

The Personal Orchestra, or Audio Data Compression by 10,000:1

Michael Hawley MIT Media Laboratory

ABSTRACT: Virtually all of western classical music could be recorded and stored on a single computer disk and rendered on a music system costing as much as a good piano.

To demonstrate and explore this, I built a system for audio and musical research which permits a workstation to control 64 synthesizers and a computerized Bösendorfer Imperial concert grand piano. It is easy to assemble and play complex scores on the piano accompanied by a synthetic orchestra. Examples here include a fully orchestrated ultra-virtuoso arrangement of Franz Liszt's "third" piano concerto, the *Totentanz*. There are also families of analytical programs that parse musical data to identify key, melodies, and other interesting features.

Since a performer generates about 3 kilobytes *per minute* of gestural data, the volume of recorded information is small: *Moby Dick* uses more space than all of Scott Joplin's piano music; Bach's is 6 megabytes. Consequently, even elementary processing provides great leverage over such concentrated information. This will open new avenues for entertainment media, while retaining a grand orchestral, yet personal, perspective.

It would be quite possible to arrange to control a distant computer by means of a telephone line.

– Alan M. Turing, foreseeing terminal-modem hookup, in his lecture to the London Mathematical Society, 1947.

Extraordinary how potent cheap music is.

– Noel Coward, 1932.

1. Introduction

In the near future, we can expect to see an orchestra in a box: a synthesis system that is controllable by computer and can produce sounds as appealing as those made by traditional instruments. Forms of this technology are already finding use in many music production situations. More than 40% of cinema and television music is now produced synthetically. The startlingly low bandwidth of musical performance information – about 3Kbytes *per minute* of piano playing when recorded at the gestural level – implies the controller can be an inexpensive computer, and enormous musical databases can fit into very compact media. Technology featuring a “critical mass” of music whose content can be amplified by an orchestral rendering system will someday be commonplace.

Suppose we had a digital Boston Symphony Orchestra as a home stereo add-on, and the apparatus to control it. What would people want to do with such stuff? Most musical experiences, and many forms of entertainment, are passive for listeners: turn on a disc player, or go to a concert, and just listen. Lack of listener interaction is usual, and social ambience – being “at the MET,” in a smoky jazz club, in a cathedral, at a parade – or watching wild personalities like Horowitz or Jerry Lee Lewis, often has as much

to do with the enjoyment of the experience as any other factors, if not more. And little to do with computers. Franz Liszt and Niccolò Paganini pioneered the image of the virtuoso in the 19th century, and there is a beauty and excitement in sharing a live performance that is only a vicarious memory in a recording. Nevertheless, the fundamental density of musical control information makes possible many interesting applications. For example, society's hi-tech "music box" at the moment is a CD player, but if the gigabyte of audio data (and the disc) were replaced by a few megabytes of control data, and if the lasers and mechanics were replaced by synthesis electronics, the music box of tomorrow would play not just one recording of Glenn Gould playing the *Goldberg Variations*, but *all* of Bach. The fact that a controlling processor is operating on music information encoded at a high level (notes, finger motions), an encoding which is well-aligned with the semantics of the situation, makes it possible to consider rendering the data in other ways – faster, slower, using different instruments – or to implement content-based computations (the *Name That Tune interface*: hum a melody and the music box retrieves the work and plays it). These examples might seem fanciful but are not at all far-fetched.

Computer music activity was for many years largely mired at the sample and sound-synthesis level, which consequently focused much of the attention away from more natural levels of gestural control. This tended to produce music which wasn't well-matched with most cognitive expectations because algorithmic controls were fitted to the low-level apparatus of synthesis, which tends to be independent of features that most people consider as characteristically "musical," like rhythm, tonality, or even "notes." Inappropriate technology at the control level kept computer-based music from competing with traditional kinds, much like any misplaced choice of tools or abstractions can limit possibilities. In fact, the development of musical devices that directly link keyboards or other transducers to interesting sound synthesis hardware is a relatively new phenomenon. The spectrum of music applications seems so exciting today precisely because attention and technology have shifted toward content. The nature of music control information makes it easily computable with present

technology, and good synthesizers, interfaces, and computers are readily available.

In the larger world of multimedia computer applications, there has been a gap between what can in practice be implemented with digital and analog systems. Broadly speaking, analog components provide fast technology amenable for signal-level work but do not often facilitate content-oriented computation. Digital systems, on the other hand, are well-suited for processing of discrete and dense content information, but because of limitations in nascent technology, have only recently grown fast enough to deal with considerable high-bandwidth signal information. Innovative connections between content and signal foster the real breakthroughs in communications media, and over time, the gap between content and signal will become less significant. For the moment, an appealing aspect of musical applications is that they clearly demonstrate what can be done when it is easy to compute along the entire spectrum of information, from script-like content to signal and back. As with other dense encodings, like text or algebra, computers can do remarkable, savant-like things with music control information especially when not bound by the inertia of general knowledge. This is reminiscent of the situation with chess, where computers with virtually no general knowledge are beating grandmasters, and finding new winning solutions to endgames that had previously been thought hopeless; or in symbolic arithmetic, where a program by Doug Lenat once rediscovered the obscure notion of maximally divisible numbers, known to Ramanujan and a few other experts. The delightful thing about music is that growing computational power has opened realtime pathways between musical content and signal sufficiently so that the nature of the information can be studied in the context of extremely enjoyable music and realistic sound.

I will describe related work in representation and musical control-level applications, then discuss our system components and what they imply. Because notions of supercompressed rich musical databases and representations for the data will be common in the future (and because they have enabled us to build better scoring tools) I talk about them in some detail. I then present a few applications with musical examples that demonstrate what can be done by combining a grand synthetic orchestra

with sufficient computational control and a high-level representation.

2. *Related Work*

A considerable amount of work oriented toward the representation of musical information to facilitate both composition and performance has been done. Seminal papers include [Buxton 1978; Buxton et al. 1978; Buxton et al. 1981] in which Buxton and colleagues developed an event-list data structure for music. They implemented pervasive forms of *instantiation* and *hierarchy*, and applications using this information. These are useful notions; I applied some similar ideas. The Northwestern group of Decker, Kendall, et al. [1986] built a well-integrated family of UNIX tools for dealing with music performance data, whether at signal-level or performance-level. Peter Langston [1986, 1990] and Hawley [1986] have also experimented with similar equipment. Much of Langston's work involves music languages and grammars for generating compositions. There have been other interesting language approaches, more for composers and researchers than for automated composing algorithms, like Schottstaedt's *PLA* [Schottstaedt 1983, 1984] and Dannenberg's *Arctic* [Dannenberg & Rubine 1986]. Much of the function of these languages is to experiment with instruments at a low level. Langston's approach, like analogous approaches to handling textual information in UNIX, has the virtue of a kind of synergy made possible by building a friendly family of components.

Authoring systems can be built for composing with computer-driven instruments. Commercial *sequencers* are examples of these, as is the arranging application described here. Systems can interactively augment or amplify human performances. Tod Machover [Machover & Chung 1989] calls this class of tools *hyperinstruments*. Barry Vercoe [Vercoe & Puckette 1985] and Roger Dannenberg [Dannenberg & Bloch 1985; Dannenberg & Mukaino 1988] have built *synthetic accompanists*: programs that track live performers and accompany them. Robert Rowe [to appear] has written a program called *Cypher* for tracking live input and improvising against it according to a mapping of musical

features in the input to desired responses. Algorithmic composing and rendering systems can be built that might pass a musical Turing test – Cope’s program [1987] has already composed plausible impersonations – and the eventual presence of large digital libraries of music data should invite forms of content analysis and synthesis-from-content that previously have not been possible.

Maxwell and Ornstein’s *Mockingbird* [Maxwell 1982; Maxwell & Ornstein 1983] was intended as a “note processor” for producing high-quality, printed music, but was also important for its elegant interactive design, and the division of labor between the human user and the slightly intelligent assistant. Casting a performance control stream into a graphical musical score is a complex job, one that requires considerably intelligent intervention. The reverse process, optical reading of scores to cast a graphical notation into a performable data structure, has been studied by Baird [1987], Kassler [1972], Prerau [1970; 1971; 1975], and Ruttenberg [1990], who is programming the Connection Machine to read Schubert string quartets. Audio transcription technologies are also beginning to emerge [Massalin 1989]. All of these help establish links between signal and control level data.

3. *Apparatus*

I use a Sun-3/260 (25MHz, 8MB, 1600x1280 bitmapped display) with a custom VME-bus interface that controls 4 real-time MIDI processors (Roland MPU-401), for a total of 64 channels of MIDI output, and 4 channels of MIDI input. The Sun runs a local window system related to Hawley & Leffler [1985] and some MIDI driver code and system tools (e.g., Hawley 1986). An IBM PC-AT with a special multibus and Z80-based controller handles realtime piano control, and is connected to the Sun through a serial port. Instruments and other musical apparatus include a Yamaha KX-88 keyboard controller (a silent keyboard with weighted keys that generates MIDI data), an MJC8 MIDI junction box, a number of synthesizers (several Kurzweil devices, an E-mu, some Yamaha synthesizers), a mixer, amplifier, and some good speakers. In

addition, there is a Bösendorfer grand piano. A NeXT is beginning to find use for synchronized digital audio work.

