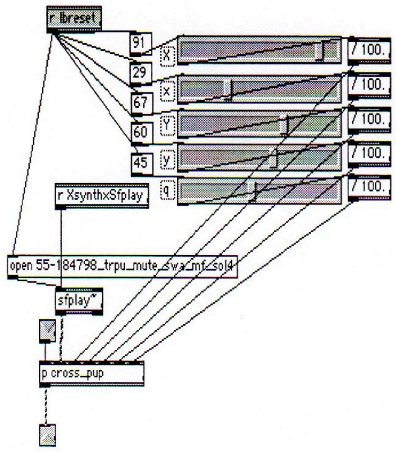


Example 25a: section2, events 32 & 51 and sefction3, event 53

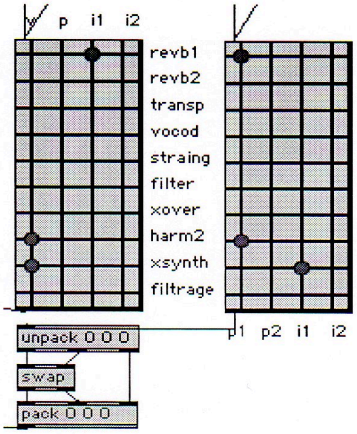


Example 25b: *p harm2*

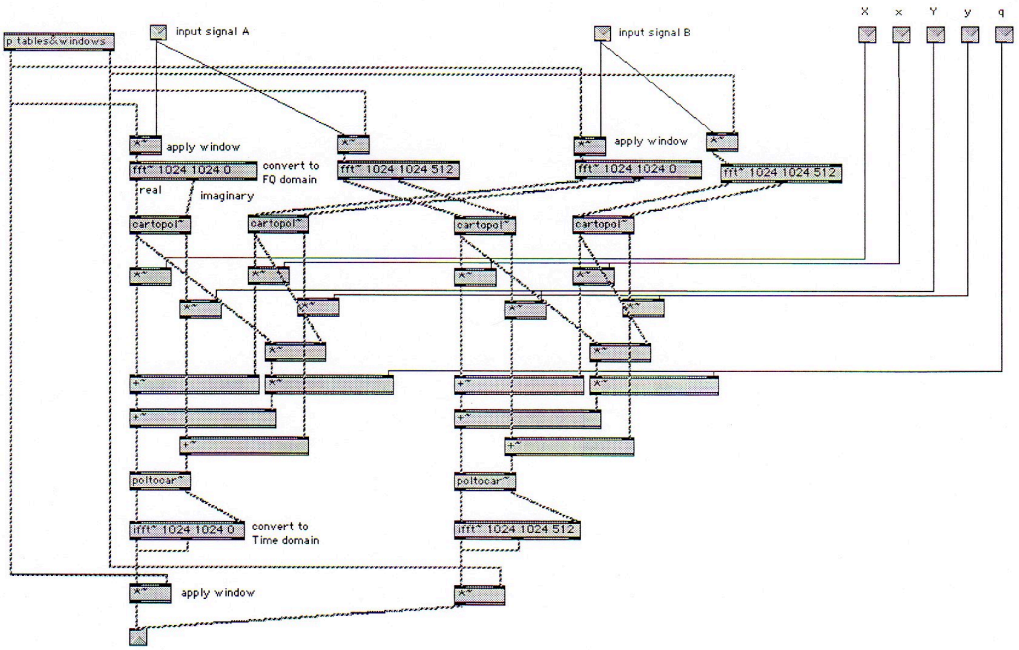
Example 25b shows *harmonizer* , which shifts a pitch in real time without changing its duration. Unlike *fshift1~* , *harmv2b~* works for transposition in midicents. I use it at events 32, 51 and 53 (see Example 25a), where there are glissandi for voice. The *harmv2b~* object is modified *harmv2~* to allow for changing of delay time and window size without clicks. The second to the left input sets for a transposition in midicents, the middle left for transposition smoothing in milliseconds, the middle right for window size in milliseconds, and the second to the right for window size smoothing in milliseconds. Transposition smoothing causes the glissando effect. Window size should be between 50 and 100 milliseconds. If it is below 50 milliseconds, there is no transposition. The right inlet sets for delay time in milliseconds and delay time smoothing in milliseconds. Delay modulator generates a vibrato effect. Each parameter can be found in Example 17. This effect produces colorful rich glissandi with voice.



Example 26a: *p xsynthx*



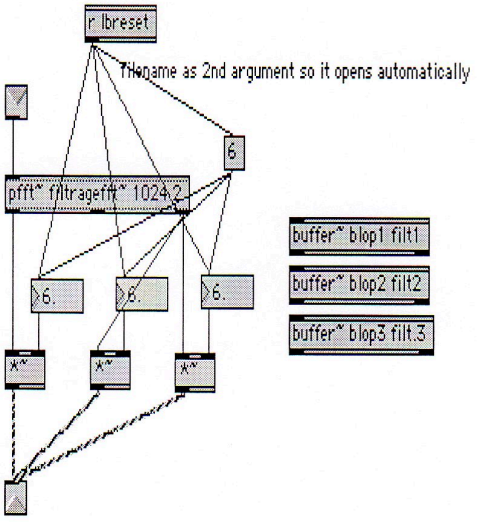
Example 26b: matrix for section3, event 53



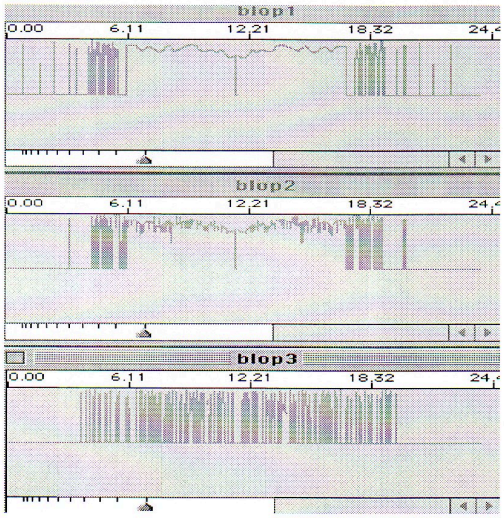
Example 26c: *p cross-pup*

I use *cross synthesis* at event 53 (see Example 26a). Input signal A is a

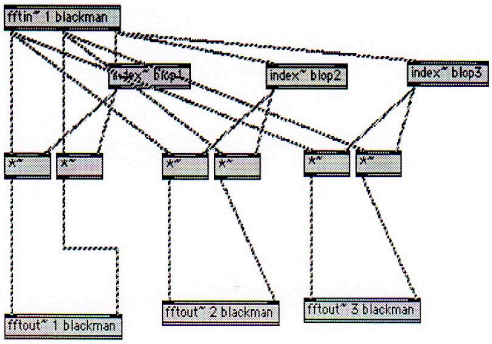
glissando by voice. Input signal B is a sample of trumpet with a wa-wa mute on G3 because the voice starts approximately on G3. Here I use the the sound of muted trumpet (signal B) to modify the characteristics of the voice. X is an amplitude for voice and x for trumpet. Y is a phase for voice and y for trumpet (see Examples 26a and c). As Example 26a shows, I found the parameters to produce a muted trumpet-like vocal sound. The *fft~* object executes a FFT (Fast Fourier Transform) on a signal. Arguments are fft window size in samples, fft period (number of samples between ffts) and phase. Besides, at event 53, the voice sends to *xsynth* and adds revb1 as well as sends to *harm2* (see Example 26b).



