abstract: The live performance uses of the Macintosh computer are discussed. First, as a source of notational information for performers. Second, as a stand alone sound-source using its internal "sound-driver routines." Thirdly, as a mouse, icon, and function table based music controller for dynamic interactive use in live performance.

Apple's Macintosh computer has opened up new vistas in affordable computer music for me. It is my intention to share some of the discoveries I've made with others working in this field under similar budget constraints. I should add that many of the ideas I will be discussing do not specifically require a Macintosh for their implementation. All that is required is a high resolution graphics capability (512 x 480 or greater), a mouse (and mouse roaming software), and a user extensible language environment (Forth and tilde assembler in my case).

1. NOTATION

The first area of investigation was to take advantage of the enhanced graphical capabilities of the Macintosh for the purposes of extending conventional music notation. The support of the QuickDraw toolbox utilities made it possible to move and animate predefined objects about the screen. When the objects happen to be musical notation, then the composer is given the opportunity to make conventional music "come alive", as it were, in an everchanging animated image. Viewed at any one instant, it would appear as conventional music, but over time musical notation could be seen to change position and orientation. The most straightforward use of this notation system is to represent scores without page turns, seamlessly scrolling from the beginning to the end of a composition. A line drawn through the center of the staff(s) can indicate the current "now" as the notation is scrolled from left to right.

If the intentions of the composer are more aleatoric, the center line can be deleted as well as any scrolling. In this context, notes and note systems can be seen to appear, move about, transform configuration, disappear, regroup, etc. all under the composer and/or algorithmic control.

This is made possible through interactive control. The mouse, the keyboard, and the use of MIDI give a source of input to the way the composer/performer may interact with the system. Any new and/or changing notation can be made to react to the players interpretation of the previous notation. The music is seen to react according to how it was interpreted. For example, if the composer wished an even probabilistic distribution of a set of pitches to be played by the performer, then the computer can journal the players use of notes in the pitch class, and delete the notes that have been oft repeated, until the distribution becomes more balanced. As a further example, an initially rather indiscriminate looking aggregation of pitches and symbols, can slowly evolve into a more and more delineated form of notation. This is accomplished by keeping track of the MIDI data stream. The routing then determines which phrases the player has chosen to perform in what manner. The notation changes to reflect the players previous performance.

I often use graphic notation accompanied with lengthy discussions with each of the players in my work. I try to configure the
performance around the individual strengths and preferences of the given performers. The problems that arise in this context are usually ones of synchronization and the misinterpretation of graphic symbols.

It is possible with the Macintosh to dynamically edit a visually based score. Therefore, the score can be changed easily, during rehearsals, to accommodate the players’ personal preferences.

There is an additional advantage, in the real-time context of passing messages and symbols to the performer, in the midst of the performance. The notation becomes a form of improvisation, in which the score is extemporaneously generated from the composer/conductor’s pre-defined segments and symbols. In this way the vocabulary of improvisations and aleatoric interaction can be opened to new realms, based not only on pre-existing forms of nomenclature, but also generating new ways in which the musician interacts with notation.

2. Macintosh as a standalone sound source

The sound production mechanism on the Macintosh, I am sure, was an engineering afterthought. It appears to be a byproduct of the byte-oriented table from which the disc drive derives its motor speed changes. There are however some interesting avenues of investigation that have yielded, for me, usable results.

First, a brief description of the 3 Rom-based sound-driver software routines is in order.

The first of these routines is a monophonic squarewave routine not unlike the one found in several microcomputers. It is not a very interesting or musically useful format.

The second routine is a four voice subroutine, that needs a user to supply a 256 byte waveform to a buffer for each of the 4 voices. It is, therefore, heterophonic. Additionally, the phase and frequency of each buffer is determined by the user.

The frequency resolution is quite good, giving the user 16 million divisions of 8.5 octaves or 0.07 cents resolution in the mid-range. This lends the Macintosh to experiments in different tunings and temperaments, which I have found to be extremely useful for tuning microtonal acoustic instruments, and investigating new harmonic relationships based on all manner of unconventional notions.