The control machinery can manage a large quantity of sound, limited in richness primarily by the number and quality of synthesizers that are plugged in. Consider that a typical symphony orchestra contains about 60 players, but most play duplicate parts. Even a very rich score, like Berlioz's *Symphonie Fantastique*, contains at most 32 distinct parts, of which only 6 to 8 parts are active in parallel (and not duplicated) at any time. I have only partly populated the channels with synthesizers, rarely using more than a dozen, yet already there are several *thousand* pre-loaded instrumental voices from which to choose, and more in libraries, so the timbral palette is large.

The Bösendorfer piano deserves a special word. It is an Imperial concert grand, the largest piano in commercial production: 9'6", 1400 pounds, 97 keys (it has nine extra low bass notes) and 3 pedals. It has been instrumented with a high-quality digital recording and playback mechanism, designed by Wayne Stahnke for the Kimball company, which now sells it as a product. Hammer timings and velocities are recorded optically (onset timings to roughly 800Hz, hammer velocity to 10 bits of precision) and played back using a solenoid stack. It is the most magnificent player piano in the world.

It is also the most expensive player piano in the world: at about \$120,000, the piano costs more than twice as much as all the rest of the equipment, though currently still only about as much as a gigabyte of silicon memory. It is worth mentioning that hammer velocities, pedal positions, and timings for these events completely characterize all the device-level information: no other information is required for perceptually perfect piano reproduction. Also, the recording parameters are comparable in resolution to MIDI synthesizer data. For instance, half of the piano's $2^{10}=1024$ velocity values are for silent but moving hammers, and another bit or two is likely lost due to slop (noise) in the mechanism, making piano resolution comparable to the 7-bit quantities used by MIDI. This makes it relatively easy to unify full-resolution piano and MIDI data structures, so that all instruments can be conducted by the workstation. The piano has recently been upgraded to handle MIDI I/O, which seems sufficiently

expressive. It is controlled by custom Z80 processors which are in turn monitored by an IBM PC-AT. This runs command-line applications to play and record. The PC, in turn, is controlled by the Sun over a serial line, acting as a remote console.

In sum, this apparatus is not far from what should be commonplace in 10 years or so: a control-level processor with enough bandwidth to drive 64 “performers,” a synthesis system capable of rendering sounds of pleasing richness (if not orchestral quality) and I/O (as well as a database) that makes interesting interaction possible. It may be some time before there is a Bösendorfer in every living room, and there is a real charm to the robotic keys on a player piano, but synthetic timbres will be much more competitive with traditional instruments over time.

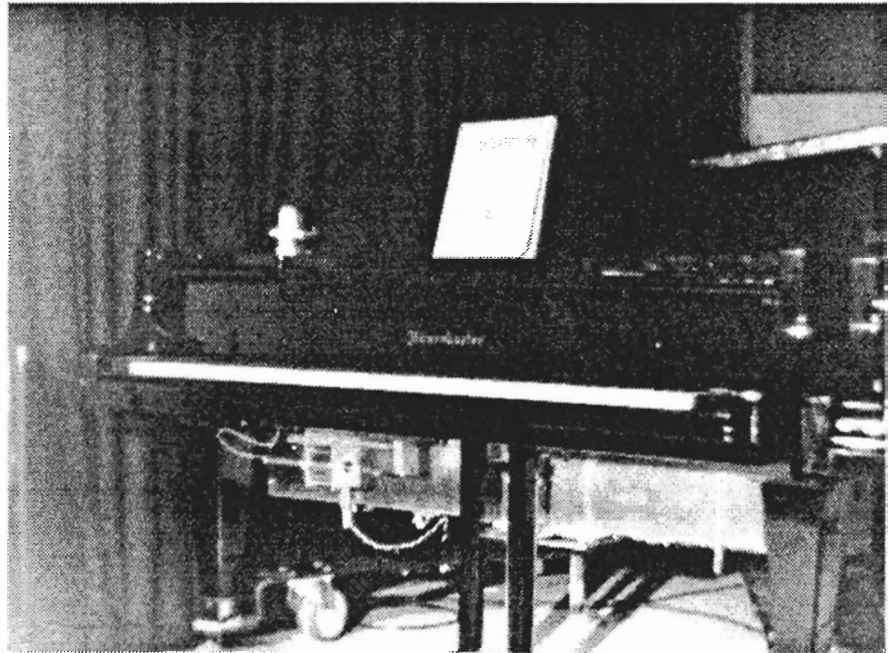


Photo of piano and stack

4. *Data Representations – How Much Music Is There?*

Statements like “...all of Western classical music... on a single computer disk” need some qualification. As was mentioned earlier, performance or control-level data is the information captured at the keystroke or gestural level. Performance event lists are treated as digital recordings of interface activity with 32-bit timings in milliseconds and velocities to 16 bits, for generality. Data of this kind encodes meaningful information in a compact way. Our measurements show that a keyboardist, whether playing on a Bösendorfer, a pipe organ, or a synthesizer clavier, almost never generates more than 10,000 bytes per minute, and the average flow is more on the order of 3,000 bytes *per minute* for reasonably active music, like Joplin or Bach. Even seemingly high-bandwidth pianists, like Art Tatum or Oscar Peterson, average in this range. I expect other kinds of instrumentalists, such as violinists or wind players, also provide roughly this much data when controlling their instruments.¹ This number seems amazingly low, at first, but piano-playing is really only fast typing and some foot-tapping with time and velocity sampling, so the data rate should not be surprising.

Now consider that digital audio, even if sampled and encoded at poor telephone quality, requires about 3Kbytes per *second*, nearly 2 orders of magnitude more than performance-level data, and digital audio of compact disk quality requires a factor of 30 more than that. (Image data, uncompressed at about 300Kbytes per frame, or 7.2Mbytes/sec, is still two or three orders of magnitude more.) By contrast, if one sat down to play Joplin for 24 hours straight, one would first run out of Joplin after 6 hours or so. The average Joplin rag takes about 5 minutes or 20Kbytes. Since he wrote roughly 50 rags, *all* of Joplin consumes about a

1. Interfaces that involve lips and skin on reeds or strings may seem more subtle, but the amount of *useful* gestural control information supplied by a performer is still probably very small. Digital wind instruments and pressure transducers on many MIDI keyboards tend to produce large bursts of data, but it's extremely compressible. Even in piano-playing, the keyboard information is, for example, about 1600 bytes per minute, while pedalling, which is not much compressed, can produce 7Kbytes per minute. Future transducers will operate increasingly off acoustical signal.

megabyte, uncompressed. This is a tiny amount of data, and straightforward compressors (like using a pitch lookup table, a key signature, run or delta-coded amplitude information) would make it even smaller, certainly by 2:1 (in fact, Woods' compress program, which is based on the Lempel-Ziv algorithm, already reaches 1.8:1 on MPU-401-format data, which contains embedded time offsets that are likely to disrupt the algorithm). Yet it could be transmitted by modem and immaculately rendered by a computerized piano or an orchestra of synthesizers. Furthermore, if Joplin's output amounts to only a megabyte, a generic composer of western classical music would yield perhaps 10 megabytes in a reasonably fertile career.² For example, all of Johann Sebastian Bach's keyboard music consumes about 45 LPs, or 30-40 hours, plus or minus a few hours of repeats. This should be about 6 or 7 megabytes (Bach was extremely prolific). At such rates, a database containing the complete works of 50 composers – which would take more than 400 *days* to play – could be kept on one 500 megabyte CD-Rom.

