Mockingbird: A Composer's Amanuensis

John Turner Maxwell III and Severo M. Ornstein

CSL-83-2 January 1983 [P83-00002]

Abstract: Mockingbird is a computer based, display oriented, music notation editor. It is especially focused on helping a composer to capture his ideas. It can play scores on the synthesizer as well as display and print them in standard notational form. It can accept both graphical input and input played on a synthesizer keyboard attached to the computer. In the latter case, the user must edit the music to turn it into standard notational form, and much of Mockingbird's interest lies in the methods by which this conversion is accomplished. The editor is highly interactive, presenting the illusion that the user can reach in and move elements of the score around as desired. This illusion is achieved by always showing the detail of the score exactly as it might be printed.

Categories and Subject Descriptors: I.3.2 **[Graphics Systems]**: Stand Alone Systems; I.5 **[Arts and Humanities]**: Music

Additional Key Words and Phrases: Graphical editors, music editors, computer music, music printing

XEROX

Xerox Corporation
Palo Alto Research Centers
3333 Coyote Hill Road
Palo Alto, California 94304

Introduction

Mockingbird is a composer's amanuensis, a computer program designed to aid a composer with the capture, editing, and printing of musical ideas. The purpose of Mockingbird is not to invent new music or to suggest variations to the composer, but simply to aid him in recording his own ideas by speeding up the notating process. Mockingbird is not a publisher's aid, although it does print music, nor is it a performer's aid, although it can play; it is strictly focused on the composer's need for a powerful scribe.

Mockingbird is an interactive music *notation* editor. It knows nothing about the rhythmic, harmonic, or melodic aspects of music except as they are represented in common music notation. To narrow the problem, we have concentrated on handling piano music. Mockingbird cannot presently handle orchestral scores or music for instruments that require their own notational devices.

Mockingbird was written in 1980, and it is somewhat surprising that no one had previously built such an obviously interesting system. We believe that there are two principal reasons: first, we had at our disposal, for the first time, an unusually powerful set of hardware and software facilities with excellent graphics capabilities, and second, we made a number of key decisions, discussed below, which allowed us to bypass some extremely difficult problems.

Overview

Mockingbird is a software package written in Mesa [1,2], an experimental language developed at the Xerox Palo Alto Research Centers (PARC). Mockingbird runs on a general purpose computer called the Dorado [3], which is an extremely powerful, experimental, single-user machine also developed at PARC. (The Dorado is now officially known as the Xerox 1132.) It has a 60-nanosecond instruction cycle, a large memory (typically two to eight megabytes of RAM), and an 80-megabyte disk. It also includes a large, high-resolution bitmap display, a keyboard, and a special graphical pointing device called a mouse. As the user moves the mouse around on the table beside the keyboard, a cursor (pointer) moves correspondingly on the display to indicate the mouse position. When Mockingbird runs, it presents a picture of a score with staves, notes, beams, etc., on the display. The mouse provides the mechanism by which the user can point at locations within the score or at particular items, such as notes, to which some action is to be addressed. The mouse has three program-readable pushbuttons on its top which are used for issuing commands to the program—sometimes in conjunction with keyboard keys.

The graphics facilities, high speed, and large memory make the Dorado a particularly suitable tool for music editing. The only special hardware we provided for Mockingbird was an interface to a Yamaha CP-30 electronic synthesizer. The Dorado can sense key positions and simulate key strokes on the synthesizer. Thus, the synthesizer can be used to "play in" music and to allow the computer to "play back" music without having to synthesize the sound waveforms by program. The setup is shown in Figure 1. Not shown is an experimental high-resolution, computer driven, rasterscan laser printer which Mockingbird uses to make hardcopy pictures. The information for these pictures is sent over an Ethernet [4] connection from the Dorado. Figures 3, 5, and 7 illustrate Mockingbird's hardcopy output (reduced 25% for printing in this document).