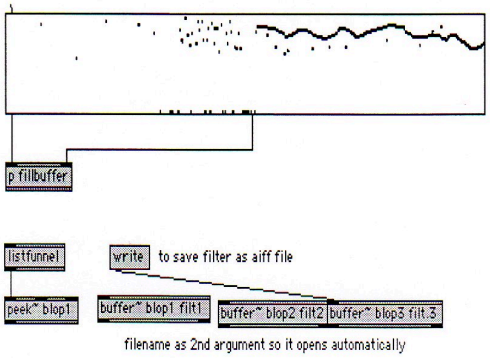
Example 27a: *p filtrage*



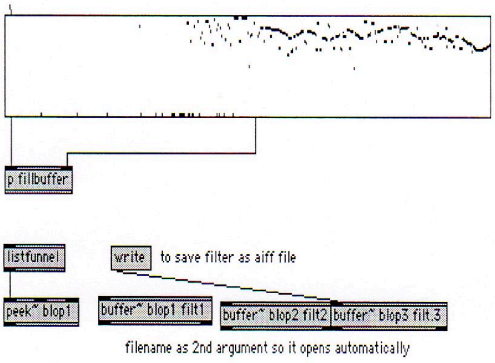
Example 27b: *buffer~*



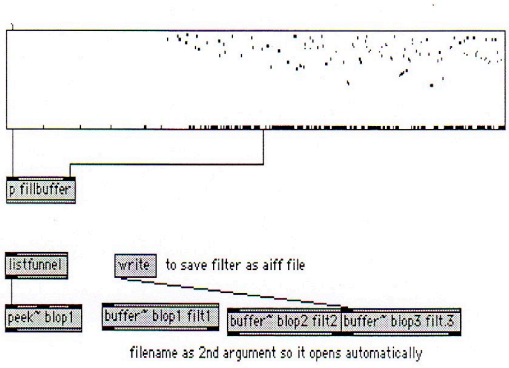
Example 27c: *pfft~ filtragefft~*



Example 27d: filt1

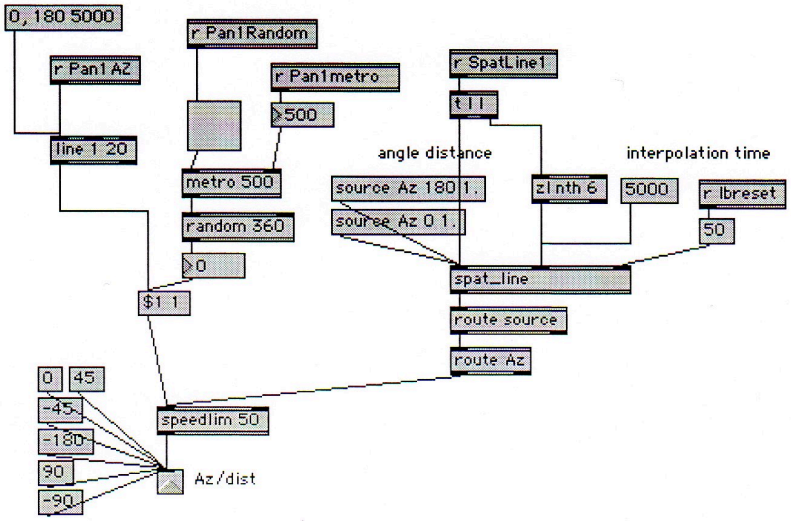


Example 27d: filt2



: filt3

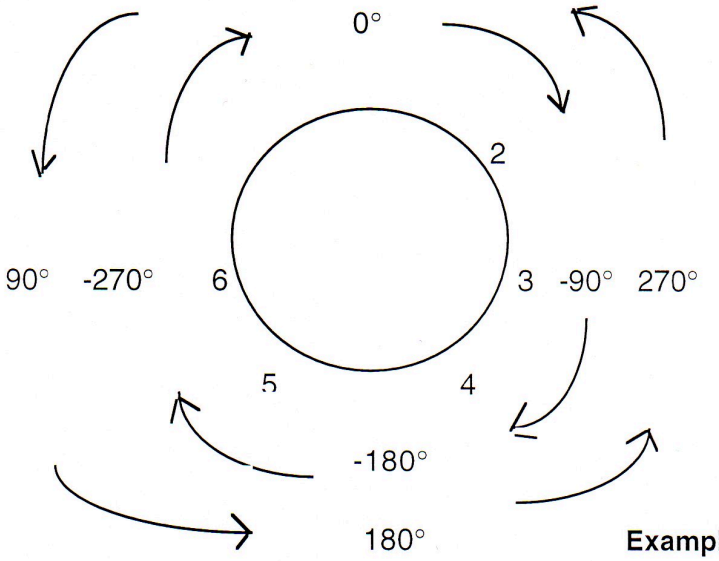
The effect, *p filtrage* is used for a Japanese temple bowl as rubbed with a triangle beater at events 56 and 78. Filt1, filt2 and filt3 are saved as aiff files in advance (see Example 27d). Three filters work at the same time to produce a very high frequency.



Example 28: *p pan1, 2 control*

The *spat~* object is a real-time spatial processing of sounds. It controls the localization of sound sources in three dimentions and the reverberation of sounds in a real or virtual space. The *panrN~* object is N-channel panpot for reproduction of all directions in the upper hemisphere, using a multi-channel horizontal speaker setup. I do not use the *spat~* object but *panrN~* to reduce CPU utilization since ten real time effects and thirty sound files use a large percentage of CPU. I use six speakers in real time with the 2 *panr6~* objects. This means I sometimes control two different diffusions of the same sound or different sounds at the same time. There is a rule for a speaker setup. Speaker numbering increases clockwise starting from the first speaker at the left of front. Azimuth increases counterclockwise starting from the front center (see Example 29). There are three ways, *pan1AZ, pan1Random* and *spatline1* to define the diffusions at *p pan1,2* in

my Concertpatch (see Example 28). For example, at event 21 there are Pan1AZ 0, -450 20000 and Pan2AZ 0, 450 20000. This means that starting from 0°, the sound of stones with *sträng* moves counterclockwise



Example 29: 6 speakers setup

at -450° and the one with *filter* moves clockwise at 450° in 20000 milliseconds (20 seconds). Next one is a random diffusion of a sound. At event4, *Pan2Random* is on and *Pan2metro* is 700. It signifies that there is a random diffusion of the sound from six speakers during 700 milliseconds. The last one is SpatLine1. The *Spat_line* object generates 3-D linear interpolation. The arguments are x (azimuth in degrees), y (distance in meters) and z (elevation in meters) for a source localization and interpolation time in milliseconds. Spatline source Az 0 1 0 means that the sound diffuses from the front, Spatline source Az 0 0 0 or 90 0 0 from the center and Spatline source Az 180 1 0from the back. For example, at

event 6, the parameters; Spatline2 source Az 180 1 0 1000 ; Spatline2 source Az 0 1 0 4000 cause the sound to be diffused from the back to the front in a linear direction during 4000 milliseconds. The parameters at event 38 ; Spatline1 source Az 90 1 0 1 ; Spatline1 -90 1 0 2500 cause the sound to be moved from left to right in the center line during 2500 milliseconds. The parameters at event 46 ; Spatline1 source Az 135 1 0 10 ; Spatline1 source Az -45 1 0 2000 signifies that the sound moves in a diagonal from left back to right front during 2000 milliseconds.

In addition, I made two versions of a CD recording. One is a stereo version. The other one is a transaural reproduction with three dimensional mode on two channels, which can be processed with *spat~* modules. The listener should sit, facing two speakers in an equilateral triangle position.

The next example (Example 30) is a summary for soundfile, effects and speakers setting for each event, which I used for a quick check during the rehersals. Event numbers in boldface have sound files. At the first event in brackets, parameters of *matrixfxout* are written.

Example 30: summary for each event

SECTION 1

Event 1 - rev. for tongue click -	speakers - spatline1 (0 1 0)
Event 2 - SF1 -	speakers 1, 2 + 7
Event 3 - SF2 delayed 600ms -	speakers 7 - 1, 2 + 3, 4, 5, 6
- fshift for voice delayed 1100ms -	speakers - spatline1 (180 1 0 1) (90 0 0 2000)
Event 4 - rev. for voice -	speakers - (0 1 0)

- rev. for rainstick delayed 1000ms - speakers - pan2random 700
Event 5 - rev. stop
Event 6 - vocoder - speakers - spatline1 (0 1 0)
- vocoder w/rev. for voice - spatline2 (180 1 0 1000) (0 1 0 4000)
Event 7 - SF3 delayed 600ms - speakers 8 + 1, 2
Event 8 - SF4 - speakers 1, 2, 8
- rev. stop
Event 9 - SF5 - speakers 3, 4, 5, 6
Event 10 - SF6 delayed 1100ms - speakers 3, 4, 5, 6 -- 1, 2, 3, 4
Event 11 - rev1 for voice - speakers - spatline1 (180 1 0)
Event 12 - inf. rev2 for voice - speakers - spatline2 (0 0 0)
Event 13 - inf. rev1 for voice - speakers - spatline1 (0 1 0)
- delay/ feedback for voice, using fshift - speakers - pan2random 800
Event 14 - SF7 delayed 1200ms - speakers - 8---7
- inf. rev1 for voice delayed 500ms - spatline1 (180 1 0)
Event 15 - rev. stop
Event 16 - SF8 delayed 950ms - speakers 1, 2, 3, 4
Event 17 - SF9 - speakers 3, 4, 5, 6 + 7, 8
Event 18 - feedback for voice delayed 3000ms, using fshift - speakers - spatline1 (0 1 0 1) (180 1 0 5000)
Event 19 - rev. for rainstick - speakers - spatline1 (180 1 0 1) (0 1 0 3000)
Event 20 - SF10 - speakers 1, 2, 5, 6 + 8
Event 21 - sträng for stones - speakers - pan1 (0, -450 20000)
- stone machine2 for stones using fshift/rev.- pan2 (0, 450 20000)
Event 22 - fshift/ rev. for voice - speakers - spatline1 (90 1 0 1) (180 1 0 7000)
Event 23 - fft02 for voice, using xover / rev. w/rain3.aiff - speakers - pan1 (0, 360 7000) pan2 (0, -360 7000)

SECTION 2

Event 24 - SF11 - speakers 5, 6, 8 ---1, 2, 3, 4
Event 25 - rev. for voice - speakers - spatline1 (0 1 0 1) (0 0 0 10)
Event 26 - rev. for voice - speakers - spatline1 (0 1 0 1) (180 1 0)
- rev. for voice delayed 500ms - spatline2 (0 1 0)
Event 27 - lowpass filt./ feedback for voice, using trsp. ftr. speakers -- pan1random 300
Event 28 - delay/ fshift/ rev. for voice - speakers - spatline2 (0 1 0)
Event 29 - SF12 - speakers 1,2 --5, 6, 7
Event 30 - rev. for voice - speakers - spatline1 (0 0 0)
Event 31 - delay/ fshift / rev. for voice - speakers - spatline1 (180 1 0)
Event 32 - harm/ rev. for voice delayed 1100ms - speakers - pan1 (0, 180 1500), pan2 (0, -180 1500)
Event 33 - SF13 - speakers 8 ---1, 2
Event 34 - SF14 - speakers 1, 2, 8
Event 35 - filter noise RM for voice, using trsp. - speakers - pan1 (0, 180 1500), pan2 (0, -180 1500)
Event 36 - rev. for voice - speakers - spatline1 (0 0 0)
Event 37 - SF15 - speakers 8 + 3, 4

Event 38 - rev. for voice - speakers - spatline1 (90 1 0 1) (-90 1 0 2500)
spatline2 (-90 1 0 1) (90 1 0 2500)
Evebt 39 - SF16 - speakers 5, 6 --8
Event 40 - SF17 - speakers 1, 2
- inf. rev1 for voice - speakers - spatline1 (0 1 0)
Event 41 - inf. rev1. for voice delayed 2000ms -
speakers - spatline2 (180 0 0 1) (0 1 0 1000)
- SF18 delayed 2500ms - speakers 3, 4
Event 42 - fshift/ inf. rev2 for voice - speakers - spatline2 (180 0 0 1) (0 1 0 1000)
Event 43 - SF19 - speakers 5, 6 --- 8
- stop inf. rev.
Event 44 - fshift/ rev. for voice - speakers - spatline1 (0 0 0)
Event 45 - SF20 - speakers 1, 2, 7
Event 46 - rev. for voice - speakers - spatline1 (135 1 0 10) (-45 1 0 2000)
spatline2 (-135 1 0 10) (45 1 0 2000)
Event 47 - SF21 - speakers 8
- noise/rev. for voice, using trsp. - speakers - spatline1 (0 0 0)
Event 48 - SF22 - speakers 5, 6 + 1, 2
- mute/rev. for voice, using filter -
speakers - pan1 (90 450 3000), pan2 (-90 -450 3000)
Event 49 - mute for voice, using filter - speakers - spatline1 (90 1 0), spatline2 (-90 1 0)
Event 50 - rev. for voice - speakers - spatline1 (0 1 0)
Event 51 - SF23 delayed 1000ms - speakers 8 + 3, 4
- harm/rev. for voice - speakers - spatline2 (180 1 0 1) (0 0 0 2000)