Now, pushing the envelope calculation, how much music exists? To pick a healthy subset, the Library of Congress currently has a collection of approximately 2M musical recordings, which they estimate average about 30 minutes each [Nichols 1989], or 1M hours, total. This is approximately 200GB in gestural transcription. (In signal form, of course, it would be about 650 terabytes of CD-rate audio, or 14TB at phone-rate.) It is safe to assume that the entropy of this sort of information would permit compression by a reasonable factor, but in any event, it is not an overwhelming amount – suppose one “LC” of music is 100GB – and it is a certainty that within 2 or 3 decades, storage media will have evolved that allows consumers to own hundreds of gigabytes of fast random access media in a rather small box. It is also fair to assume that industrial-grade acoustic-to-gesture converters of

2. The typical great author also writes about 10 megabytes of text in a lifetime. *Moby Dick*, at 1.3Mbytes, is a significant fraction of Melville's output, and all of Shakespeare is about 6MB. Since musical performance data flows about 5x “faster” than typing text, one might expect great composers to produce something more like 50MB per life. This is true of improvisers, but thoughtful music seems to be slower and harder to compose than text, in a certain sense, probably because the decisions invited through various musical constraints (e.g., polyphony, harmony, voicing, etc.) have about the same complexity as text.

reasonably high quality can exist by then. In terms of hours of long-playing hi-fi, an LC of music takes about a third of a millennium to play, if played 8 hours a day, day in and day out. 200 acoustical transcription machines could do the conversion in about 6 months, though, and it does not seem at all unrealistic to suppose that this can happen within 20 or 30 years. Thinking quantitatively about the sheer volume of music in the world is not difficult, though it is bizarre to imagine what commonplace personal collections of this size will imply.³

Returning to representational issues, the situation for music is analogous to text as a dense encoding of speech. Like graphical music notation, printed text doesn't carry all the nuance of an utterance but this is of little consequence compared to the value added by retrieval and other kinds of mass-dissemination or vivid content-oriented access. More importantly, the vital difference between, say, a performance of Joplin that sounds mathematical, metronomic, mechanical, and dull, compared to a performance that sounds lifelike and compelling, is encoded primarily in the timing values of only 15K of data. To give a more specific example, the musical notion of *ritardando* – slowing down at a cadence, analogous to “phrase-final lengthening” in speech – is one of the most basic expressive devices in music. But quantifiable theories of *ritardando* are feeble at best. Is time being stretched in some kind of exponential or Fibonacci proportion (2:3:5...) because the brain is doing so many binary-grouped autocorrelations? Or is it a harmonic stretching (1/2:2/3:...)? Phenomena like these should be easy to measure and test from performance data. Future “smart” music rendering systems will have to include parameters to control expressive features like these, but their nature is not yet well-known.

As we begin to explore what can be done with a useful gestural recording of music data, we should remember that analogous

3. To put music library content in a more local perspective, the MIT music library has 10,599 books, 1,478 journals, 359 pieces of microfilm/fiche, 24,381 musical scores, 17,348 LP's, CD's, and tapes, and 73 videos. The audio material probably amounts to roughly 2GB, which can be thought of as the disk an MIT student will someday buy when enrolling in a music class. The score content is somewhat harder to estimate, perhaps 2-10GB. The ratio at LC is roughly similar, about 2M audio recordings and 7M other musical items, which includes scores, manuscripts, texts, and “realia” (e.g., death masks, busts, a chunk of Beethoven's hair).

opportunities await for speech and other kinds of recording. After a speech recognition algorithm has done the work of transcribing text from an audio utterance, the “intermediate” information – timing, pitch modulation, etc. – should not be thrown away, for it will be useful in training synthetic devices to speak or sing more naturally. Current speech synthesis from text sounds stilted and unappealing for much the same reason that algorithmically-performed music sounds as if it was “played by a computer.” None of the work here addresses the encoding of vocal information in music, but the same reasoning applies. We will eventually require a recording process that captures a structural analysis of vocal parts so that they, too, can be resynthesized subject to content-level modulation. This may take the form of language translation (mapping timing, pitch, and vocal parameters onto a different phonemic skeleton) or source substitution (e.g., resynthesizing a voice part, but performing it on a violin, or with the voice of an archetype, like Howard Cosell). In fact, Massalin has already shown how sheep and chickens can be persuaded to sing the *1812 Overture* [Massalin 1989] and the synthesis opportunities afforded by techniques such as LPC and sinusoidal approximations [Quatieri 1985] are well-known. These resources will certainly become available for content-based music recording and rendering systems.

Our software deals primarily with three control formats – MPU-401 MIDI data (which is raw MIDI information with time tags), piano data (which is a readable ASCII list of timed events), and a generalized event-list format (called *mu* data) that encodes MIDI and piano data as well as being general enough to accommodate other kinds of controlling information. Both MPU data and *mu* data can be dealt with in binary or printable form, and in addition, *mu* and MIDI files can be assembled into a score which is an event list of music fragments that also carries some graphical information. I will briefly discuss each type.

4.1 Piano Data

The Bösendorfer generates two kinds of sensor data: hammer velocities and pedal positions. These can be recorded as ASCII files giving time, sensor, and data value:

```

% cat promenade
time key value
3830 67 80
4365 65 73
4451 67 0
4872 70 72
4995 65 0
5354 72 67
5424 70 0
5599 77 66
5650 72 0

```

Middle C is 60; these notes are G-F-Bb-C-F (the beginning of the *Promenade* from Moussorgsky's *Pictures at an Exhibition*). Pedals are sensors 1, 2, and 3 (none shown). The soft and loud pedals generate 8 bits of data; the *sostenuto* pedal (middle pedal) is a binary switch. The piano also sends and receives MIDI format data through the MPU-401's, with a small loss of velocity resolution. In practice, the piano is used like a 2000dpi typesetter: draft work is done on synthesizers, and final copy on the piano.

4.2 MPU-401 MIDI Data

The Roland MPU-401 MIDI processors read and write MIDI data using a one-byte time tag and some number of bytes of MIDI data. The *time* is an offset in MPU clock ticks (240/second by default) from the previous event. MIDI data elements are typically one byte wide. Musical data can be read and written by this device using shell-level commands like `record` or `play`, and can be disassembled using the `da` command. For instance, the command “`record | da`” writes a straightforward listing of the raw data (to the left of the semicolon) and its disassembled information to the standard output:

```

% record | da
0 f9          ; 2.400 2 tcwme [ 0]      timing clock w/
                                         measure end

```

```

        f8          ; 3.600  3 tcip          timing clock
                                in play
66 90 3c 38 ; 4.110  4 kon   [60]=56 C4 key on
25   40 22 ; 4.295  5 kon   [64]=34 E4 key on
26   43 34 ; 4.485  6 kon   [67]=52 G4 key on
21   48 2d ; 4.650  7 kon   [72]=45 C5 key on
1e f9          ; 4.800  8 tcwme [ 1] timing clock w/
                                measure end

 4   4c 33 ; 4.820  9 kon   [76]=51 E5 key on
 d   43  0 ; 4.885 10 koff  [67]=0  G4 key off
18   48  0 ; 5.005 11 koff  [72]=0  C5 key off
 0   43 29 ; 5.005 12 kon   [67]=41 G4 key on
22   4c  0 ; 5.175 13 koff  [76]=0  E5 key off
 3   48 1c ; 5.190 14 kon   [72]=28 C5 key on

```

Tools like these and their implementation have been described in more detail elsewhere [Hawley 1986].

4.3 mu – A Performance-Level Data Structure

This is an event list data structure meant to encapsulate piano, MIDI, and other data. An Event is:

```

typedef long Time; /* 1000th of a second ticks */
typedef enum {
    tNull = 0,
    tInstrument,
    tNote,
    tControl, /* pedal, modulation, etc. */
    tSysEx, /* midi system exclusive packet */
    tEventList, /* e.g., pointer to a melody */
} Type;

typedef struct Event {
    struct Event *prev, *next;
    Time offset; /* offset from previous event */
    Type type; /* what kind of event */
    /* pointer to event of type 'type' */
    char *e;

```

```

    /* if NULL, use current instrument list */
    List *instrument;
    /* remaining fields for application use */
    long (*free)(), (*copy)(), (*print)(),
        (*read)(), (*write)();
    long addr;
    Time t;          /* holds absolute time */
                    /* true if event has been written */
    int  written:1,
        /* true if event has been read */
        loaded:1,
        /* true if event has been selected */
        selected:1,
        /* true if next ptr resolved */
        nextdone:1,
        /* true if prev ptr resolved */
        prevdone:1;
    Rectangle r;
} Event;

```

Other event types, like structures for phrasing and dynamics, or digital audio soundfiles, could be added. Event-specific data is kept in `Event->e`; these are usually simple structures:

```

typedef struct {
    short pitch, amplitude;
    Time duration;
} Note;

typedef struct {
    /* eg "kurzweil", "bosendorfer" */
    char *synth;
    char *voice;          /* eg "chif flute" */
                    /* indexes for internal use */
    int mpu, channel, program;
} Instrument;

```

The decision to represent a `Note` as a single event rather than as a pair of implicitly attached MIDI events makes subsequent processing easier. Similarly, using symbolic instrument names for instrument lists is essential for abstracting minimal orchestration

information. Future data structures for richer encodings, like violin or voice synthesis, will have to support more expressive data per note or instrument. Various utilities for processing this kind of data have been written (e.g., record, play, disassemble, etc.). This event-level structure can expand to hold cues for soundfiles and mixing control. It is clear that there will always be heterogeneous devices and control languages. Future representations will increasingly have to support rather general data. Eventually, the event list will not simply be a gesture recording, but will have to lend itself to a hierarchy of different representations to facilitate various kinds of computation.

5. Applications

Now we look at a few applications, some for analysis, and some for assembly. The `mbed` program is a data viewer that shows how interesting features in the input are exposed with a good digital lens; `nc` is a note-counter, which when combined with a histogram display or other filter can be used to analyze key, but which reveals other subtleties reflected by the statistical tonality of music; the `ntt` program implements melodic indexing (a function which will be sorely needed with very large music databases); the `nv` program is useful for filtering gestural data to find changes in voicing activity; the `patch` program begins to address the issue of instrumentation; and finally, `mudraw` is a fairly rich scoring program for composing and arranging. Most of these programs are also demonstrated with audio examples.