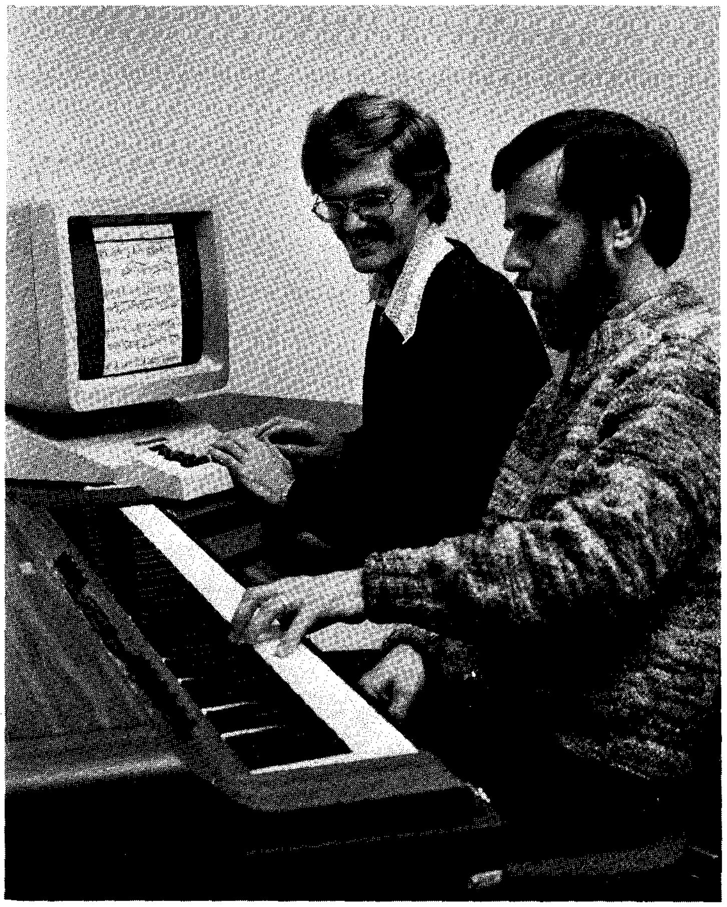


Figure 1: Mockingbird: The User's Set-up

In addition to these hardware resources, Mockingbird relies heavily on a general purpose graphics software package [5] that provides simple commands for displaying characters and drawing both lines and curves. It also provides a common interface for both displaying material on the screen and printing high-resolution hardcopy.

Mockingbird is designed to handle standard piano music notation. It knows how to deal with notes, rests, accidentals, beams, chords, ties, grace notes, n-tuplets, time signatures, key signatures, clef indications, ottava, and a variety of different kinds of measure lines, embellishments (mordents, etc.), and sheet layouts. Mockingbird deals with these elements not only in the graphics but, where appropriate, in the playing, as well. Mockingbird understands about the key of a piece and will propagate and properly suppress accidentals.

Despite its sophistication, the reader should remember that the Mockingbird editor is only a "research prototype." Many features are still missing and, in general, we did only enough to demonstrate feasibility. For example, there are numerous notational devices that we never got around to incorporating such as rolled chords, staccato markings, fermata, and so on. Furthermore, although Mockingbird can handle ties, it cannot handle the more general slurs, nor can it display text such as lyrics or tempo markings. We do not feel that the addition of further features would present any insurmountable problems or violate the premises from which we proceeded.

Important Decisions

We feel that Mockingbird's success is largely dependent on the decisions we made in the following five critical areas.

Amanuensis vs. Automatic Transcriber • We decided not to try writing a program that converted synthesizer keystrokes directly into a score. We made this decision for a number of reasons. First, we weren't sure that it could be done for the class of music we were interested in. Rather than pursue that question, we wanted to produce a tool that worked. Second, we knew that an editor would be needed anyway, both to correct mistakes and to satisfy the composer who did not want to use the synthesizer keyboard to enter material. So, instead of a recognizer, we built an amanuensis or scribe, which provides a human transcriber with powerful editing tools. Our strategy was to build the editing tools first and then work on automatic heuristics to augment the editing process. The editing tools can assist either in performing the conversion from played input to score or in entering scores graphically.

Data Structure • We believe that one of the most important decisions we made was the choice of data structures. Mockingbird treats music simply as a sequence of events. This allows us to simultaneously handle raw ("played in") material and more finished ("structured") material. Furthermore, it is convenient for presenting the material in its various external manifestations—displayed, printed, and played.

User Interface • In Mockingbird, rather than doing a lot of typing on the computer's keyboard, the user operates directly on the picture of the music that appears on the screen. To do this, he makes heavy use of the mouse. This approach is facilitated by a strong correlation between the internal representation of the music and its visual display (as well as how it is played). All of the elements of the data structure are displayed, and everything shown on the screen corresponds to some part of the data structure. If the user moves something on the screen, the data structure is immediately updated to reflect it; if the data structure is changed, the screen is immediately repainted. Not only is the picture faithful to the data structure, but so is the synthesizer "performance." For example, if the user puts a trill marking on a note, Mockingbird will trill when playing it.

Piano • Music notation is very extensive and non-uniform—instruments present individual requirements and employ special notational devices. By choosing to focus our attention specifically on piano music, we limited the scope of our ambitions to a manageable size. The piano was an obvious choice as it is an instrument frequently used by composers in trying out their ideas. Furthermore, because of its keyboard structure, it lends itself naturally to connection as an I/O device to a computer.

Voices • Music is broken "vertically" into separate parts or voices. Such partitioning is obvious in multiple-instrument music, but it is also present in piano scores as an essential structural feature. The recognition of this fact, and its explicit representation in Mockingbird, greatly facilitates the editing and formatting of scores. This topic is further discussed below.