The main disadvantage of the four voice routine is that it is a steady state buffer, therefore the spectrum of any waveshape put in the buffer is fixed, making it organ like in its musical quality (either on or off).

The third routine is called the free-form sound driver. It is more like the output driver found in a typical computer music environment. A buffer of precompiled sound information can be passed as a block to the sound buffer. The sound buffer then plays the block at a speed set by a user variable. It is ideal for creating a low resolution single channel “sampling machine” (i.e. FairlightB, Emulator®, etc.), or for executing Music V style precompiled sound files.

So far, I have found it more convenient to use the four-voice sound driver routines to implement and experiment with timbre algorithms. I have not fully investigated the free-form buffer routine.

Working with the 4 voice driver, I tried to find a way to change the timbre of these fixed waveshapes, and create some sort of spectrum change in time. I came up with three different and satisfying approaches to this problem.

First, taking an example from the model of the oscillator in Salvatore Marzano’s “Sal Mar Construction”, I created a stochastic waveshape oscillator. In this model, the period of the waveshape is supplied by the user, but the waveshape itself varies in time, according to an indeterminate process. The indeterminate process I used was to initially
plot anywhere from 3 to 16 random points, in a 256x256 array. I then linearly interpolated between them to create a waveshape. A time-active parameter is passed to the routines. The routine then varies the waveshape, so that it is collapsed over that time period, to steady state or dc (i.e. silence).

This routine is asynchronous with the pitch determining mechanism, and can, or can not, be construed as an envelope, in the conventional sense, depending on how it's used.

The second form of spectral change I implemented for the four-voice driver was a form of what I call, transformational filtering. The underlying principle, in this form of timbre manipulation, is to find a mechanism for linking divergent waveshapes, into a continuously varying sound context. In this model, the user supplies destination points and decibel settings, for each of several static waveform sources. The routine then maps the transitional waveshapes, that are between the source waveshape, and the destination waveshape.

At this time, the path between any two waveshapes is a linear one. The transition progresses evenly from source to destination. It is my intention, to augment the software to support nonlinear transformations, or the mapping of the transitions, between source and destination, along a user-defined curve. The characteristics of the instantaneous waveshape are more or less like the destination, depending on its relative distance from the destination waveshape.

The third and final timbre experiment to implement the Timbre Modulation algorithm, as put forward by Daniel Arbib (1973) and Mark LeBrun(1979), and commercially implemented in Don Buchla's Tocure' and 400 series instruments. The initial advantages of this technique, in the Macintosh user-environment, are in using the mouse and graphics to instantaneously input new modulating functions to the timbre modulation table. This synthesis technique is simple, and yet very rich in content. I am constantly finding new ways of combining material, in this context, to come up with rich new resources.

One option I have pursued, with both the four-voice and the free form drivers, is to treat the sound driver routines as monophonic voices. I have gotten good results with both options in this way. This does limit the Macintosh's use as a stand alone music source, but in the context of a Macintosh based MIDI environment, it is an added source of color and change.

3. Using the Macintosh with MIDI

The interfacing of the Macintosh to the MIDI environment can be handled in one of two ways. First, one port of the 8530 Serial Communications Controller, on the Macintosh, can be configured to handle an external clock in, on its handshake control line, if a 1 Mhz signal is applied to this line, the Macintosh can divide this into the MIDI freq. (3:25 Khz). The Macintosh must then have outlines for the protocol of all the MIDI parameters that it is handling.

Secondly, the Macintosh can communicate with another computer, by way of a High Speed (19.2 Kbaus) RS-232 link. The other computer can then handle the MIDI protocol, and simply pass parameters that are relevant to the routines running on the Macintosh.