SECTION 3

Event 52 - SF24 - speakers 3, 4
- SF25 delayed 14sec. - speakers 7, 8
Event 53 - harm/ rev. for voice - speakers - spatline1 (0 1 0 1) (180 1 0 2000)
- xsynths/ rev. for voice w/ muted trumpet - spatline1 (0 1 0 1) (180 1 0 2000)
Event 54 - filter/ rev. for voice - speakers - spatline2 (0 1 0 1) (0 0 0 800)
Event 55 - delay/ fshift/ rev. for voice - speakers - spatline1 (0 0 0)
Event 56 - filtrage for JtbL - speakers - spatline1 (0 1 0) (180 1 0 30000)
Event 57 - sträng/ rev. - speakers pan2random 200
Event 58 - noise/ rev. for voice, using trsp. e/ noise.aiff -
speakers -pan1 (90, -90 1000), pan2 (-90, 90 1000)
Event 59 - vocoder/ rev. for Jwb. - speakers - pan1 (45, 180 1000), pan2 (-45, -180 1000)
Event 60 - inf. rev1 for voice - speakers - spatline1 (0 0 0)
Event 61 - inf. rev2 for voice - speakers - spatline2 (0 0 0)
- SF26 - speakers 1, 2 + 7
Event 62 - SF27 - speakers 3, 4, 8
- rev. stop
Event 63 - rev. for JtbL - speakers - spatline1 (0 1 0 1) (180 1 0 15000)
Event 64 - trsp./rev. for JtbL - speakers - spatline2 (0 0 0)
Event 65 - rev. for voice - speakers - spatline1 (0 0 0)

Event 66 - trsp./rev. for voice - speakers - spatline2 (0 1 0)
Event 67 - trsp./rev. for voice - speakers - spatline1 (180 1 0)
Event 68 - rev. for voice - speakers - spatline2 (0 0 0)
Event 69 - SF28 - speakers 1, 2, 5, 6
Event 70 - noise/rev. for voice, using trsp. w/noise.aiff -
speakers - pan1 (90, 180 2000), pan2 (-90, -180 2000)
Event 71 - SF29 - speakers 1, 2, 7, 8
Event 72 - inf. rev1 for voice - speakers - spatline1 (180 1 0 1) (0 1 0 15000)
Event 73 - inf. rev2 for voice - speakers - spatline2 (0 0 0)
- SF30 delayed 2000ms - speakers 1, 2, 3, 4, 5, 6
Event 74 - rev. stop
Event 75 - trsp./ rev. for voice - speakers - pan1 (90, 0 3000), pan2 (-90, 0 3000)
Event 76 - rev. for voice - speakers - pan1 (180, 90 3000), pan2 (-180, -90 3000)
Event 77 - rev. for voice - speakers - spatline1 (180 1 0)
Event 78 - rev. for voice - speakers - spatline1 (0 1 0)
- filtrage/ rev. for JtbL delayed 1000ms - spatline2 (0 0 0 1) (180 1 0 4000)
Event 79 - rev. for voice - speakers - spatline1 (180 1 0)

* rev. = reverberation, inf.= infinite, trsp.= transposition, SF= sound file, JtbL = Japanese temple bowl (L)

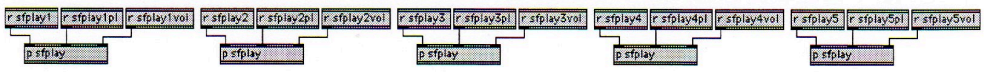
SECTION 3

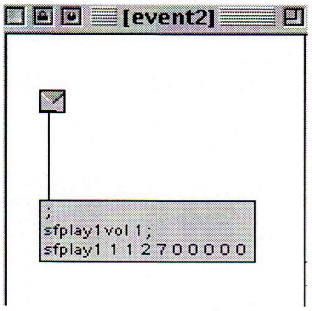
"ELECTRONICS"

Configuration

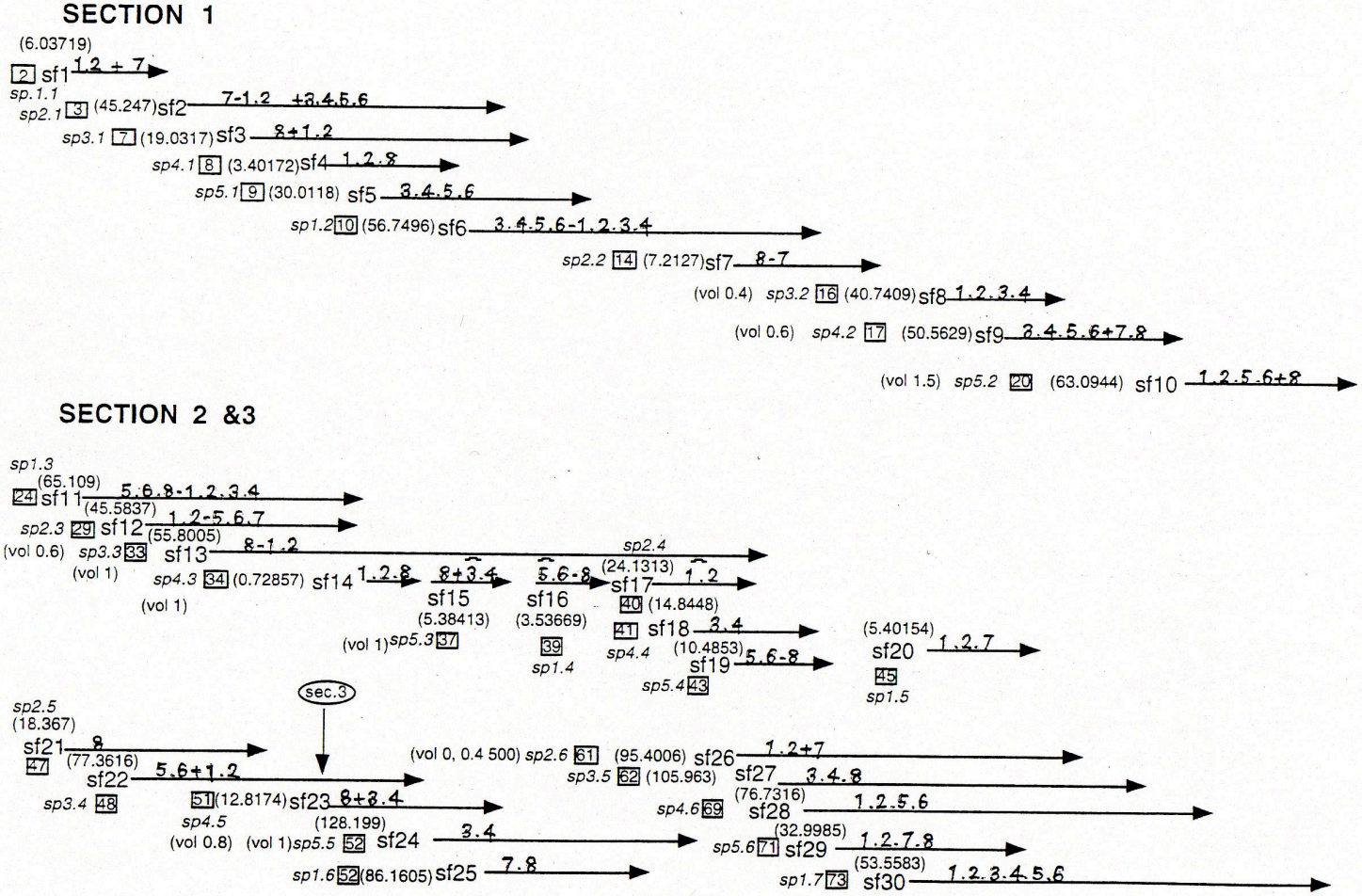
There are thirty soundfiles which are distributed into five *p sfplay*s (Example 31), because five soundfiles are superimposed at most in the ending of the piece. At Example 9: *p INITS,* all of the names of soundfiles are initialized in the respective *sfplay*. The parameters for volume and speaker settings for soundfiles are written in Example 11: *p score.* For example, Example 32 shows parameteres for event 2. It means that *sfplay* 1, preload 1, speakers 1, 2 and 7 are set up. In terms of speakers, as I mentioned at page 10, I use eight speakers for electronics. The *sfplay~* 8 object for eight channels is used to play soundfiles. The rest of the five speakers should be written with 0 0 0 0 0 to be figured out in the Concertpatch. Each soundfile is bounced for each speaker on Protools and then merged as one soundfile with multichannel by Sound File Splitter & Merger in advance (see the speakers setting in Example 33). These multichannel files are read by Max/Msp.

Example 31: *p soundsplayer*





Example 32: section1, event 2



Example 33

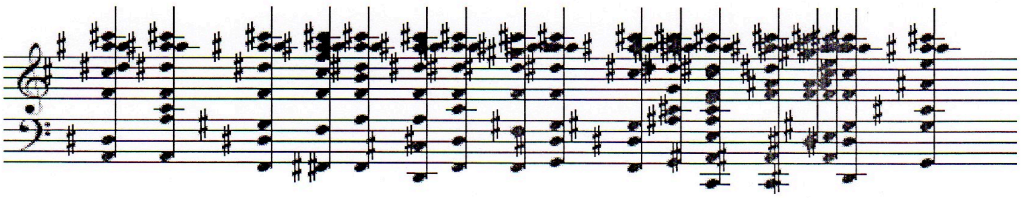
* sp = sfplay, sf = soundfile, square number = event number, ( ) = duration in sec. , numbers on the arrow = speakers setting, ~ on the spekers= panning

Pre-recorded Materials

Before starting to create electronic sounds, I used a recording with a singer to make materials for electronics. They are shown below. The first three materials are the result of spectral analysis, using AudioSculpt and OpenMusic for suikinkutsu*, stone and Japanese temple bowls. I recorded the pitch of each of them with voice and used them with chords in the electronic part. I also recorded the singer reciting the three texts naturally and the inscription pitched about G3 without accentuation. The inscription (see Example 49) is recorded with three separated parts to use in the piece respectively.



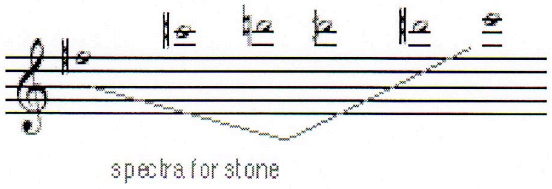
Material 1: recorded with [i] in 5 seconds (used in section1)



chord sequence from *suikinkutsu* 's spectral analysis by Audiosculpt and OpenMusic

* *Suikinkutsu* is built in conjunction with a water basin underground. Water flows underground into a ceramic echo chamber filled with water. The sound of water resonates, is amplified, and is reflected back to the garden above. It is a Japanese way of feeling joy with sounds in the garden. The

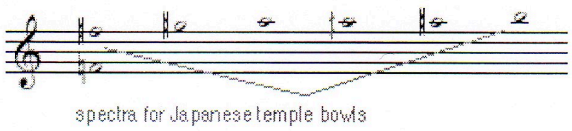
sounds depend on the amount of water, shape, material, and size of the chamber with timing and spacial balance. I also use the sample sound of *suikinkutsu*, transformed in timbre and pitch by AudioSculpt in the electronic part of section 1.