The Editor

Mockingbird consists of a number of functionally distinct parts integrated into one editor. The editor allows the user to record, edit, play, and print a single piece of music. Commands are issued by making selections and typing characters. Some commands are invoked by pointing the mouse at an object on the screen and clicking a button. When the user has finished with the piece of music, he can name it and file it away in the Dorado's filing system. Later it can be retrieved by its name.

As it appears on the display, a score looks like a piece of sheet music (see Figure 2).

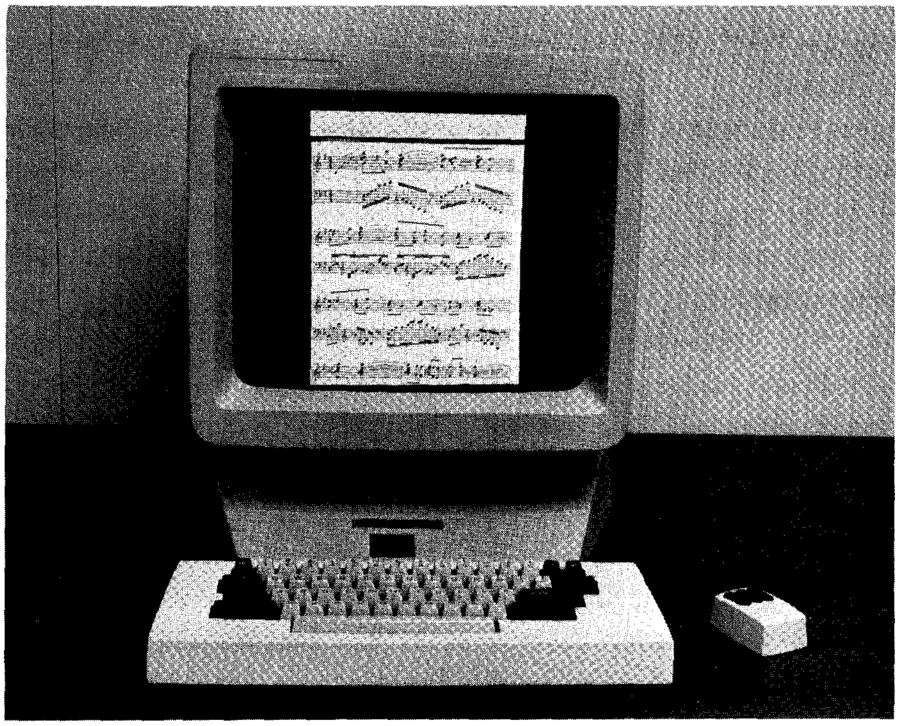


Figure 2: Computer Display, Keyboard, and Mouse

There are usually four to six staff sets (lines), each composed of two to four staves. At the left of each staff is a clef sign and an appropriate key signature. Scattered over the staves are notes, chords, beams, measure lines and other symbols commonly found in music.

Only about a page of the score can appear on the screen at a time, so there are commands that allow the user to look at different sections. "Scrolling" causes the current section of the score to be moved up or down so that neighboring lines can appear. The user can scroll as little as a single line or as much as an entire page. "Thumbing" allows the user to jump to an arbitrary point in the score. To thumb, the user specifies approximately how far into the score he would like to be, and the program moves the display to that point. Both thumbing and scrolling are accomplished by moving the cursor into a special "scroll bar" area at the left of the score and clicking one of the mouse buttons.

The score can be edited with Mockingbird just as documents are edited with word processors. The usual paradigm for making an edit is to "select" some portion of the music and then to issue a command. The command will apply only to the selected portion. The display is updated immediately to reflect the changes.

Selection

A user may select either a contiguous section of the score or an arbitrary collection of individual notes. Both types of selections are made by moving the mouse over the desired objects while holding down a mouse button.

If the user selects a section of the score, Mockingbird indicates that section by displaying it in reverse video (white on black instead of black on white). The section may be as small as a portion of a measure or as large as the entire score. It encompasses all of the notation that appears on all of the staves. Section selection is typically used for coarse editing operations such as copying, deleting, etc. However, it may also be used to apply a function to all of the notes in a particular region, such as designating them all to be sixteenth notes.

Mockingbird indicates individual note selections by painting the selected note heads grey. An individual note may be selected by pointing at it with the mouse and clicking the left mouse button. Notes may be collected into a selected set for combined action either by a series of individual mouse clicks or by sweeping the mouse over the note heads while holding down the mouse button. Notes accidentally included in a selection may be deselected by using the middle mouse button.

Since selection persists after execution of a command, it is possible, with a single selection, to issue a succession of commands which all apply to the same material.