309 ICMC '85 Proceedings
The second technique represents much less overhead on the Macintosh, and therefore is less consumptive of the Macintosh's computing power. This leaves the Macintosh more room for computing and input analysis. Additionally, the slave MIDI computer can be down-loaded from the Macintosh with subroutines, that perform specific MIDI macro-instruction sequences. A library of slave computer subroutines can be transferred, with no computational overhead. For these reasons, I have opted to implement my user environment in this manner.

4. MOUSE & GRAPHICS AS A REAL-TIME MUSIC INPUT STRUCTURES

The advent of the Macintosh user environment has put forward many possibilities for musical control in real-time. I will describe experiments, I have undertaken, to optimize its use as a musical controller.

Initially, I experimented with using just a graphics window, as a stand alone input source. I first wrote several different routines that created visual patterns algorithmically. (Lissajou's, Epicycloid, Hypocycloid, Bessel functions, Gaussian & Poisson distributions, etc.) Then, I quantized the X and Y coordinates to reflect a division of pitch. X determined the pitch chroma, and could be set to any irregular scale. For example, the full range of the X-scale, for a given graphic spike, could be divided to represent an 11 limit just-intoned scale. The Y coordinate can be set to represent which octave the correlated frequency would sound in.

Pitch and the execution of just one figure, as described above, is probably the simplest case. It is equally as easy to execute the creation of several different graphic algorithms, each passing specific musical parameters. This can include the execution speed of the graphics window. Therefore the rhythmic relationship of the visual and aural information is generated by a given algorithm or formula.

I then realized that the execution of the quantization need not be limited to a scalar sort of representation. The way in which any axis is divided up can be represented by any set of numbers. It does not need to be arranged from minimum to maximum. In this way, an axis can just as easily represent a melody fragment, or the result of a previous quantization of a function.

The model for this form of transformation, is not unlike that of the Timbre Modulation technique I discussed earlier. In this case, however, the time frame is significantly slower.

In my current model, I have available 256x256 functions that can be called and/or "re-recorded" at any time. In this sense a function is a 2-dimensional map of any user generated 256 byte contour. An input stream is mapped against a function that outputs a resultant function. The input function has the option of being derived from a MIDI input stream, or a mouse path, or the record of a mouse path, or a current mouse path mapped against the record of a MIDI input stream, etc. The combinations are limitless.

These ideas are derived from the, as yet, unknown work of Daniel Johnsson on music process control techniques. He originally developed software function boxes of this sort, for use in dynamic control of Serge Modular Music analog systems, when

ICMC '85 Proceedings 310
working with Serge Tjörnstrand.

The advantage that the mouse bring to this environment are twofold.

First, the mouse can be used to create linkages between input streams and function boxes, to the output parameter passing routines of the system. (MIDI and/or Macintosh resident sound-driver routines). There are several ways in which the inputs and functions can be linked, especially since the function boxes are allowed to average all inputs. The Mouse can control the flow of information, in real-time. it can also enable or disable functions, or change the definition of a function instantaneously.

Secondly, the mouse can act as an input stimulus. The idea of using the mouse, as a real time controller, in a music generating system can only be successful in the right context. There is, in any interesting musical instrument, the need for a challenging technical facility. This should be no less true for the electronic arts. Further, since the parameters and nuances of the electronic arts are of greater range, an even higher level of virtuosity must be attained to harness the full range of human expression of which these new tools are capable. The mouse is, by no means, the final solution to this sorely neglected problem. I have attempted to maximize its potential for real-time music control, though. Hopefully, it has an inherent virtuosity that is implied when used in the context of this music generating system.

The mouse is capable of expressing two instantaneous coordinates, or rates of change (X & Y) and, in the case of the Macintosh, a single rhythm (or mouse button open and close sequences). It is possible to give macro commands with a single keystroke on the computer's keyboard. A graphics space like the one illustrated below, can have buttons described in it that are directly congruent with these single key macro commands. When, in the mouse music input mode, I have chosen to place the 'activity' of the mouse in three distinct regions of behavior. The distribution and allocation of the pushbuttons in region1 and #3 are user-defined. Additionally, each individual pushbutton can be a user created icon to facilitate rapid identification. This is to tailor the input and output structure to the needs of the user's specific hardware system(s).