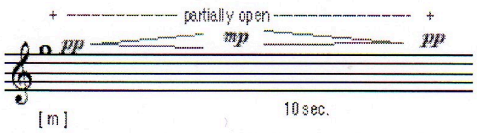
Material 2: recorded with [a] [i] [e] in 2 seconds and with staccato (used in section1)



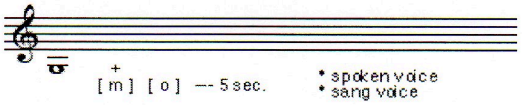
chord sequence from stone's spectral analysis



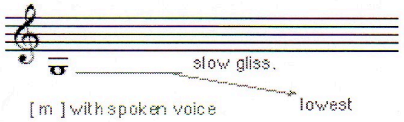
Material 3: recorded with [o] in 5 seconds (used in section3)



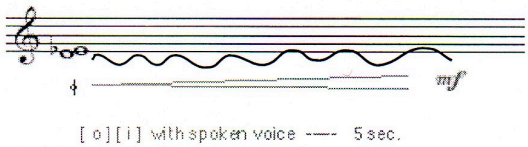
Material 4: (used in section3)



Material 5: (used in section3)



Material 6: (used in section3)



Material 7: (used in section2)

Material 8: voiceless sounds

[tsu], [chi], [hi], [fu], [ku], [ki], [sh], [shi], [r] --short, 2sec., and 5sec. sounds sec1, 2

Material 9: voice effect

[ha], [ya], [iya] -- short, 2sec.

tongue click at random

gliss up/down with phonemes [o], [m], [a], [r]

Material 10: tremolo and trill within 5sec.

D4-D4, D4-Eb4, D4-E4, D4-F4, D4-F#4

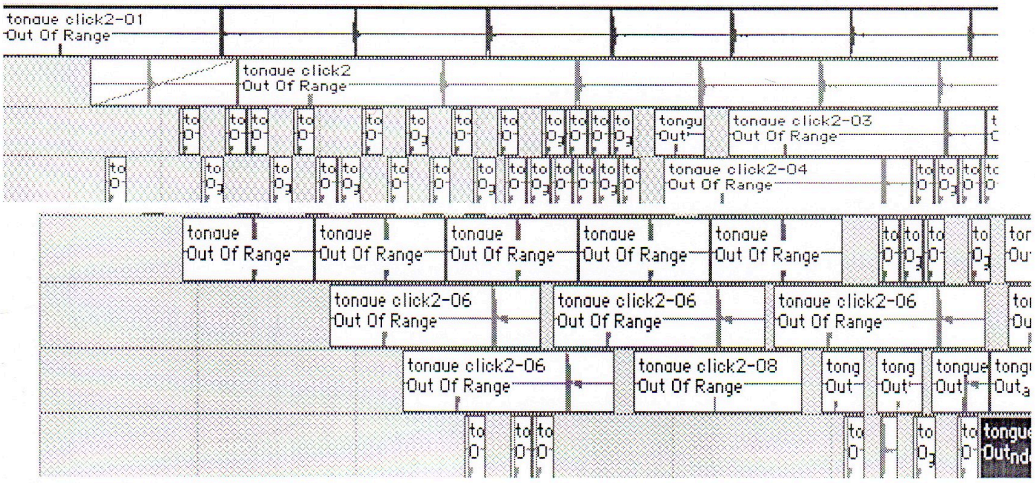
G3-G3, G3-Ab3, G3-A3, G3-Bb3, G3-B3

G4-G4, G4-Ab4, G4-A4, G4-Bb4

Electronics Contents

Each soundfile uses several treatments by different software. I explain here in detail the main treatment for thirty soundfiles (SF1 to SF30).

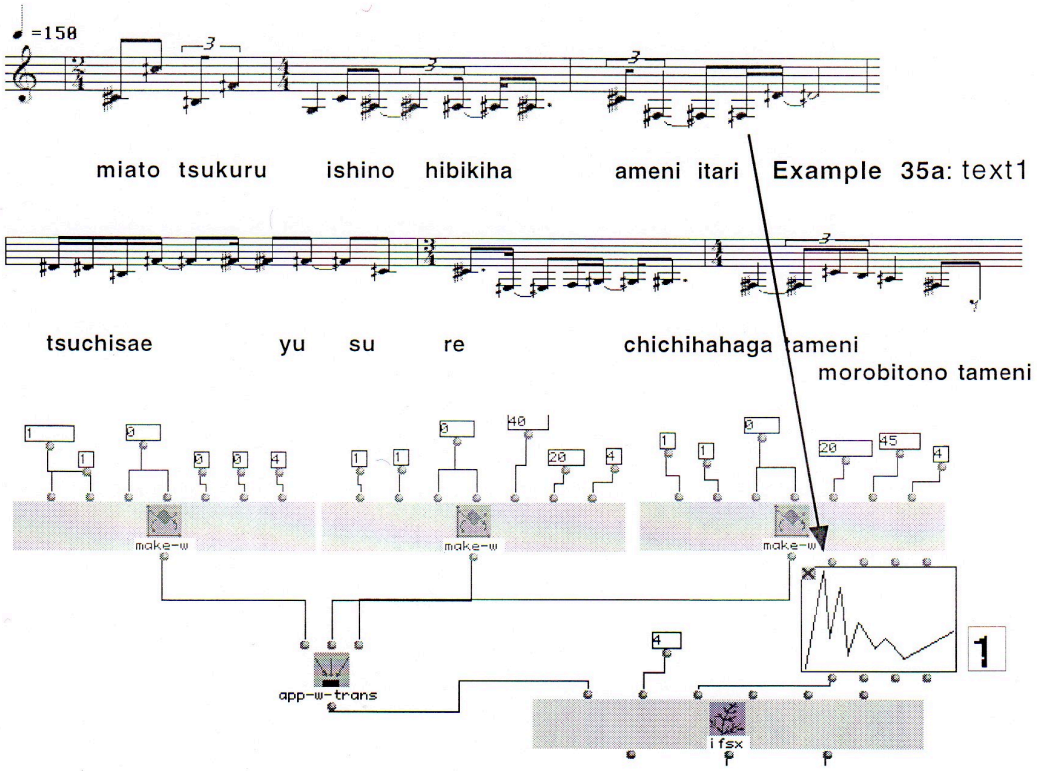
SF1 is the sound of a tongue click. It changes the pitch on Melodyne. The last pitch is F#, which is the same as the first pitch of a voice. Here is the Protools session showing how I used the sounds of tongue click.



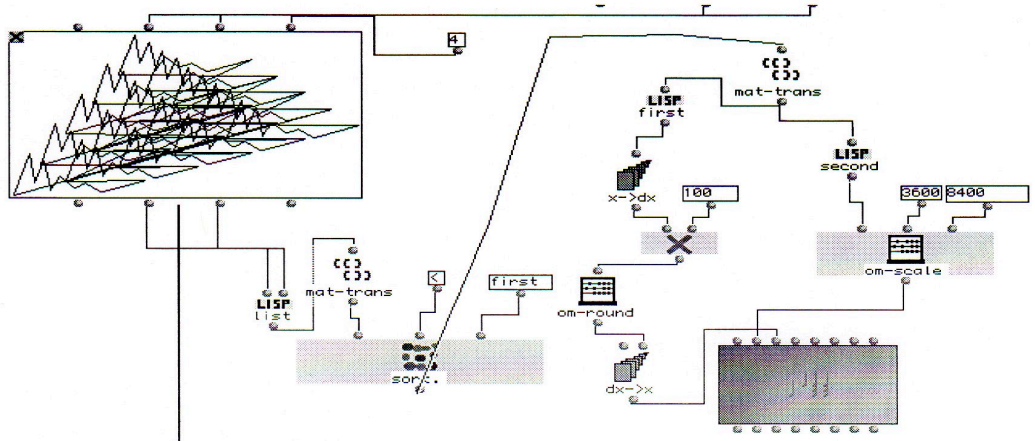
Example 34: SF1

At first, I analyzed the pitch and rhythm of text1 (Example 35a), in which I recorded the singer's reciting by AudioSculpt and Kant/OpenMusic (library for rhythmic quantification and segmentation) for SF2. I then generated the contour of this melody on Chaos/OpenMusic (library of chaos theory functions). As an example, I wrote in the BPF window in Example 35b, following the contour of the first half of the text1 in Example 35a. The three-

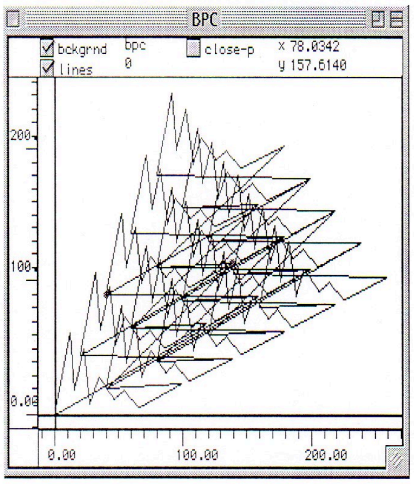
-dimensional shape was created by the parameters inside the square boxes, using the contour of the text in Example 35c. The pitch material was generated by the shape, using pitches between 3600 and 6500 midicents. Example 35d is the other example of shapes I used. Example 35e is used for the second half of the contour of the text. After I created the pitch materials, I saved them as midi to use on Max/Msp. Example 35f is a patch of *samplor~* with Max/Msp. I transformed the sound of stones with pitch material I created by Chaos/OM. Example 35g is an overview of the SF2 on Protools. Originally, the sounds of stones and rainstick were used here.