Voices

In order to play notes at the proper time and to place them correctly within the score, it is necessary to know where they occur within the rhythmic structure of the piece. Measure lines provide reference points from which rhythmic position is measured. For music which has only one voice, one starts at the beginning of the measure and counts forward through the various note and rest values until arriving at the note of interest. Its position in the measure's rhythmic structure is then given by the sum of the time values of these preceding items. If the music has several parallel voices with differing time values, one must know which notes belong to which voice in order to count forward properly. Voicing thus provides a lateral indication of what goes with what.

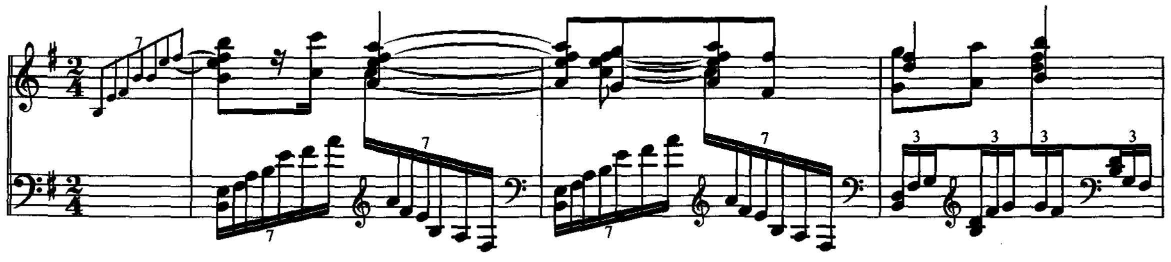


Figure 3a: A Segment of a Piano Score

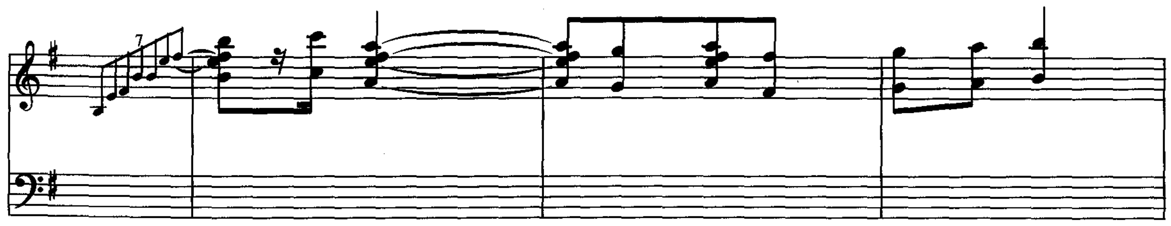


Figure 3b: The Same Segment Showing Only Voice 1

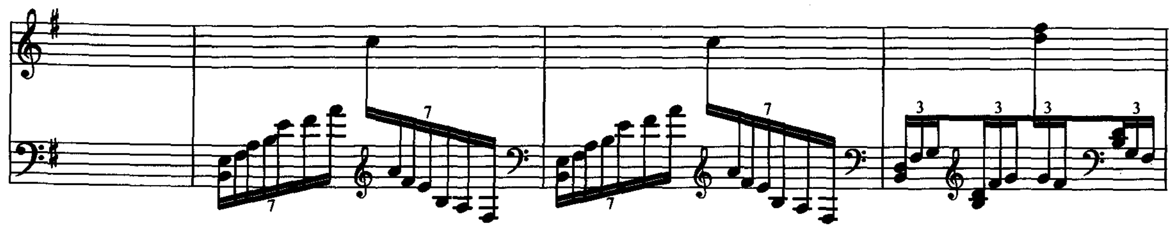


Figure 3c: The Same Segment Showing Only Voice 2

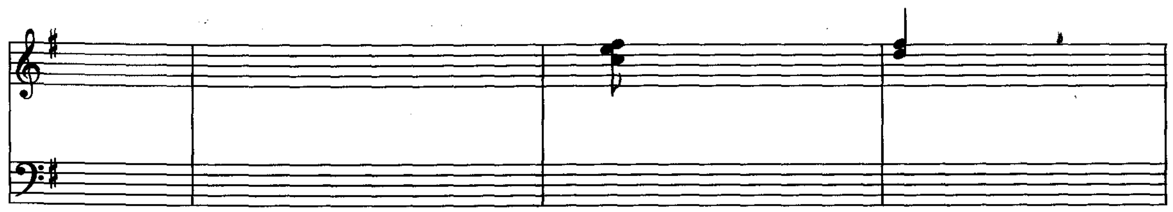


Figure 3d: The Same Segment Showing Only Voice 3

Figure 3a shows a section of a full piano score whereas Figures 3b-3d show its separate voices. To find out when the C octave occurs in the second measure of Figure 3b, add the prior eighth chord and sixteenth rest and conclude that the octave comes three sixteenths into the measure. Now consider the situation in the third measure of Figure 3d. Here a voice commences within the measure rather than at its beginning, so there is no way to count forward to it. Instead, its rhythmic position is determined by its synchrony with an element of another voice, in this case, the octave in voice one. (To enhance visual clarity, the chords are slightly separated in the full score shown in Figure 3a. However, they are, in fact, rhythmically synchronous.)