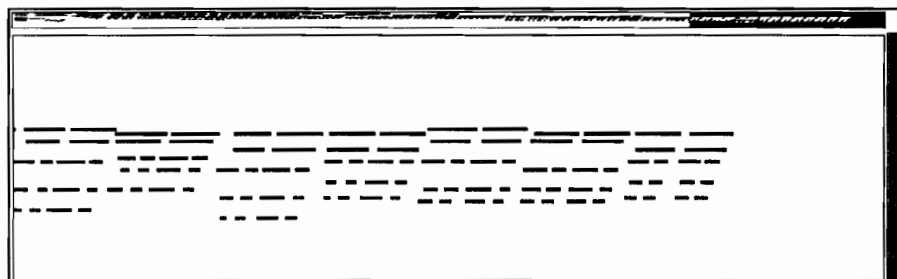
5.1 Key Analysis (`nc`, `mbed`, and `key`)

The note-count command `nc` (analogous to UNIX `wc`) produces a count of pitches (or cumulative pitch durations) in a stream of music. Folding pitches into one octave and looking at the 12-bin pitch histogram gives a good measure of traditional key. Here is the first prelude from Book I of the *Well-Tempered Clavier* by Bach:

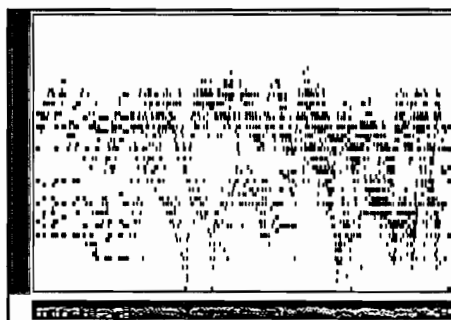
PRAELUDIUM I



[O23 – a performance of this piece, recorded and played on the Bösendorfer, is example 23 on the disc.] It has a piano roll display that looks like this:



This data viewer, mbed, does not show amplitude intensity (the Sun is only a 1-bit device), but does show timing and pitch information. The entire piece is drawn in the scroller, and interactively, the scroller behaves like a lens. The dark region slides freely over the piece; stretching it zooms in or out. This can be quite revealing. Here is a 5-minute piece in C minor zoomed all the way out:



The white, horizontal striations in the image are a result of the fact that in a tonal piece like this one, certain pitches are played frequently (like C and G, the tonic and dominant) and certain others (like C#) are played seldom, if ever. This suggests that the

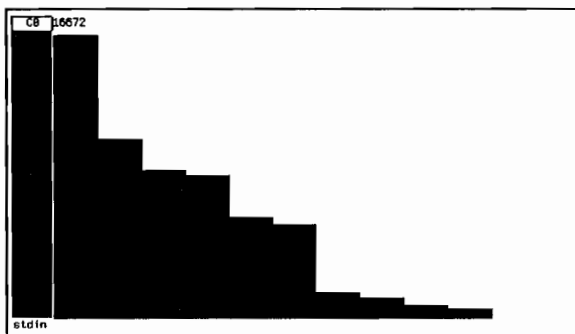
bias in the pitch “spectrum” contains meaningful information about the tonality.

The note count for the prelude in C major, showing number of pitch events and cumulative duration, is:

```
% nc -m prelude.1.01
prelude.1.01:
CO      0    110  16672
C#-1    1     4    247
D-1     2    73   9931
D#-1    3     6    767
E-1     4    63   8212
F-1     5    67   7974
F#-1    6    14   1709
G-1     7   113  15545
G#-1    8     4   1017
A-1     9    51   5701
A#-1   10    10   1389
B-1    11    41   5307
```

(-m counts modulo 12, i.e., folded into one octave). The pitch histogram looks like this:

```
% cat prelude.1.01 | nc -m | \
awk '{ print $1, $4}' | sort +1nr | histo
```



The prelude is in C major and the commonest pitches are C, G, D, E. This is typical for a tonal piece (tonic, dominant, dominant-of-dominant, and major or minor third). The key program

computes this histogram and looks for the traditional key-determining triad:

```
% key prelude.1.01  
C
```

Besides reflecting the traditional sense of key with about 80% accuracy (which is somewhat remarkable, since this is a first-order measurement, and independent of temporal order) this 12-number statistic points up subtler implications of pitch content [O24]:

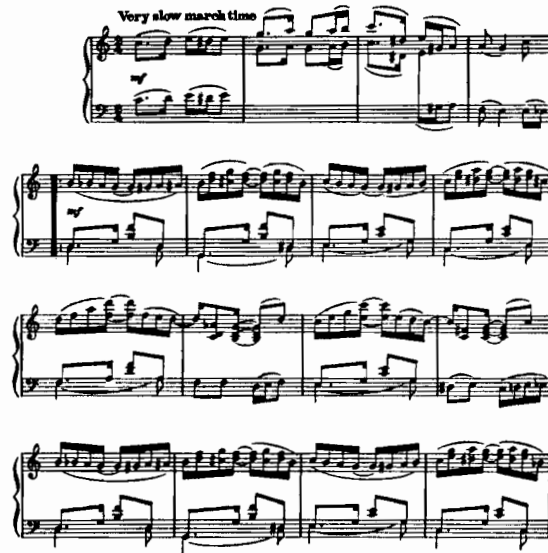
```
% cat solace | key  
F  
% cat solace | nc -m | ... | histo
```

“SOLACE”

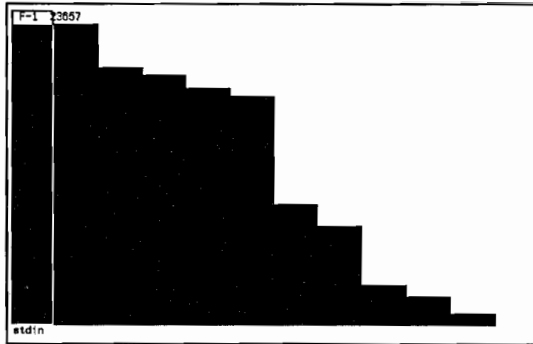
A Mexican Serenade.

By SCOTT JOFLIN
Composer of “Night Leaf Rag”

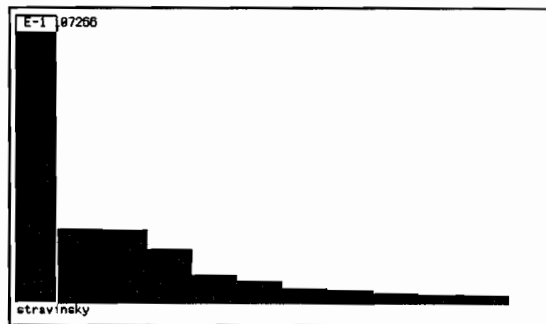
Very slow march time



Copyright 2009 by Academy Music Co. 123 W. 24th St. NYC. International Copyright Renewal.

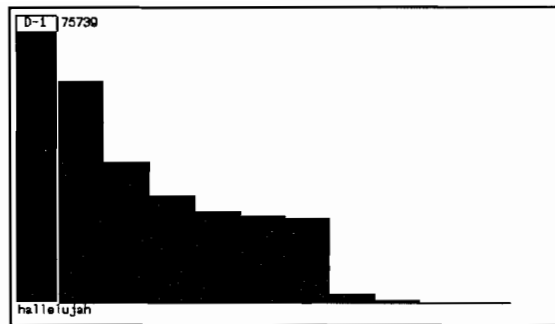


Solace, a rag by Scott Joplin, looks somewhat bitonal – it begins in C and ends in F, and produces a bimodal distribution that reflects the bitonality by showing a bulge instead of a steep slope. The pitches enter in this order: F, C, E, A, D, G... This results in an ambiguous key choice (either F (FC...A), a (CEA), or C (CE...G). In this case, more time is spent in F or a than C, which skews the final choice of key to F. In fact, it turns out that, more often than not, Joplin rags begin in the dominant and end in the tonic. Similarly, atonal music produces flattened distributions; and a filter like this, if applied in a sliding sampling window, would detect strong key changes. The first movement of Stravinsky's *Les Noces* is not particularly tonal, but pounds repeatedly on E's (to evoke ringing wedding bells); on the other hand, the *Hallelujah* chorus from the *Messiah* is resoundingly in D major:



E D# D A# F# A C# G# G F B C

Stravinsky – *Les Noces*



D A F# E G B C# G# A# C F D#

Handel – *Hallelujah* Chorus

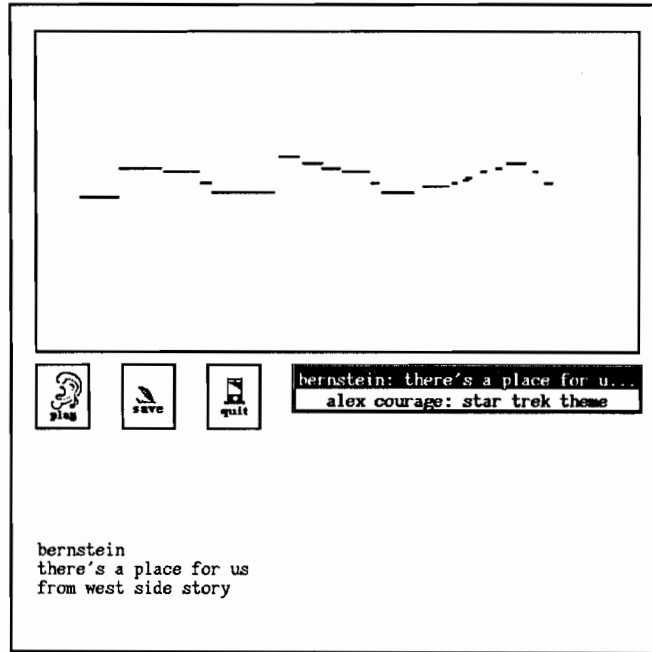
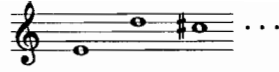
Moreover, an artifact of tuning systems and conventions of musical form is that a certain sort of music, a kind of musical *affekt*, is attracted by each tonal key on average and persists across centuries. For example, C# minor is the key of the *Moonlight Sonata*, of Rachmaninoff's *Prelude in C#*, of various bittersweet works by Chopin or Bach, etc., and these feelings come to mind when a musician thinks of that key. It would be quite trivial to sample such a statistic over a large database and discover correlations like these. Although a histogram in and of itself may seem like a low-level analysis tool in that it does not appear to exhibit the nonlinearities that are generally thought to be characteristic of high-level processing, it is in fact being applied here to highly nonlinear content data. Thus similarity metrics, when considered over a wide range of music, will be distributed in a relatively complex and meaningful manner. The nonlinearity is inherent in the data at this point, not in the processing.

Finally, it's interesting to note that `nc` is sort of a coarse-grained, octave-folded Fourier spectrum. In fact, by summing and octave-folding FFT's, one could perhaps make the same measurement from an audio signal. I have not tried this yet.

5.2 *Melody Recognition* – `ntt`

The `ntt` program⁴ takes a melody played at the keyboard and finds possible matches from a database of musical themes:

4. *Name That Tune*



It's a naive implementation of a profound function. In the database, each melody is represented as a string of relative pitch changes. The first few notes of *There's A Place for Us* from *West Side Story* would be C-Bb-A-F-D or four intervals, one up, three down, [10, -1, -4, -3], plus a starting tone (C). The input interval sequence is coded as a string of bytes around ASCII 'O', so that melodies are simple, sortable text strings. Rhythm is ignored. A melody played on the keyboard is quantized (attempting to remove embellishments), and then looked up literally (binary search) in the database. Since the encoding is relative, transpositions are found, but fuzzier approximate searches are not currently done. The problem is similar to spelling correction.

The idea that relative pitch ought to be a more salient feature than rhythm relates to the combinatorial fact that a melody of N notes can contain at least 12^N pitch combinations (if available pitch choices are restricted to just one octave; there are more in practice, of course). Rhythm tends to provide less variation, hence less information. Psychological studies of human memory

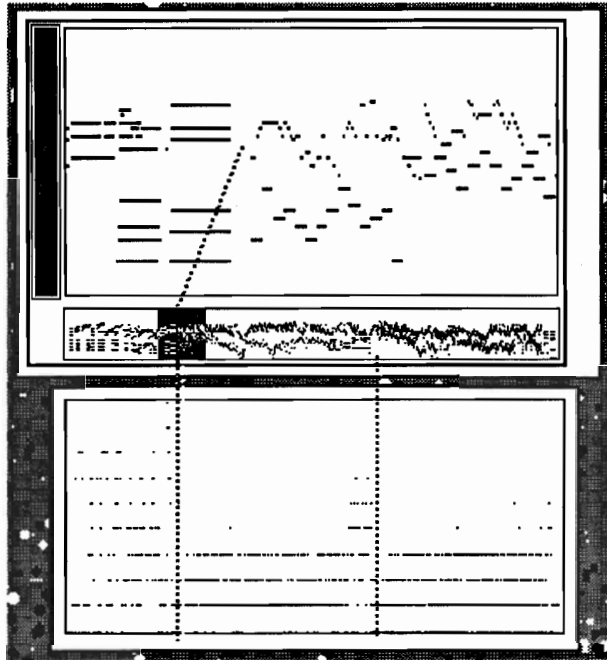
for melodies tend to support this. Several papers (e.g., in the *Journal of Music Perception*, c.f. Dewitt & Crowder [1986]) have discussed factors of pitch and rhythm in memory for melodies, and they generally indicate that while both rhythmic and pitch-relative representations are used, pitch contour is more important. One paper [Sloboda et al. 1985] discussed the memory of a musical *idiot-savant*, an individual who could play back a piano piece after only 2 or 3 hearings. Sloboda observed that gross and highly structural substitution errors occurred: for example, the savant mistakenly folded the rhythm from one phrase onto the pitches of another. This also tends to favor pitch as a more persistent feature. It seems that what is combinatorially expedient agrees with what is perceptually meaningful in this case.

It is clear that the matching could be done other ways – with regular expressions, proximity metrics, etc. – but also that the number of melodic strings in a seemingly large corpus is manageable. For instance, Brahms’ piano music runs to about opus 120, and even if there were 10 or so memorable melodies per opus we would still have only about a thousand short strings, but probably far fewer. (*The Dictionary of Musical Themes* contains only 10,000 [Barlow & Morgenstern 1948], 348 by Brahms.) Melodic indexes will become important as the volume of information in common music systems increases, but the amount of computation required to recognize one from a set of all melodies is likely to be relatively small. We also note that Barlow observed that even a short melody reveals enough of a “fingerprint” or “melos” to guess the composer, though we know of no attempt to quantify this.

5.3 Looking at Polyphony – nv

For keyboard music, measuring the number of voices currently active is also a useful statistic. The `nv` program does this:

```
% nv partita2.1
```

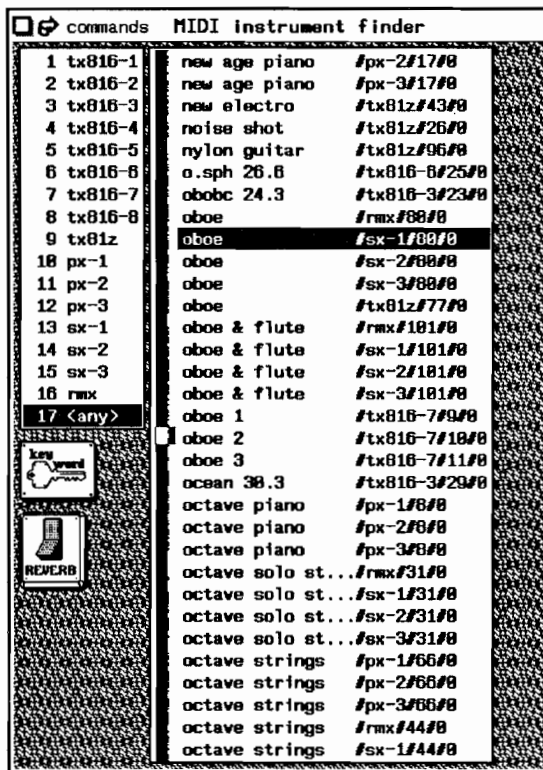


This locates cadences and other important changes in voice texture. This piece segments into three logical sections, which are delimited by cadences. (The lower frame corresponds to the entire piece in the scroller, and shows time on the x -axis, and average number of active voices in y). The cadences appear as an accumulated chord (lots of voices) followed by a short pause (no voices). One thing we have not tried yet is separating parallel voices into logical parts (e.g., by grouping events that are closest together in pitch-time space into a single stream). Note that `nv` slides a filtering window over the length of the piece, and uses an averaging window to compute values. In general, the choice of a width for this window is significant and depends on the tempo of the piece, which is not determined automatically.