Example 35a: text1

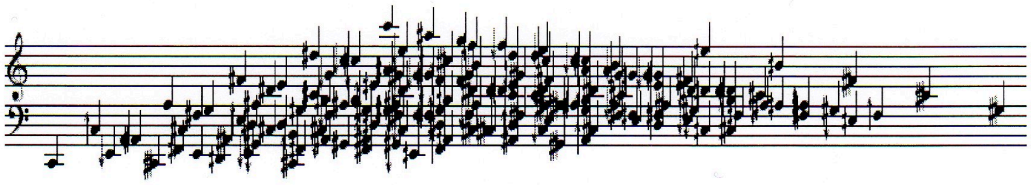


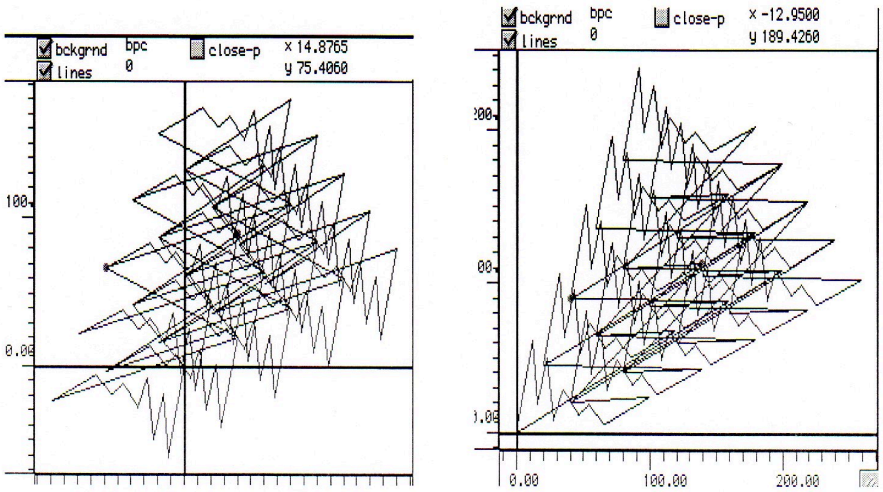
Example 35b: Chaos/OpenMusic



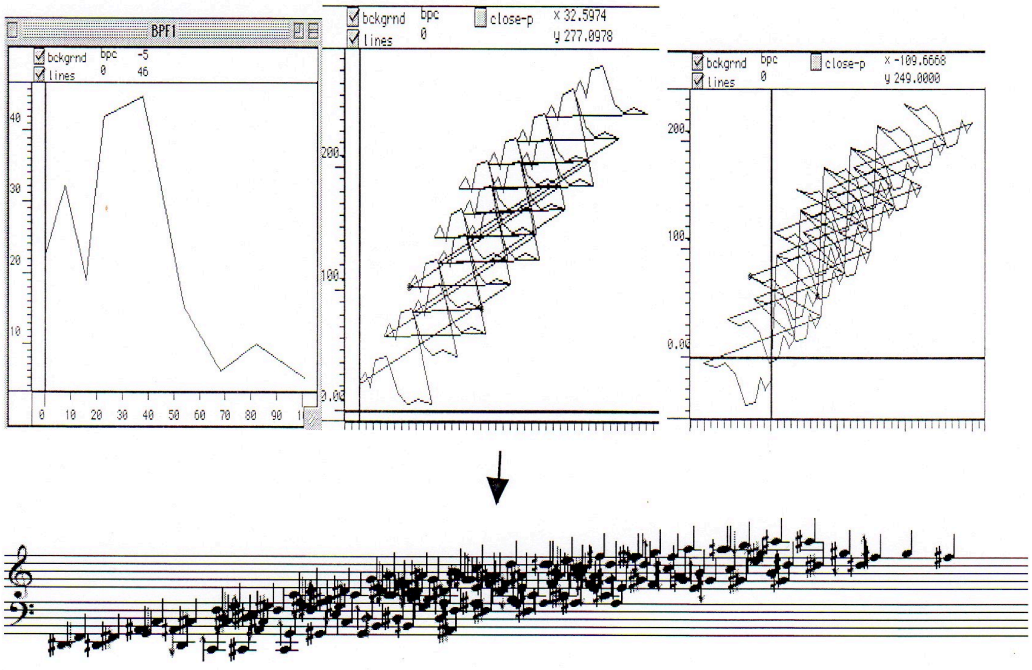
Example 35c: bpf window (above)

: pitch material (below)

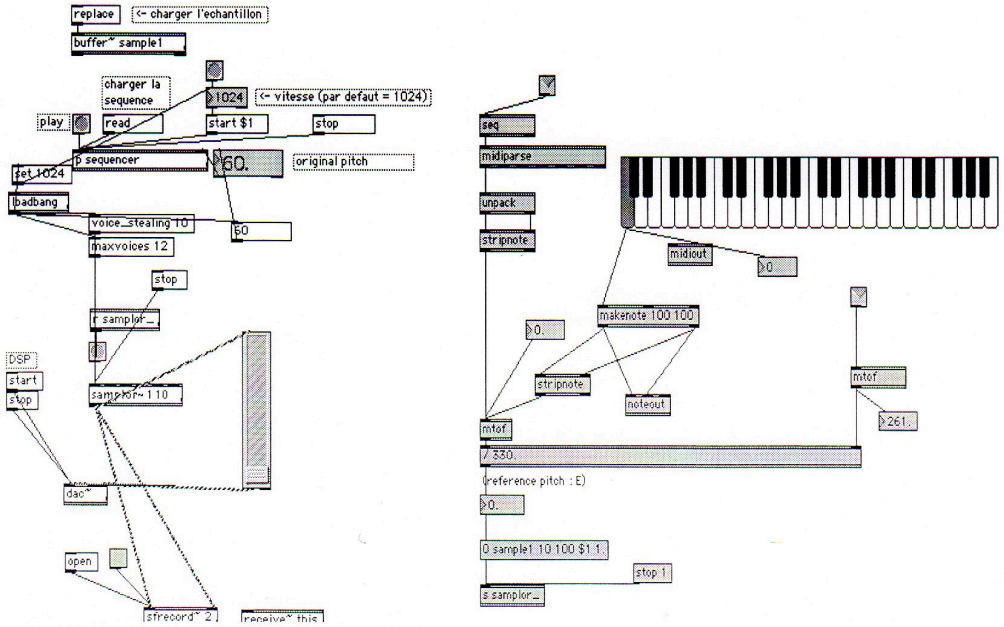




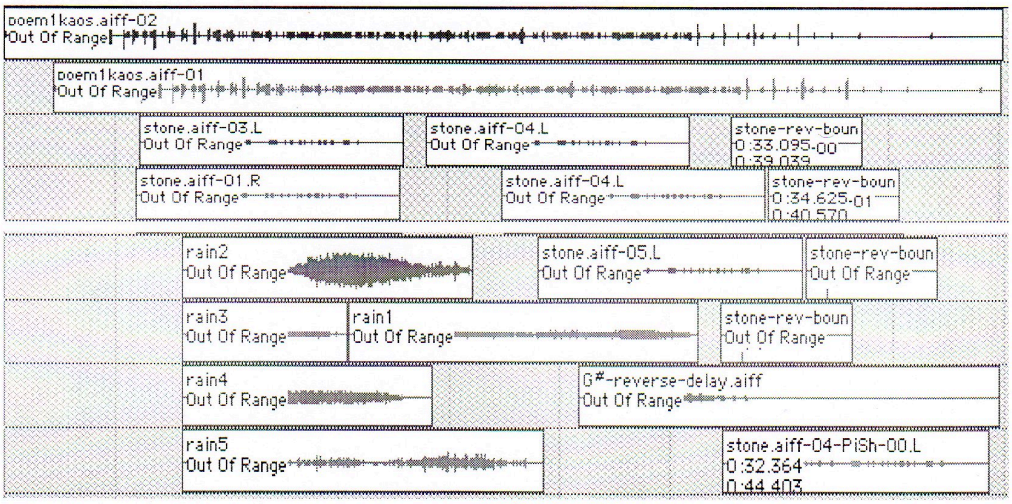
Example 35d: bpf window



Example 35e: bpf window and pitch material used for the second half of the contour of the text1



Example 35f: *samplor~* with Max/Msp

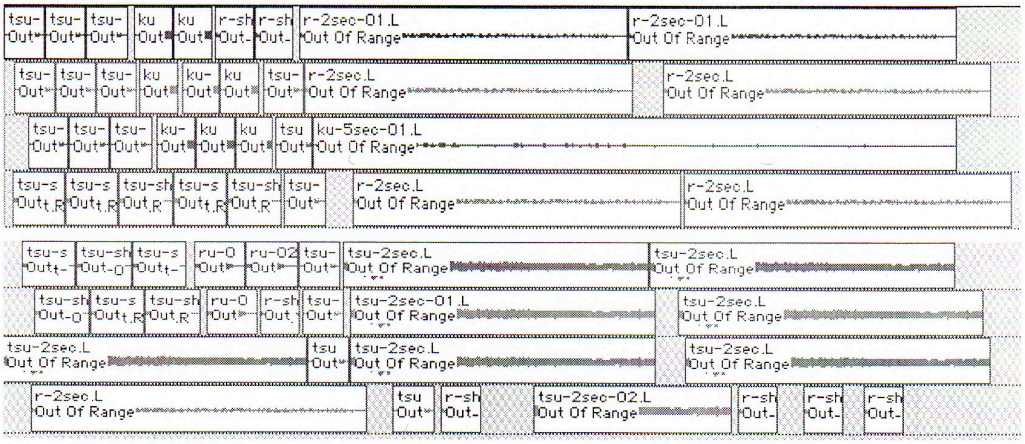


Example 35g: SF2

SF3 is the sample of G#-reverse-delay.aiff in Example 35g. I separated it to be triggered at event 7, which has the similar figure with pitch C for a

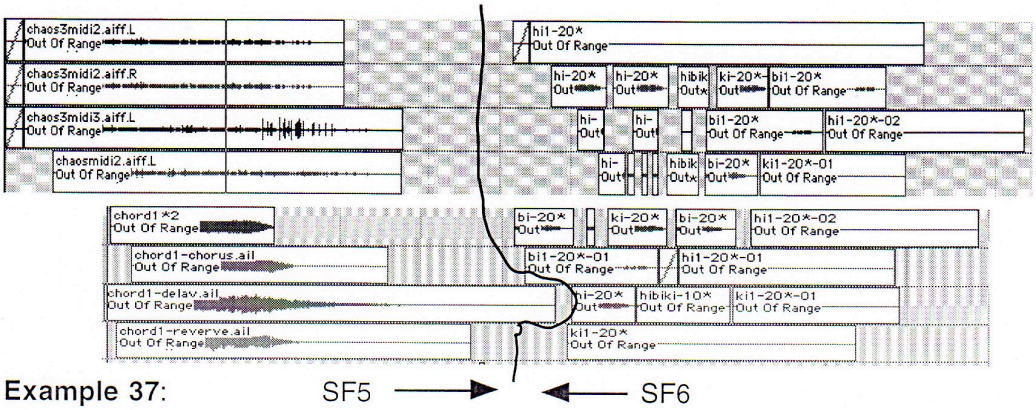
vocal part.

SF4 uses voiceless sounds: [tsu], [ku], [ru] which are the phonemes from the text (see Example 36). The vocal part also has the the same word [tsu ku ru].