In that same measure, consider the effect of erroneously placing, say, the sixth note of voice two in voice three. The effect would be significant: first, the note would be incorrectly located (played) at the mid-point of the measure (i.e., one eighth after the chord in voice three), and second, due to the omission in voice two, all the ensuing notes of that voice (up to the next reference point) would be located slightly earlier than they should. Mockingbird contains a mechanism to assist the user in finding such errors. Upon request, the program will check to see whether the rhythmic "time" of every measure is properly filled by the notes and rests of the voices it contains (with due allowance for parallelism of voices). Measures that don't "add up" correctly will be marked with a stipple pattern on the display.

The voicing of a piece of music is hence an important part of its underlying structure which is revealed by the way in which the score is drawn. Thus, voicing in a piano score is indicated by such clues as shared beaming, chording, stem direction, and staving. The human reader uses these clues, together with vertical alignment within the score, in determining how to play the music, i.e., when to play the notes, which hand to use, etc.

In Mockingbird, the problem is the converse one of determining what the finished score should look like, starting from raw notes that have neither explicit voicing nor any of the structural clues of a completed score. Everything must be determined from scratch: note values, chords, staving, beaming, rests, ties, etc. One of our earliest insights was that if we could once determine the voicing of a piece, it would greatly facilitate determination of all of these other features. In studying existing piano scores, we saw no general way to infer voicing automatically, and so instead decided to give the user explicit control over its definition. Hence, Mockingbird includes commands for assigning notes and rests to voices and for indicating synchrony between the elements of different voices.

Once a piece has been voiced, the user can designate a particular voice for viewing, in which case, the notes of that voice appear in black on the display while the rest of the score appears in a light grey for reference. In this mode, the user can access only items within the designated voice (i.e., area selection will select only the notes in that voice, and individual notes in other voices cannot be selected). Any editing commands issued when viewing a single voice will thus affect only that voice; the part displayed in grey is not affected.

Editing Commands

Mockingbird's editing commands include assigning note values, assigning notes to voices, transposing notes, changing their stem direction, changing their spelling, and changing the staff on which they appear. In addition, various elements of the score, such as notes, measure lines, and time, key, and clef signatures, can be individually deleted or picked up and moved to a new location with the mouse. There are also commands that group notes together with beams, chords, or slurs. Many of these commands work with either note or section selection. For instance, the user may transpose a single note down an octave or an entire voice up a fifth. The first of these actions would be accomplished by selecting the note and issuing the "transpose" command. The second would be accomplished by designating the desired voice, selecting the entire score, and issuing the command.

The user may also rearrange large sections of the score in a "cut and paste" manner. To replace one section of the score with another, the user selects the section to be replaced (the primary selection), then the section to be copied (the secondary selection), and issues the replace command. The primary selection does not have to be the same size as the secondary one. If there is no secondary selection, the resulting operation is a deletion. If the primary selection merely points at "empty space" in the score, the resulting operation amounts to an insertion.

In addition to changing the structure of the music, the user can change the way it appears on the sheet. For instance, the number of staves for each line can be changed on a line-by-line basis. The user can switch a staff's clef in the middle of a measure or designate a section to be displayed in ottava notation. Changes in key and time signatures can also be inserted within the score wherever necessary.

The user can add new material to the score by picking up items from a pop-up menu that can be made to appear under the cursor (see Figure 4). The menu includes a number of small symbols (called *icons*) representing various elements of the score. There are icons for a note, a rest, bass and treble clefs, several kinds of measure lines, and a variety of markings such as trills, accidentals, etc.