5.4 *An Instrument Palette* – patch

The fact that literally thousands of discrete “instruments” (the synthesizer analog to organ stops) are present in the orchestra causes an amplified form of an old problem for composers: the

choice of instrumentation, and the disposition of orchestras and ensembles has been the subject of many treatises, as has the selection of organ stops (the “registration”). It is also common to create arbitrary new timbres through sampling or synthesis, and this requires a coherent mechanism for managing all this information. Any of these voices can be played from a central musical keyboard (the KX-88 MIDI keyboard controller), and managed with a palette-like application called patch:



This program allows users to map instruments to the keyboard by choosing them from menus of per-synthesizer or global contents, or by specifying a keyword. The instruments are tested and played on the keyboard, and patch information can be passed on to other programs. No knowledge of device level information (like MIDI channels) should be necessary. Scores contain parts that are played by instruments whose channel and program numbers are compiled at run-time. Available voices and their locations are kept in a single file:

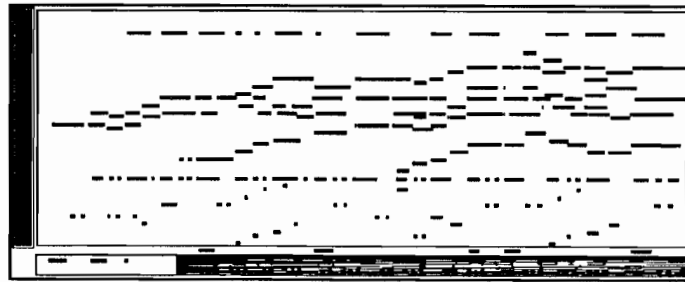
voice	synth	program#	mpu#
a bass/vibes	px-1	25	0
a bass/vibes	px-2	25	0
a bass/vibes	px-3	25	0

Library routines like `WhatChannel("bosendorfer")`, `WhatSynth(channel, mpu)`, `WhatProgram(channel, voice)`, etc., make it easy to look up instrumental attributes.

Although this deals with the immediate problem of finding one from among thousands of instruments, there are at least two missing ingredients here: first, the ability to map out instruments into some sort of timbral feature space (which implies a proximity metric that gives some indication of the similarity of two timbres), and second, a more formal language for computing from timbre. Among other things, the ability to find a timbre similar to some arbitrary timbre is what will permit intelligent synthesis systems to re-cast orchestrations to play well on local machinery. This remains a ripe area for research, as Wessel [1979] and Grey [1975; Grey & Moorer 1977] indicated.

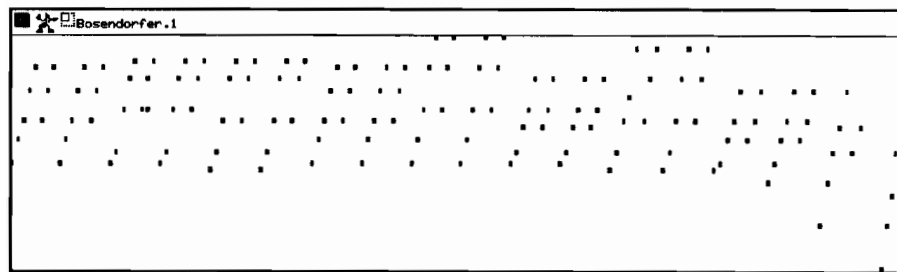
5.5 Scoring and Performing on Piano and Synthesizer

Performance scores for piano and orchestra can be assembled using an event-list editor called `mudraw`. Parts can be played from any keyboard, or generated by programs, and they appear as segments of event data located at various times in the score. From the scorefile, a performance list is compiled (for pure music, usually an MPU-401-format file with instruments resolved for the local patch setup). Here is a short arrangement of the *Pavane* from Peter Warlock's *Capriol Suite* [O25]:



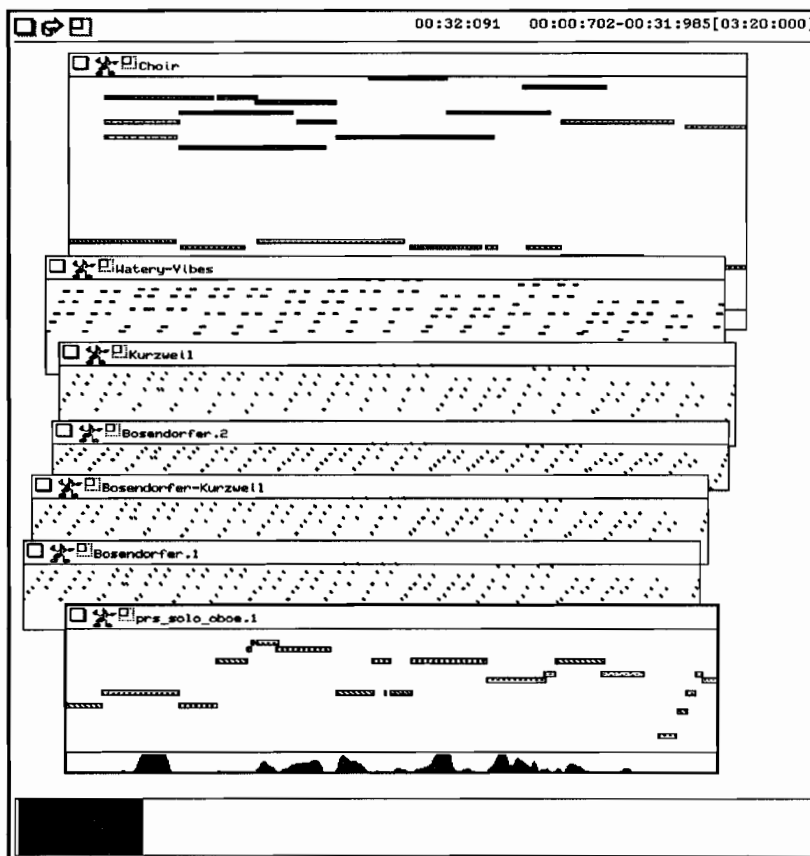
It is based on a late 16th century dance from Arbeau's *Orchesographie*, so I arranged it to sound like a Renaissance woodwind band. There are five or six instrumental tracks – flutes, bassoon, oboe, tambourine, drum – and the assembly process was, for each part, pick an instrument (from patch), record some data in mudraw by playing it on the keyboard, edit it if necessary (positioning it in the score, touching up wrong notes or articulation). This indicates the quality of synthetic voices which are available in present consumer synthesizers.

The next example is a multi-layer orchestration of the Bach prelude we showed earlier, scored for piano and synthesizers [O26], using choir, oboe, synthetic pianos, and a sort of chime. It bears some resemblance to the original, though relatively few musicians recognize it. In this case, the original piano performance was edited by removing the pedal information, and averaging (low-pass filtering) the relative time offsets and durations. After this editing, the piano part sounds like this [O27]:



This image is the part that appears in the graphical score. The titlebar icons are: ■ – close (hide), ▾ – pulldown the patch menu of thousands of voices, □ – reshape the part. *Vertical* reshape is a convenience, so parts can be collapsed but features remain visible. *Horizontal* reshape does a linear stretch of offset and duration values. This is a convenient way to adjust a

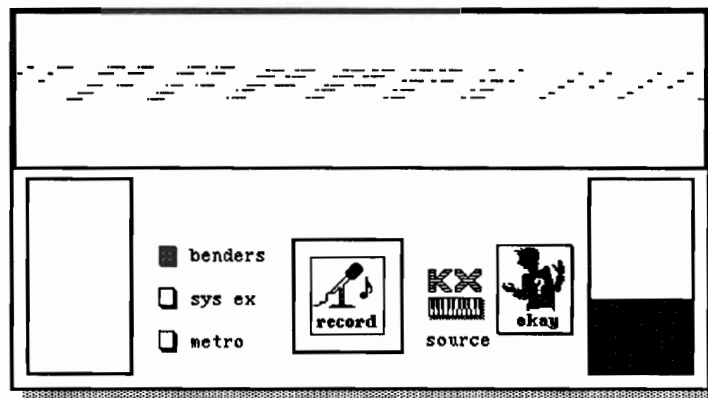
segment of music (or other temporal event stream) to fit a time constraint. By selecting all the notes in this part and copying them into a new part offset by an eighth note, a phased version of the original results [O28] (not playable with fingers on a piano). A synthetic piano, a choir part, and other voices [O29] are then added. The full score as it appears in mudraw looks like this:



Parts can be “wired” together into a group, so that making time changes in one component (shift or stretch) propagates to all elements in the group. Any set of notes, within or across parts, may be selected and edited. For note-level editing, besides cut/copy/paste, operations include: *play* the selection; *record* (overdub in synch with) the selection; *filter amplitudes*: set to an average or arbitrary level, louder or softer by a factor, apply a linear crescendo or decrescendo (amplitude values may be specified numerically or symbolically, *pppp* – *ffff*), and are filtered

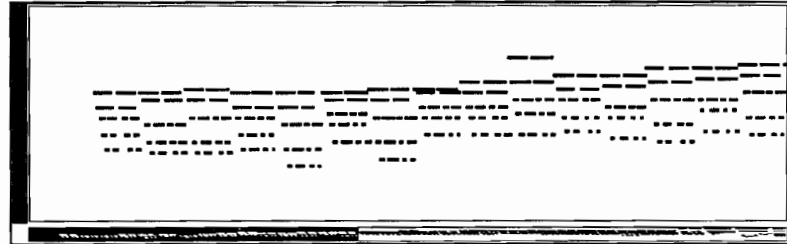
through device-specific map tables, since mechanical pianos and synthesizers all respond differently to keystroke velocities); *filter timings*, both relative offsets and durations: set to an average or arbitrary value, make faster or slower by a factor, apply a linear acceleration or deceleration, *legato*, *staccato*. It would be desirable to provide a compressor filter to ensure that deviations in amplitude remain within a certain range. There is also a function which filters the selection through an arbitrary external command – e.g., to embellish a note sequence with fractal effects. Transposition works this way (tr +1 octave), or by sliding the selection up or down with the mouse. A macro facility makes it easy to map these operations to keystrokes. A last missing ingredient is the ability to synchronize one part with another based on a sensible relationship (like beat correlation). This would require factoring the timing contour into a separate data structure (we currently simply filter the time values in place), and although this is not done now, it is a first step toward providing the capability to imprint an arbitrary performer's style onto the data. But it is most expedient to simply re-record a segment two or three times until the articulation is adequate.