Example 36: SF4

At Example 37, the begining of the soundfiles called chaos.aiff chaos is for SF5 and the rest is for SF6. SF5 uses the contour of text2 on Chaos/Om with the sound of stones using *samplor~* with Max/Msp. I created the sound in the same way as explained on page 47. SF5 uses the spectra of stones (see page 45) with voice as a chord and phonemes [hi] [bi] [ki] from the text where the vocal part uses.

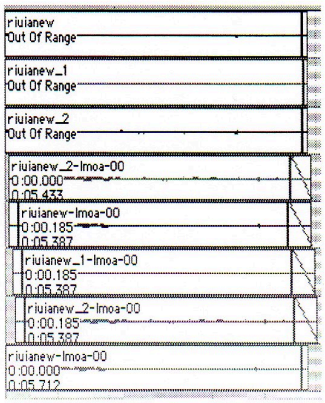


Example 37:

SF7 is created by formant filter with AudioSculpt, using the sample recording of singing [r]. Example 38a shows the formant filter parameters I used to produce vowels;[i] [u] [i]. The first column is the time at which to apply the parameters and the second one is the number of formants. The following columns are the central frequency in hertz, the gain, and the bandwidth of each formant in hertz. I use the parameters from the table at Example 40.

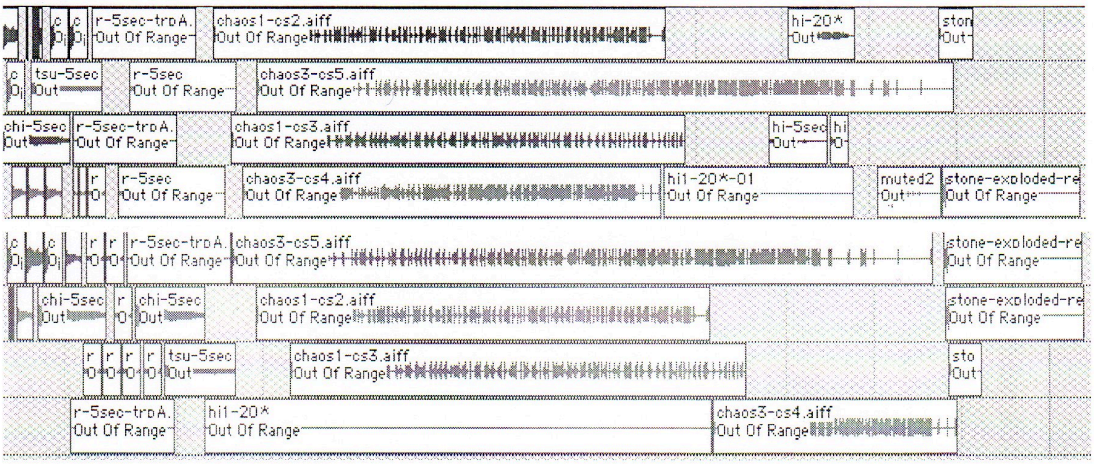
Example 38a: formant filter parameters - [i] [u] [i] (SF7)

0 5 270 0 60 2140 -10 90 2950 -20 100 3900 -20 120 4950 -30 120
1 5 270 0 60 2140 -10 90 2950 -20 100 3900 -20 120 4950 -30 120
2.5 5 325 0 50 700 -12 60 2700 -30 170 3800 -35 180 4950 -50 200
4.5 5 270 0 60 2140 -10 90 2950 -20 100 3900 -20 120 4950 -30 120



Example 38b: SF7

As Example 39 shows, SF8 and SF9 have phonemes of [tsu] [chi] [r] [hi] and sounds of Chaos/OM and stones. In the end of SF9, there is a big sound of stones, which is transformed. It is the climax of section 1.



Example 39: SF8 and SF9

Example 40: - Table of formant values for vowel sounds

TABLE :
Formant values

	f1	f2	f3	f4	f5
soprano "a"					
freq (Hz)	800	1150	2900	3900	4950
amp (dB)	0	-6	-32	-20	-50
bw (Hz)	80	90	120	130	140
soprano "e"					
freq (Hz)	350	2000	2800	3600	4950
amp (dB)	0	-20	-15	-40	-56
bw (Hz)	60	100	120	150	200
soprano "i"					
freq (Hz)	270	2140	2950	3900	4950
amp (dB)	0	-12	-26	-26	-44
bw (Hz)	60	90	100	120	120
soprano "o"					
freq (Hz)	450	800	2830	3800	4950
amp (dB)	0	-11	-22	-22	-50
bw (Hz)	70	80	100	130	135
soprano "u"					
freq (Hz)	325	700	2700	3800	4950
amp (dB)	0	-16	-35	-40	-60
bw (Hz)	50	60	170	180	200
alto "a"					
freq (Hz)	800	1150	2800	3500	4950
amp (dB)	0	-4	-20	-36	-60
bw (Hz)	80	90	120	130	140
alto "e"					
freq (Hz)	400	1600	2700	3300	4950
amp (dB)	0	-24	-30	-35	-60
bw (Hz)	60	80	120	150	200
alto "i"					

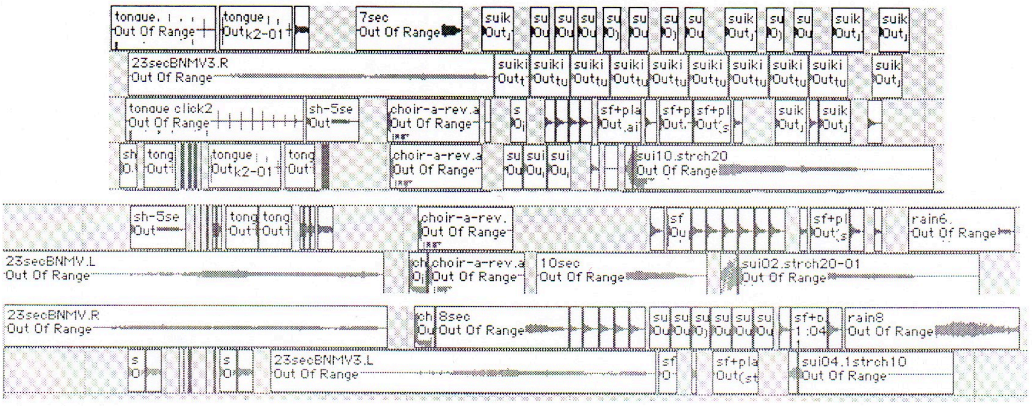
	f1	f2	f3	f4	f5
tenor "a"					
freq (Hz)	650	1080	2650	2900	3250
amp (dB)	0	-6	-7	-8	-22
bw (Hz)	80	90	120	130	140
tenor "e"					
freq (Hz)	400	1700	2600	3200	3580
amp (dB)	0	-14	-12	-14	-20
bw (Hz)	70	80	100	120	120
tenor "i"					
freq (Hz)	290	1870	2800	3250	3540
amp (dB)	0	-15	-18	-20	-30
bw (Hz)	40	90	100	120	120
tenor "o"					
freq (Hz)	400	800	2600	2800	3000
amp (dB)	0	-10	-12	-12	-26
bw (Hz)	40	80	100	120	120
tenor "u"					
freq (Hz)	350	600	2700	2900	3300
amp (dB)	0	-20	-17	-14	-26
bw (Hz)	40	60	100	120	120
bass "a"					
freq (Hz)	600	1040	2250	2450	2750
amp (dB)	0	-7	-9	-9	-20
bw (Hz)	60	70	110	120	130
bass "e"					
freq (Hz)	400	1620	2400	2800	3100
amp (dB)	0	-12	-9	-12	-18
bw (Hz)	40	80	100	120	120
bass "i"					

http://www.computermusic.ch/files/misc/Table_of_formants-values.html

freq (Hz)	350	1700	2700	3700	4950
amp (dB)	0	-20	-30	-36	-60
bw (Hz)	50	100	120	150	200
alto "o"					
freq (Hz)	450	800	2830	3500	4950
amp (dB)	0	-9	-16	-28	-55
bw (Hz)	70	80	100	130	135
alto "u"					
freq (Hz)	325	700	2530	3500	4950
amp (dB)	0	-12	-30	-40	-64
bw (Hz)	50	60	170	180	200
countertenor "a"					
freq (Hz)	660	1120	2750	3000	3350
amp (dB)	0	-6	-23	-24	-38
bw (Hz)	80	90	120	130	140
countertenor "e"					
freq (Hz)	440	1800	2700	3000	3300
amp (dB)	0	-14	-18	-20	-20
bw (Hz)	70	80	100	120	120
countertenor "i"					
freq (Hz)	270	1850	2900	3350	3590
amp (dB)	0	-24	-24	-36	-36
bw (Hz)	40	90	100	120	120
countertenor "o"					
freq (Hz)	430	820	2700	3000	3300
amp (dB)	0	-10	-26	-22	-34
bw (Hz)	40	80	100	120	120
countertenor "u"					
freq (Hz)	370	630	2750	3000	3400
amp (dB)	0	-20	-23	-30	-34
bw (Hz)	40	60	100	120	120

freq (Hz)	250	1750	2600	3050	3340
amp (dB)	0	-30	-16	-22	-28
bw (Hz)	60	90	100	120	120
bass "o"					
freq (Hz)	400	750	2400	2600	2900
amp (dB)	0	-11	-21	-20	-40
bw (Hz)	40	80	100	120	120
bass "u"					
freq (Hz)	350	600	2400	2675	2950
amp (dB)	0	-20	-32	-28	-36
bw (Hz)	40	80	100	120	120

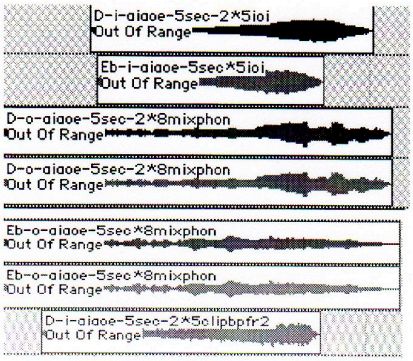
SF10 is transformed from the sounds of wind, tongue click, phonemes [shi] and [shu] and *suikinkutsu* (see page 45). The singer's voice prerecorded the pitches (see page 45) from the spectra analysis of *suikinkutsu* is added as a chord to sound like a choir.