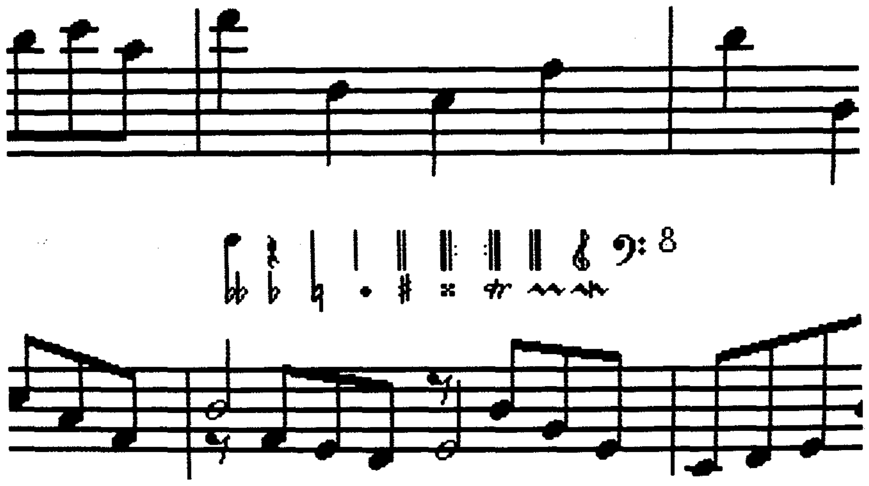


Figure 4: Photograph of the Display Screen Showing the Pop-Up Menu

As the cursor is moved over the menu, its shape changes to correspond to the icon immediately beneath it. When the mouse button is released, the cursor retains the last shape. The user can then insert instances of that icon by pointing to a place in the score and clicking another mouse button. (Mockingbird automatically selects the inserted note or rest for the user's convenience. Thus, the user may immediately issue commands that affect the note.)

The Synthesizer

The synthesizer is both an input and an output device. As an output device it can be used to listen to music stored by Mockingbird. This is especially helpful in proofreading scores. Mockingbird "reads" the score and plays it by simulating key strokes on the synthesizer. As the synthesizer plays the notes, a pointer tracks the performance on the displayed score. Mockingbird's rendition handles polyphonic music correctly, taking into account such things as grace notes, n-tuplets, trills, ottava, and metronome markings. Thus, the composer may listen to what he has written. Although the performance sounds a little mechanical, it is sufficient for catching erroneous note values and pitches. Moreover, the music can be played at double speed for rapid scanning or half speed for careful listening.

As an input device the synthesizer is used to capture music played by the composer. As the user plays, Mockingbird "watches" the keys and records when every note was struck. Music in this form is displayed as a note head time plot, which we call a "piano roll," illustrated in Figure 5a. Mockingbird chooses default staffing and spelling. Figure 5k shows the final score for this same section of music.

When playing music, the composer isn't restricted to a single melodic line, nor must he follow a metronome; he may play whatever he wants as freely as he wishes. Mockingbird captures his idea in a rough form that, although a far cry from a standard score, nonetheless contains enough information to reconstruct his original intent. At this point, the composer may go on to capture more music, or start transforming the piano roll into a score by editing it.

Raw piano roll material can be mixed in freely with standard music notation, both on a measure-by-measure basis and within a single measure, as shown in Figure 5e. All of the commands that apply to standard music notation can also be applied to the piano roll or to mixed sections. Thus, the composer can rearrange material, put notes into different voices, specify the durations of the notes, and add structural elements such as beams and chords. The ability to mix piano roll and standard music notation gives the user a lot of freedom; he can work on the score in whatever order pleases him. Mockingbird even plays correctly across the boundaries of mixed sections of piano roll and standard music notation.

Another feature of Mockingbird is that it is possible to "play against" previously entered material. While Mockingbird plays, the user can play along with it on the synthesizer. The combined product is heard as Mockingbird records the new notes, simultaneously merging them with what it is playing. This allows the composer to build a piece one voice at a time or to lay in new material over an existing score. It also allows him to construct music that cannot be played by one person on a standard instrument.

Converting Piano Rolls

Figure 5a shows some source material recorded by Mockingbird directly from the synthesizer. Such raw piano rolls are hard for humans to read; there are no key or time signatures, or measures, chords, or beams to group things, nor have the notes been separated into voices. Because part of the composition process includes specifying such syntactic structuring, it was necessary for Mockingbird to go beyond piano roll notation. The process of converting from piano roll to standard music notation is not handled automatically, but rather involves the user. However, Mockingbird does provide a number of heuristics that assist in the transformation.

Figure 5 shows a typical succession of steps for turning some piano roll input into a final section of score. As we discuss this process, it is important to remember three things: first, the particular sequence of steps shown in the example represents only one order in which the job can be accomplished; second, at any point the composer can stop to enter further material; and third, playing on the synthesizer is not the only method of entering music into the editor. At any point, the composer can use the mouse and menu to add material since Mockingbird allows piano roll and standard music notation to coexist.

The first step is one of alignment. In order to remove the inevitable imprecision in playing notes that should be simultaneous, the user can apply a heuristic that runs through the piano roll and aligns notes that occur very close to the same time. Figure 5b shows the results of that step.

Typically, the user provides key and time signatures in the next step, as shown in Figure 5c. Assignment of key signatures not only allows for proper display of the signatures themselves, but also enables Mockingbird to spell notes properly, following the usual rules for propagating the effects of accidentals within measures. Mockingbird's spelling algorithm for accidentals does not correctly follow the rules of harmony; instead we use a simple algorithm that is roughly correct and provide means for the user to modify any spelling he wishes. Assignment of time signatures causes them to appear in the score and enables Mockingbird to check for certain timing violations (e.g., the assignment of four quarter notes to a voice in a 3/4 measure).