The *record* dialog box shows the selection being overdubbed, if any, and provides switches to do mild filtering of the control stream input (e.g., ignore pressure or pitch bend information), select the source device (a Bösendorfer, a synthesizer control keyboard), etc.:



After the performance is confirmed, the new music appears in a part, placed in the score, where it can be edited. The next

example [O30] which was made by again taking the Bach prelude, filtering it with a pitch inverter (to flip the keyboard by mapping bass notes to treble, and vice versa):



This has a rather curious effect. Major chords become minor chords, traditional tonic/dominant relationships become modal harmonies, and the result is somewhat surprising.

As a final illustration, here is a longer piece, a segment of Liszt's *Totentanz*. [O31] is an excerpt of a studio recording made by Jorge Bolet with the London Symphony (London CD 414 079-2). [O32] is the synthetic version, arranged in a few hours using these tools. It is a concerto for full orchestra and piano solo, in the form of a set of variations on the Gregorian chant *Dies Irae* ("day of judgement"). Here is the score, and a graphical display of the first 60 seconds of performance data:

piano can mechanically play). The play dialog (and the record dialog) both compile a performance list (in this case, for the MPU-401's) from the current score information. This is an edited and enhanced version of what a live performer would play, and in some respects the Bösendorfer can play rings around a live artist. The synthesizers are not quite equal to the London Symphony, but the richness is sometimes remarkable, and this trend will continue. Our conducting and arranging is deliberately different – I prefer tempi and effects with more punch, have added percussion and choral parts to the score, etc.

6. *Future Work*

I plan to continue expanding the scoring application, *mudraw*. Some editing functions need refinement, and we are beginning to require editing of control channels (e.g., pedal or pitchbend data) and higher level operations. One of the difficulties with simple graphical editing of performance data is in shaping subtle effects of phrasing. Although it may seem at first that it would be useful to graphically sketch a curve for *crescendo* or *diminuendo* or *accelerando* or *ritardando*, or hand-edit continuous control streams like pedalling or pressure, these are actually extremely unnatural ways to contour a phrase. A better approach would function like the synthetic accompanists of Vercoe and Dannenberg: the user plays a selected part, accompanied by the rest of the system, and this re-performed version is replaced on the original. Attributes from the new performance (perhaps just the tempo, but perhaps amplitudes and/or pitches) could be selectively replaced over the original. In this way, the arranger molds the score by performing it into shape.

The event list handled by *mudraw* is capable of incorporating digital audio sound effects (played remotely by another computer) and client/server code for this is already in place. We will probably enhance the editor to include this.

In the analysis area, while dictionary lookup of melodies is easy, it is hard to isolate interesting melodies in an arbitrary polyphonic texture. This should be improved. Doing that requires polyphonic streaming (voice separation), which seems

manageable, followed by analysis of voice parts to find melodies. As with famous soliloquies, the most memorable part of a melody is at the beginning of the string, and applying this to groups of pitches that cluster closest in time usually results in a meaningful chunk. There has not been enough work with rhythmic statistics, generally, to usefully characterize tempo changes (it would be good to have a few statistics to indicate tempo, meter, and percentage of rubato), but again, the histogram approach will probably help here. The *ritardando* problem would be fun to study by comparing a computed performance to a live ones, to extract a tempo derivative. Our work suggests that this will be an extremely worthwhile thing to do, for it seems to be primarily the rhythmic contouring of a performance, and secondarily the amplitude modulation, that carry performer attributes. By abstracting this information it should be possible to encapsulate the style of given performers to some extent, so that algorithms can be made to render new works in the style of a particular performer. Until these areas are explored it will be difficult to build a metric that correlates musical data based on performance nuance or composer. Furthermore, the presence of a large repertoire of second-order templates for things like ritardando or articulation should make it possible to build this kind of analytical information into a rendering system (e.g., “cast the phrase to make it sound less like Schnabel and more like Glenn Gould”).

Finally, to help with all of this, I plan to accumulate an extensive database of on-line music here at the Media Laboratory to provide firmer ground for statistical work. In addition to gathering up performance data the way we do now, we also expect to begin using paper piano roll readers and polyphonic pitch extraction (from acoustical recording) as input channels.

7. Conclusions

A set of demonstrations has been presented that begin to show what can be done by computing along the range of musical information, from signal to content. The existence of MIDI is not a fluke. It indicates a trend in coding that pushes the computational fulcrum toward content, and the implementation of meaningful

interactions. Even seemingly unimaginative processing can do rather remarkable things when it is brought to bear on content-level data. In a sense, this is because useful imagination was already spent in casting the input into a more content-oriented form. This will eventually apply across all media, but music and audio information have already achieved considerable success. This work immediately suggests a personal instrument that combines:

- a critical mass of music – years of it – on a single data disk
- analytical tools that help uncover meaningful properties in the content
- rendered audio quality more like grand orchestras than simple upright pianos
- data-rich algorithmic composers and performers, synthetic accompanists, and other engaging interactive systems.

We have seen how seemingly elementary artifacts of high-level data, like intervallic content or histograms of pitch occurrence, correlate with rather deep human notions, like the identification of melodies or the “feeling” associated with a tonal key. Although properties like these may seem obvious in retrospect, their implications are hard to appreciate until they have been implemented and studied. This phenomenon applies to other information. For example, studies of the Brown corpus [Kucera & Nelson 1967] showed that relative word frequencies correlate well with literary “genre,” though large-scale text databases are still so uncommon that there has not yet been much need to apply this. Several years ago, Gabura discussed a composer-guessing algorithm that takes a strong cue from the frequency of harmonic change – e.g., the modulation of the pitch histogram over time [Gabura 1970]. It would not be at all surprising to learn that the pattern of cuts in a movie’s visuals correlates with content (e.g., more cuts and other rapid pixel-level change during action-filled scenes), or even with a particular moviemaker; or that the pacing of laughter identifies a standup comedian. One lesson here is that as we begin to operate on increasingly higher-level encodings of a signal, simple-seeming processing elements require more thought to apply and interpret. This is because we are no longer operating on a relatively uninformative low-level signal: we are measuring and modulating a

densely-coded content stream. What was once a numerical low-pass filter for low-level samples is now a rhythmic smoother, or a wrong-note finder. A trivial inverter transforms familiar music by Bach into “new” music with haunting, unfamiliar harmonies.

We have also noted a trend towards an instrument that synthesizes not one voice, but a grand orchestra. Traditional instruments seem to be fading away much as horses and carriages were supplanted by cars. This may seem heretical, but it is already true in major studios, and not without precedent. The player piano was once a booming pastime in American homes. In 1930, more than 2.5 million player pianos were manufactured and sold in this country. Great concert pianists, like Rachmaninoff, Hoffman, Horowitz, Paderewski considered piano rolls to be a legitimate and faithful recording medium. In fact, reproducing piano rolls were their recording medium of choice for about 15 years. But the player piano vanished almost overnight. By 1931 very few were made. It was pushed out of the living room by improved entertainment technology: radio and phonograph, primarily, but also television, telephone, and a little later, the Hammond organ. For that matter, the piano did an effective job of supplanting the clavichord and harpsichord, which earlier had eclipsed lutes and citterns. The move toward synthetic ensembles with thousands of timbral possibilities is a step along the same path.

People seem to want increased access to information, and more leverage for manipulating it. The piano in the 19th century functioned something like a bitmapped Macintosh interface to music, and with it, the user has enormous access to a rich literature. It lacks the full expressive control or the timbral diversity of an orchestra, or a violin, but it adequately carried most of the essential musical information into the hands of many people, and of course evolved into an art medium and a cultural icon of its own. The piano, like the Macintosh, also helped narrow the gap between “professional” technology and “amateur”: the same works that are performed by concert artists (or published by professional software authors) can be played in individuals’ living rooms, often using the same devices. For many years, the piano score was the common medium of communication for composers. Liszt ported many of Bach’s greatest organ works to the piano, and all of Beethoven’s symphonies, not to show off, but so that

mass musical audiences everywhere could gain access to it. This roughly parallels personal computers and third party software, although proprietary data formats prevent substantial software interchange. One always has to “port” an application. At the moment, the lack of a good device-independent interchange format and emulators are preventing exactly this kind of exchange for synthetic instrument scores. Synthesizers are too heterogeneous and disorganized to permit useful exchange of data, and instrument information is not abstracted (with MIDI) in a way that would make it easy to re-orchestrate a score optimally for a local device. Although music control information is so small that it clearly could be hidden in the cracks of audio CDs, or broadcast in a sideband of a radio or television channel to implement novel entertainment media, no publishing or datacasting is likely to happen until rendering devices are made reasonably homogeneous, and general enough to emulate orchestration information as well as simple note lists. It is likely that two key prerequisites for this will be reliable instrument identification and robust polyphonic transcription in a way that permits interesting resynthesis. Further, a variety of representations will have to be attached to the controlling information to foster a good family of software for manipulating it. For example, the representation would have to make it easy to cast a vocal part from voice to LPC to a phonemic list with a simple pitch-amplitude list, and back, because analysis and synthesis operations can be applied at many levels.