Example 41: SF10

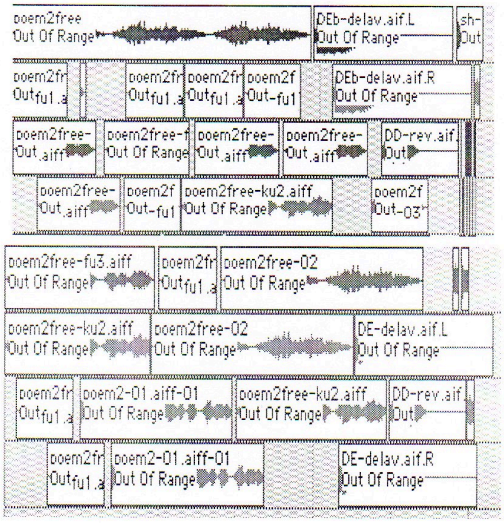
SF11 is recorded, using *vocoder* with Max/Msp for the second sound. I apply the text2 reading by the singer as a envelope (see Example 21) with a sound of wind as an amplitude. I created variation of these sounds by *vocoder.*

SF12 employs a formant filter for vowels from the text2, using the sounds of D and Eb with voice (see Example 42). The list of vowels from the text 2 is thus: i o i a a i, u a u o a a i, a o u a o, o a e u i o o, u i i a o o o o, a e i o a u a o. I adapted the parameteres from the table of formant values in Example 40.



Example 42: SF12

SF13 utilizes again a *samplor* with text2 generating by Chaos/OM and phonemes [fu] and [ku]. The samples of tremolos, [DD] and [DEb] with reverberation, filtering, time stretching etc. are applied (see Example 43).



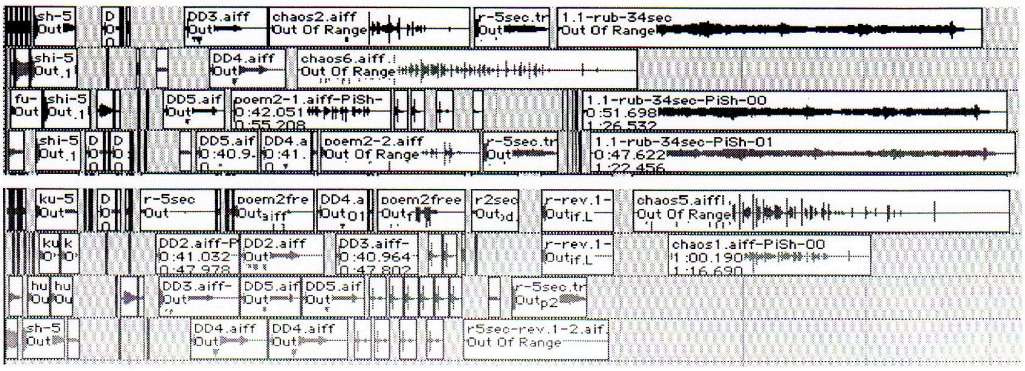
Example 43: SF13

I use a vocal effect, like Noh performers' shout, with sounds of [ya], [iya] [ha] etc. to produce a complicated rhythmic moment with instrumental sounds

and actors' movements spatially and temporally in Noh Theater at events 34, 37, 39 and 43, which refer to the SFs 14,15, 16, and 19. Each SF uses different treatments based on the samples of a recorded singer's voice such as lowpass, bandpass and formant filters along with time streching and pitch shifting. These sounds are triggered by computer to create an atmosphere like that of Noh Theater.

SF17 uses tremolos, DEb and DE by the voice. It diffuses from the back to the front speaker. It works well with the vocal part, phoneme [r] with vibrato and reverb in real time.

Example 44a shows for SF21 and 22. The original samples are phonemes; [shi] [shu] [ku] [fu] [r] and tremolos,DD by voice. I exploit the software Modalys to create metallic sounds to merge with the sound of a Japanese temple bowl in the end of section 2 and the begining of section 3. Modalys utilizes the physical modeling synthesis to produce sound. It uses the programming language called "Scheme". Example 44b is one of the texts I use to create sounds. It uses a midi file to realize the synthesis. Here I applied the midi file created by Chaos/OpenMusic to the model of the key (referred to as a blade in Modalys) of a xylophone. I also use the object of a hybrid between plate and string.



Example 44a: SFs21 and 22

; ;;-*-Scheme-*- **Example 44b**: Modalys

```
;;;----------------------------------------------------------------------
;;; Modalys, Lames de xylophone excitées par un midifile.
;;; Jean Lochard - Fev Dec 2000
;;;----------------------------------------------------------------------

(new)
(set-sample-rate 44100)
(set-midi-dir "Macintosh HD:Modalys-miyuki:")

;;;
;;; Définie une fonction qui crée une lame de xylophone à la fréquence
;;; correspondant à la note midi joué + le controleur dynamic pour pincer
;;; la lame.
;;;

(define (fabrique-lame freq)

  (define my-xylo (with-pitch-adjustment 'length freq (make-object 'rect-free-bar
    (width .05)                    ;; largeur
    (thickness .01)                ;; epaisseur
    (density 550)                  ;; densite du materiau
    (young0 1.2e10)                ;; module young en z
    (young1 1.2e9)     ;; module young en y
    (poisson .25)      ;; module de poisson
    (freq-loss 8.22e-7)            ;; pertes fonction de frequence
    (const-loss 0.2)   ;; pertes generiques
  )))


                    (define my-hammer (make-object 'bi-two-mass (small-mass .01)
                          (large-mass .01)
                          (stiffness0 150000)
                          (stiffness1 150000)))

                    ;;;
                    ;;; Définition des controleurs et des connections
                    ;;;

                    (define dyn (make-controller 'dynamic 1 0.001 (list .005)))

                      (define my-xylo-strike (make-access my-xylo (const 0.4 0.4) 'normal))


                    (define my-hammer-tip (make-access my-hammer (const 1) 'trans0))
          (define my-hammer-top (make-access my-hammer (const 0) 'trans0))
```

```
                    (make-connection 'pluck my-xylo-strike my-hammer-tip 0 .005 (const 10))

            (make-connection 'position my-hammer-top dyn)

                    ;;;
                    ;;; make output
                    ;;;

                    (define my-xylo-out (make-access my-xylo (const .3 .2) 'normal))
                    (make-point-output my-xylo-out)

;;; fin fonction fabrique-lame et renvoie du controleur dyn

dyn)

;;;
;;; Déclaration d'une fonction pour pincer la lame de xylophone
;;;

(define (start-env stk-env)
(let ((dest (if (> (get-info 'value stk-env 0) 0)
                        -.05
                        .05)))
   (set-breakpoint stk-env (list .01 dest)))
)

;;;
;;; Definition d'une fonction qui convertit les notes midi en fréquence
;;;

(define (midi-hz-najo x) (* 440 (pow 2 (/ (- x 69) 12)))) ;; x est exprimé en midicent / 100

;;;
;;; Déclaration d'un controleur midi qui récupère les noteon et les numéros de note MIDI
;;;

(define midi-ctlr (make-controller 'midi 2 "~cs6.mid" 1 (list 'noteon 'lastnote)))

;;;
;;; Initialisation de variables
;;;

(define dernier-noteon 0)
(define liste-note (list 0))

;;;
;;; Déclaration d'un vecteur de 128 emplacements (il va contenir les contrôleurs 'dynamic associés
;;; à chaque objets...
;;;

(define ctl-dyn-vector (make-vector 128))


;;;
;;; Déclaration d'un controleur scheme qui exécute la fonction fabrique-lame si la note n'a jamais été jouée et
;;; la fonction start-env à chaque reception d'un noteon
;;;

(make-controller 'scheme 0 .0
            (lambda ()
            (if (not(= (vget 0 0) dernier-noteon))
      (begin
         (format #t "Un nouveau noteon ! n°")
         (print (vget 0 1))

                        (if (eq? (member (vget 0 1) liste-note) #f)

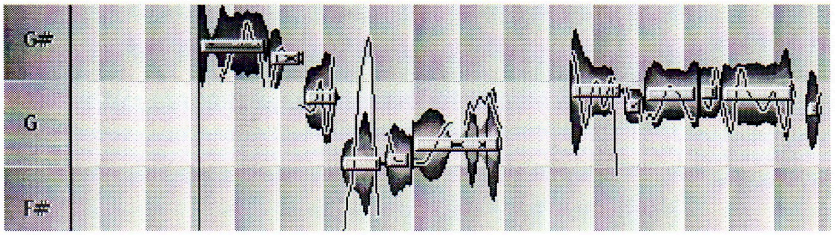
            (begin
              (format #t "Cette note n'est pas dans la liste...~%")
              (set! liste-note (cons (vget 0 1) liste-note ))
```

```
            (print liste-note)
            (vector-set! ctl-dyn-vector (vget 0 1) (fabrique-lame (/ (midi-hz-najo (vget 0 1)) 1)) )
          )
        )

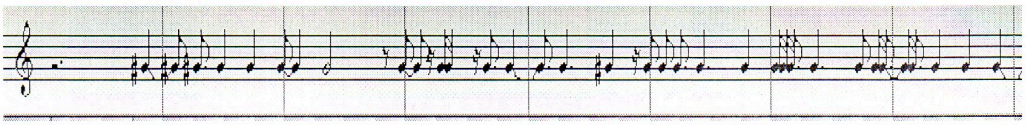
        (start-env (vector-ref ctl-dyn-vector (vget 0 1)))
        (set! dernier-noteon (vget 0 0))
                    )
    )
          )
1 (list midi-ctlr))
;;;
;;; Synthèse
;;;
(run 33)
(play)
```

SF23 is the sound of glissandi with the phoneme [m] by the voice. SFs24, 26, and 30 are the sounds of Japanese temple bowls rubbed with a metallic stick and struck with woodenstick, processed by filters and pitch shifting .

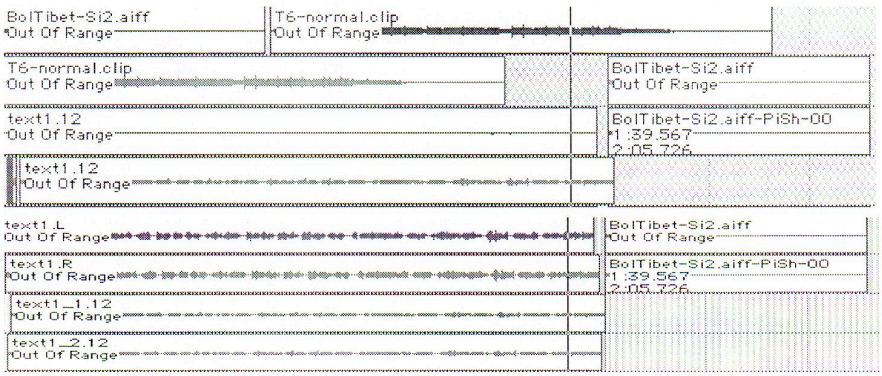
SFs25, 27 and 28 are sounds of a singer reciting the inscription. Here, as I mentioned before, I intended to create a buddhist chanting-like sound, using a soprano singer's voice. I use Melodyne software for pitch shifting and formant change to generate a very low sound with her voice (Example 45a). I made some variations transposed into around Gb2, G2, G2# and changed formants in SF25. Besides, the sound of Japanese Temple Bowl, Tibetan Bowl, wind, phonemes; [m], [sh], voice with a spectra of Japanese temple bowls (Example 45b), etc. are also transformed and employed. From event 61, five soundfiles, SFs26, 27, 28, 29 and 30 are superimposed. The music becomes accelerated and tensed into the climax of the piece with singing around event 76.