**Region1** (to the left of the screen) It represents the initialization and enabling phase of what the upcoming mouse activity is about to change. It is primarily a field of push-buttons that the user predicates to represent:

- allow how much scaling of each axis will occur
- allow how much quantization of each axis will occur & by what function (o-F) each axis will be quantized,
- what parameter (pitch, amplitude, timbre, modulation, velocity, LFO) will be affected by which axis,
- whether the results of the upcoming mouse activity will be averaged and to which rhythm (table(0-F)) and parameter (pitch,amplitude,filter,modulation,velocity, LFO) will be averaged with which axis(x, y, xy) and by what percent (%).

**Region2** is the mouse activity purview area. It is in region2 that parameters that are actually input. When the mouse enters region2, all the parameters and functions that were initialized, in region1, become active. Along the bottom edge of the region, a time-arrow indicates the elapsed time in the region. This may have been set by:

- a time setting in region3
- by previous pass in region2
- by the current pass in region2 (starting at the exit of region1 and ending with the entrance to region3)

In case a,b, the time that region2 is active is preset by definition. The arrow is not activated in case c.

**Region3** is the link to the concurrent processes that are being executed while new mouse parameters are being input. It is,
also, a field of push buttons for enabling or disabling process controls for the next pass. The push buttons represent:

a) which of the six sixteen presets for Region#1 will be active

b) which of the three time-arrow states mentioned above will be active

c) how the output will be parted out on the next pass (1,2,4,8,0:1,2,4,8:0:1,2,4,8)

d) which polyphonic algorithm will be active (1:round-robin, 2: serial/parallel, 3: on, until release)

e) which rhythm table (0-F) should be active with which voice (0-F)

f) whether to make the last pass an active concurrent background task (0-1) with the next pass (overdub)

g) which of the active concurrent background tasks (0-F) to kill before next pass

h) jump to section#1 for new pass, and return cursor to region#1

A typical session with this instrument always starts in region#1, proceeds to region#2, and ends in region#3, before jumping back to region#1. Macro functions, that are enabled requiring numbers to be input to them, blink until either, the appropriate numbered keys are sequentially activated, the numbers are input from the computer keyboard, or the Return key is struck. If the return key is struck the macro function maintains its previous value.

The arrangement of the mouse push buttons is made in an initialization mode. In this mode, a library of Fortm words are linked to specific pushbutton regions. A set of pre-defined, and user definable, symbols and special characters are linked to each specific pushbutton region. The Fortm vocabulary includes all the functions outlined above, and is user extensible. This flexible arrangement leaves the instrument design open to rearrangement and augmentation to suit the players and/or composers individual dispositions.

The limitations and connotations of accurate use of this instrument are inherent in the nature of the mechanism, on which its based. This is also true of the violin, the flesch horn, the Pratt-Reed diode array, keyboard, etc. My point is that any musical instrument is constrained by the elements that constitute it. It is within the limitations of an instrument, that artists find the edge of their expression.

5. Conclusions

The Macintosh's power and affordability make it an attractive tool for experimentation in new music ideas. The use of visual and kinaesthetic data, as input to the computer, are unlike new facilities. This is, by no means, an end to my investigative work with this, and other, new affordable devices nor is it fully complete. Further technological developments will bring even more innovation and opportunities to our field. Hopefully, someday, powerful computer music systems will be more in the hands of artists, and less dominated by large institutions and commercial studios. The Macintosh, I believe, portends this change.

I hope that some of the ideas discussed herein are of use to you in your own work.

I would like to thank my wife, Kathy, for her editorial assistance. My associates at the Center for Contemporary Music at Mills College (David Rosenboom, Larry Fojarndy, Richard Povall, Maggi Payne,), and my associates at Studio Software, (Darral Johansen, Tim Perks, and Chris Brown), have been invaluable in their assistance, support, and inspiration on this, and related, projects.

References