People can also be very stubborn about accepting new ideas. Someone once asked composer Edgar Varese why he refused to adopt the 12-tone serial style, popular with other *avant-garde* composers. Varese replied: “Just because you have invented the automobile, doesn’t mean I have to shoot my horse.” We grow terribly attached to knowledge or other possessions that represent substantial investment, and there is currently no replacement for the vitality or sense of humanity conveyed through a live performance. But Varese was also reported to have experienced an unusual thrill when, fairly late in life, he discovered a radical new representation tool – stacks of graph paper in the supply room at Bell Laboratories! New tools bring new insights, and future Beethovens, Liszts, amateurs, or AIs will have an interesting time

of it when personal orchestras and centuries of musical information become the norm.

Acknowledgements

This work would not have been possible without the help of many people:

- Yamaha corporation donated much of the synthesizer hardware
- Marvin Minsky loaned his Kurzweil synthesizer
- the Bösendorfer piano was purchased by special arrangement from Kimball International, and maintained by Hal Vincent and others from Kimball
- the IBM PC which controls the piano was donated by the IBM Scientific Center to our Visible Language group, and then given by Patrick Purcell there to us
- the Sun workstation was donated by Bill Joy of Sun Microsystems
- the NeXT workstation was donated by Steve Jobs of NeXT, Inc.
- Other Kurzweil synthesizers were donated by Ray Kurzweil
- the original MPU-401/Sun work was inherited from Gareth Loy at UCSD
- the VME MIDI interface was designed and built by Dave Cumming, working from plans provided by Don Jackson and Dan Steinberg at Sun
- the new MPU-401 device driver, and many low-level improvements were contributed by Dan Steinberg at Sun
- Peter Langston of Bellcore contributed much MIDI software

We also thank Robert Rowe for his patience and determination, and Dave Cumming for his work on hardware, signal processing, and driver-level code. Jim Turner from Kimball has helped us recently with the piano. Discussions with Andy Moorer, Peter Langston and Henry Massalin have always been thought-provoking. Advisors Andy Lippman, Barry Vercoe, and Marvin Minsky have allowed this project to exist, under the

research areas of Movies of the Future and Music and Cognition. Finally, Nicholas Negroonte deserves a special word of thanks for helping us mobilize some of the heavier pieces of equipment. Moving a 1400 pound piano is never easy, and without Professor Negroonte's personal support, it would never have joined the ensemble. Andy Lippman, Henry Massalin, Jim Davis, Peter Salus, and others provided comments which improved this paper.

References

- Henry Baird, Bibliography on Reading Music by Image Processing of Scores, AT&T Bell Labs, 2c-577, 600 Mountain Ave, Murray Hill, NJ, 07974, April 1987.
- Harold Barlow and Sam Morgenstern, *A Dictionary of Musical Themes*, Crown Publishers, New York, NY, 1948.
- William Buxton, Design Issues in the Foundation of a Computer-Based Tool for Music Composition, University of Toronto, CSRG-97, October 1978.
- William Buxton, William Reeves, Ronald Baecker, and Leslie Miezzi, The Use of Hierarchy and Instance in a Data Structure for Computer Music, *Computer Music Journal* 2(4):10-20, 1978.
- William Buxton, Sanand Patel, William Reeves, and Ronald Baecker, Scope In Interactive Score Editors, *Computer Music Journal* 5(3):50-56, Fall 1981.
- David Cope, An Expert System for Computer-assisted Composition, *Computer Music Journal* 11(4):30-46, Winter 1987.
- Roger Dannenberg and Josh Bloch, Real-Time Computer Accompaniment of Keyboard Performances, *Proceedings of the 1985 International Computer Music Conference*, Computer Music Association, pages 279-290, 1985.
- Roger Dannenberg and Dean Rubine, Arctic: A Functional Language for Real-Time Systems, *Computer Music Journal* 10(4):67-78, Winter 1986.
- Roger Dannenberg and H. Mukaino, New Techniques for Enhanced Quality of Computer Accompaniment, *Proceedings of the 1988 International Computer Music Conference*, Computer Music Association, pages 243-249, 1988.
- Shawn L. Decker, Gary Kendall, Brian Schmidt, Derek Ludwig, and Daniel Freed, A Modular Environment for Sound Synthesis and Composition, *Computer Music Journal* 10(4):10-20, Winter 1986.
- Lucinda Dewitt and Robert Crowder, Recognition of Novel Melodies after Brief Delays, *Music Perception*, 3 #3, pages 259-274, Spring, 1986.
- J. Gabura, Music Style Analysis by Computer, in Harry B. Lincoln, ed., *The Computer and Music*, Cornell University Press, 1970.
- J. Grey, An Exploration of Musical Timbre, Ph.D. Thesis, Department of Psychology, Stanford University, 1975.

- J. Grey and J. A. Moorer, Perceptual Evaluation of Synthesized Musical Instrument Tones, *Journal of the Acoustical Society of America* 62:454-462, 1977.
- Michael Hawley and Samuel Leffler, Windows for UNIX at Lucasfilm, *USENIX Proceedings*, Summer, 1985.
- Michael J. Hawley, MIDI Music Software for UNIX, *USENIX Proceedings*, Summer, 1986.
- M. Kassler, An Essay Towards Specification of a Music Reading Machine, in B. S. Brook, ed., *Musicology and the Computer*, New York: City University of NY Press, 1970.
- M. Kassler, Optical Character Recognition of Printed Music, *Perspectives on New Music* 11(1), page 250, Fall-Winter 1972.
- Henry Kucera and W. Nelson, *Computational Analysis of Present-Day American English*, Brown University Press, Providence, RI, 1967.
- Peter S. Langston, 201-644-2332, or, Eedie & Eddie on the Wire, *USENIX Proceedings*, Summer, 1986.
- Peter S. Langston, Little Languages for Music, in this issue of *Computing Systems*.
- Tod Machover and Joe Chung, Hyperinstruments: Musically Intelligent and Interactive Performance and Creativity Systems, *Proceedings of the ICMC*, 1989.
- Henry Massalin, personal communication, 1989.
- John Turner Maxwell, Mockingbird Manual, Xerox PARC, January 1982.
- John Turner Maxwell and Severo M. Ornstein, Mockingbird: A Composer's Amanuensis, Xerox PARC CSL-83-2, January 1983.
- Tom Nichols, Library of Congress Sound Recordings Collection, Personal communication, 1989.
- D. S. Prerau, Computer Pattern Recognition of Printed Music, Ph.D. Thesis, MIT, September 1970.
- D. S. Prerau, Computer Pattern Recognition of Standard Engraved Music Notation, *Proceedings of the Fall Joint Computer Conference*, AFIPS Press, Montvale, NJ, November 1971.
- D. S. Prerau, DO-RE-MI: A Program that Recognizes Music Notation, *Computers and the Humanities* 9:25-29, Pergamon Press, 1975.
- Thomas Quatieri and R. J. MacAulay, Speech Transformation Based on a Sinusoidal Representation, *ICASSP*, March 1985.

- Ted Ross, *The Art of Music Engraving and Processing*, Charles Hansen, New York, 1970.
- Robert Rowe, Implementing Real-time Musical Intelligence, *Computer Music Review*, to appear.
- Alan Ruttenberg, Optical Score Reading, MIT Media Laboratory, Master's Thesis, January 1990.
- Bill Schottstaedt, PLA – a Composer's Idea of a Language, *Computer Music Journal* 7(1):11-20, Spring 1983.
- Bill Schottstaedt, PLA – a Tutorial and Reference Manual, Stanford University CCRMA, STAN-M-24, December 1984.
- John Sloboda, B. Hermelin, and N. O'Connor, An Exceptional Musical Memory, *Music Perception*, 3 #2, pages 155-170, Winter 1985.
- Leland Smith, Music by Computer, *Journal of Music Theory*, pages 291-308, Fall 1983. See also *J. Audio Eng Soc* 20(1), January 1972.
- David Wessel, Timbre Space as Musical Control Structure, *Computer Music Journal*, 3(2):45-52, 1979.
- Barry Vercoe and Miller Puckette, Synthetic Rehearsal: Training the Synthetic Performer, *ICMC Proceedings* (1985), pages 275-278.

[submitted Aug. 2, 1988; revised Nov. 21, 1989; accepted Jan. 3, 1990]