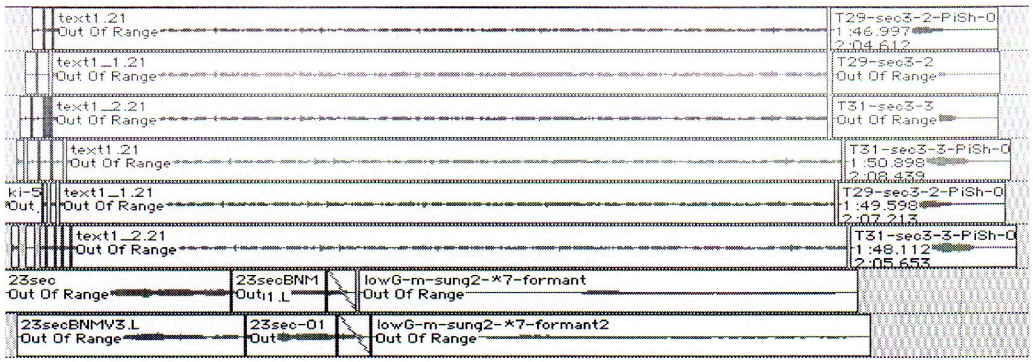
Example 45a: inscription1/Melodyne



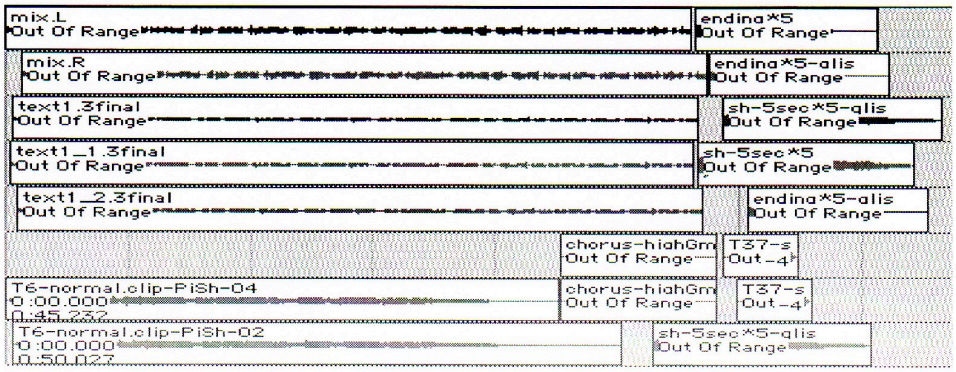
(pitched 2 oct. lower)



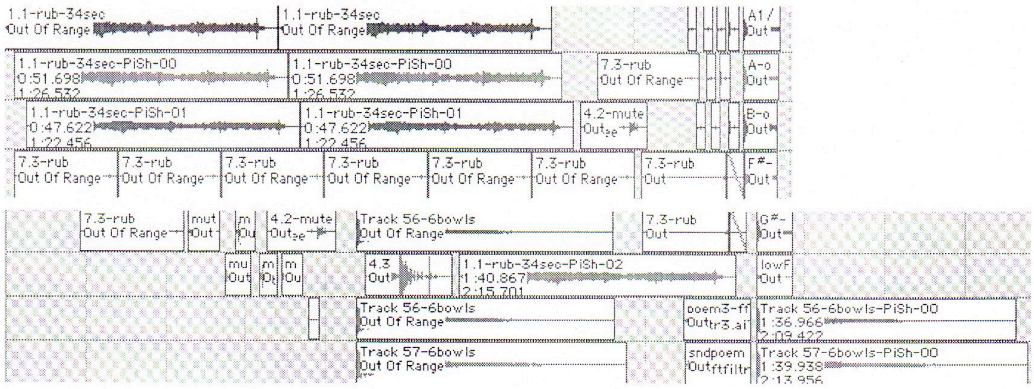
Example 45b: SF25, text1(inscription1)



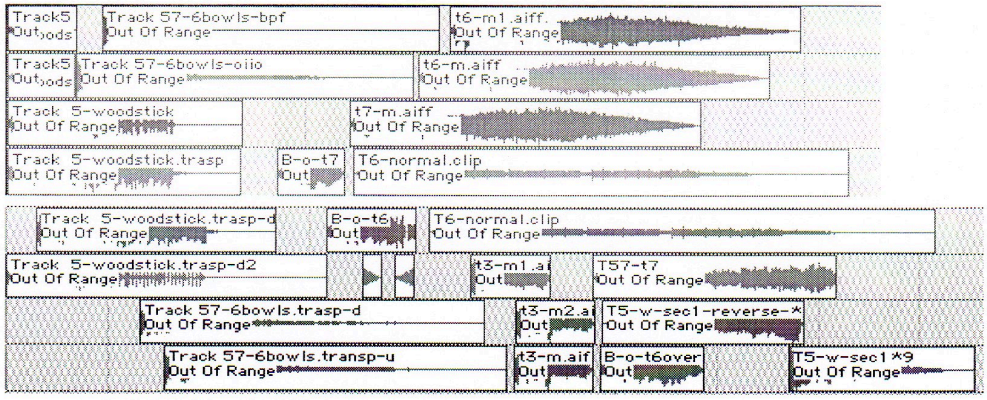
Example 46: SF27, text2 (inscription2)



Example 47: SF28, text3 (inscription3)

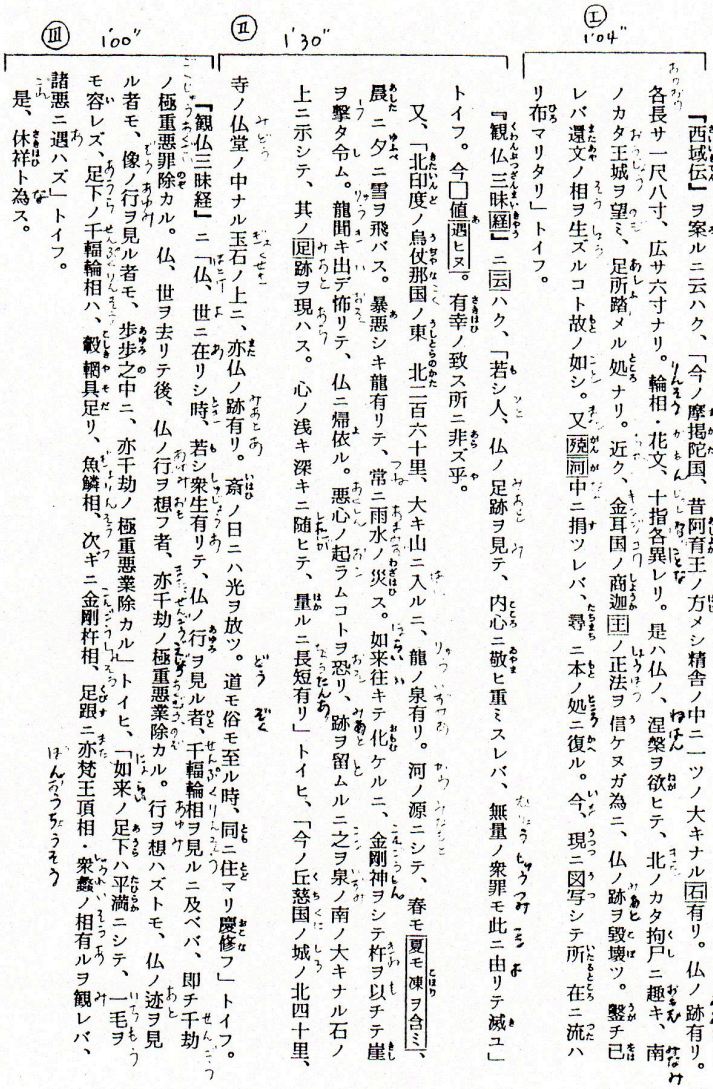


Example 48: SF24, jtb1



Example 48: SF26, jtb2

Example 49: Inscription in Japanese, used in the electronics, section 3



SECTION 4

"CONCLUSION"

Each section of this dissertation contains a detailed explanation of each element of the compositional process for *"Réminicence d'un ancien esprit"*. However, while I was composing the music into the score, I created electronics as well as tried real-time effects simultaneously. I believe this is very important for making interactive music with computer, because if you think and work each element separately, contradictions may arise that cause things not to work musically.

Most of the software developed at IRCAM, such as AudioSculpt, OpenMusic, Modalys, Max/Msp are convertible to share a file. They are still being developed at IRCAM. It is very convenient to process the same material with several treatments by different software. It gives more possibilities for generating new sounds and conceiving further ideas. There are also several ways to create similar sounds with different software or effects. It is significant to try several possibilities in order to discover the best solution. Each result has a slight difference. To search for the right timbre for my music is one of the processes of composing. Since I am interested in timbre or sound itself when I compose, I believe that new technology I learned during my year at IRCAM inspired me to have different ideas.

Music computer technology develops radically every year. At IRCAM, the software I learned about was already being updated and changed. This demonstrates the need to stay aware of ongoing transformations in this rapidly changing field.

APPENDIX

1. Software used for "Réminiscence d'un ancien esprit":

Audio Sculpt (for Macintosh) : Software for sonogram (spectra graphical analysis) as well as for sound modifications such as filtering, cross synthesis, expansion/compression of time, transposition etc. with graphical interface, developed at Ircam.

Max/Msp: Software for creating real time computer programming for live music, distributed by Cycling '74 and developed at Ircam.

Melodyne: Software for sound processing, distributed by Celemony Software GmbH. Special features include correcting of the formant position in order to preserve the character of a voice as well as change of the formant position to create a new character of a voice.

Modalys (for Macintosh) : Software for sound processing, developed at Ircam, used for creating virtual instruments based on physical instruments such as strings, metal plates, tubes, membranes, plectra, reeds, and hammers.

Open Music (for Macintosh) : Software for computer aided composition, developed at Ircam, using Common Lisp language on Machintosh. Completed with a collection of libraries such as OM Kant, OM Chaos etc.

Protools: Software for recording, arranging, editing, and mixing sounds, distributed by Digidesign, Inc.

2. Score of "Réminiscence d'un ancien esprit"

: see following pages