Next, the user might enter measure lines. He can do this by picking up a measure line icon from the pop-up menu and depositing copies at suitable places into the piano roll. Alternatively, the user can tell Mockingbird to play the piano roll back on the synthesizer, and as it does so, he can "beat in" measure lines simply by striking the keyboard's spacebar on the first beat of each measure. Errors can easily be corrected by moving, deleting, or inserting measure lines as appropriate.

Next, the user would normally assign notes to different voices. An individual note is assigned to a voice by first selecting the note and then indicating which voice it belongs to. More typically, collections of notes are simultaneously voiced by selecting them together and then issuing a single voicing command. Similarly, notes can be moved to a staff different from the default assignment. Figure 5d shows the score with these steps completed.

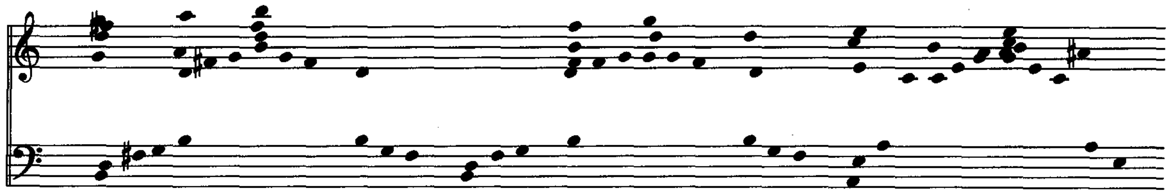


Figure 5a: Piano-Roll Input Directly from the Synthesizer Keyboard

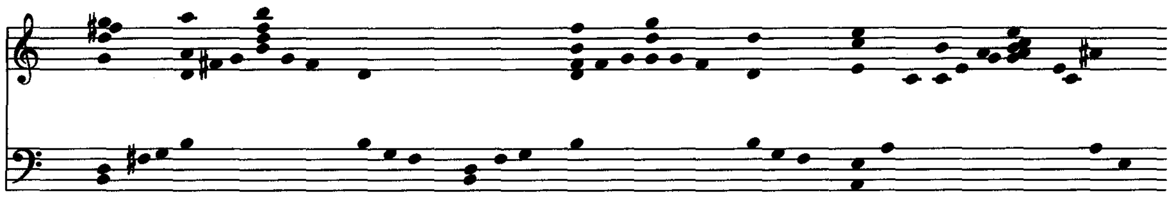


Figure 5b: After Alignment

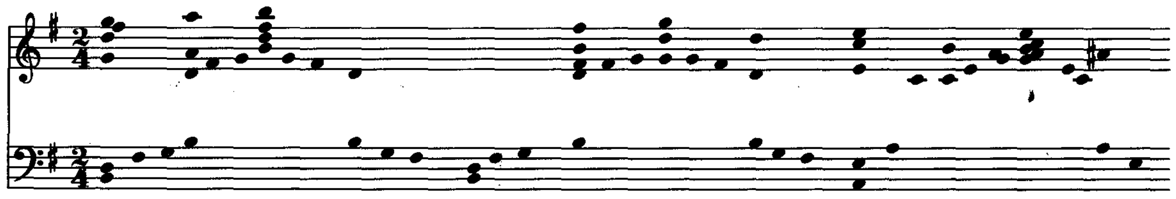


Figure 5c: With Key and Time Signatures Added

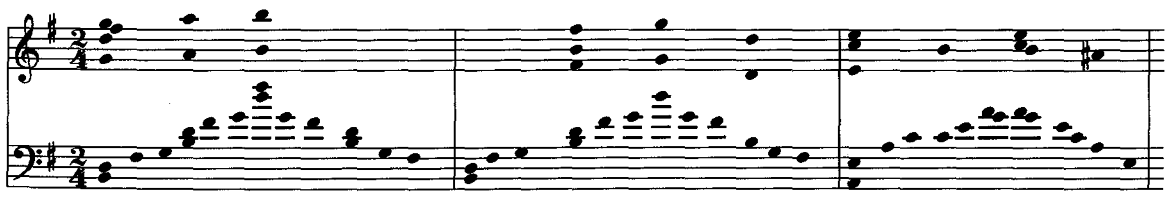


Figure 5d: With Voicing and Measure Lines Inserted

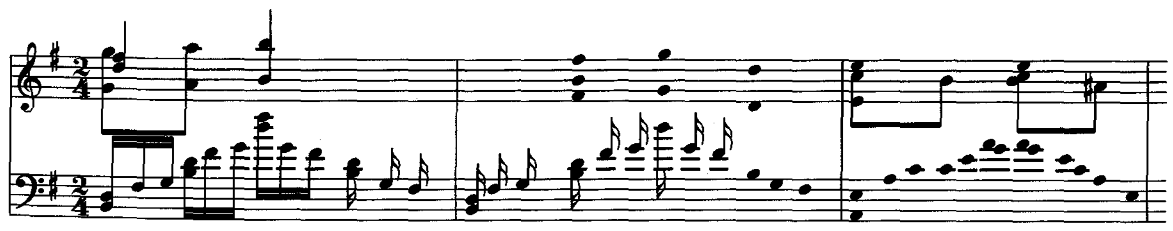


Figure 5e: With Some Time Values, Beams, Etc., Added by the User

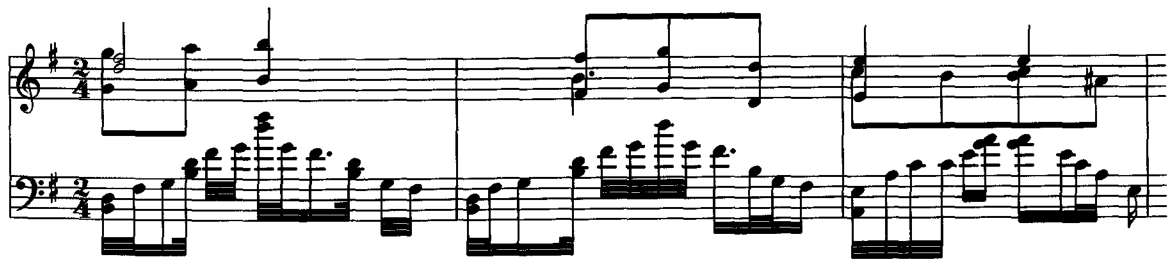


Figure 5f: Results of Timing and Beaming Heuristics

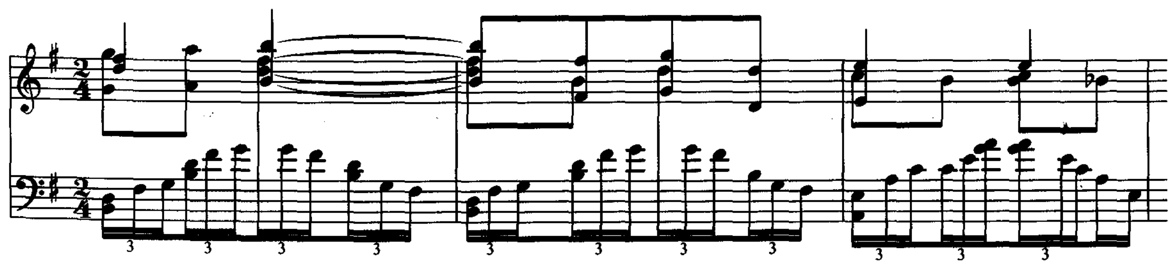


Figure 5g: After Correction by the User

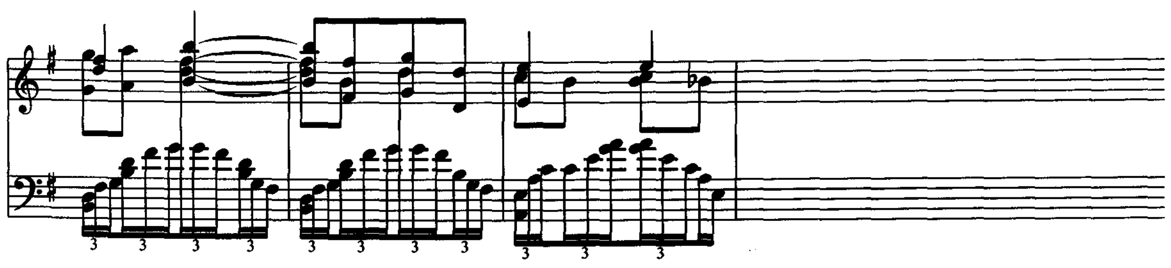


Figure 5h: Tight Compression by the Justifier

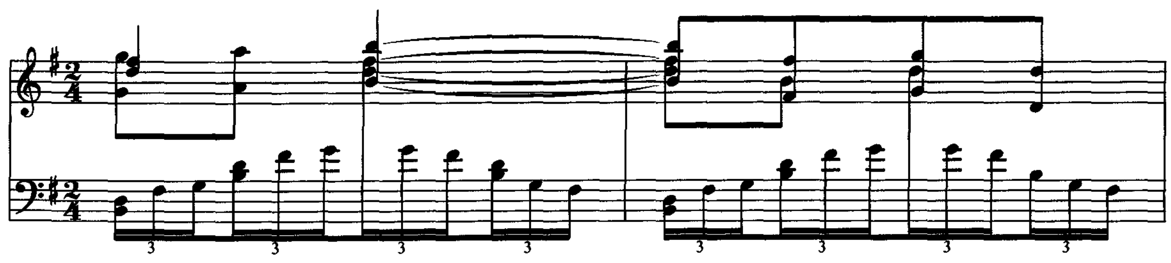


Figure 5i: Justified with Too Low Density

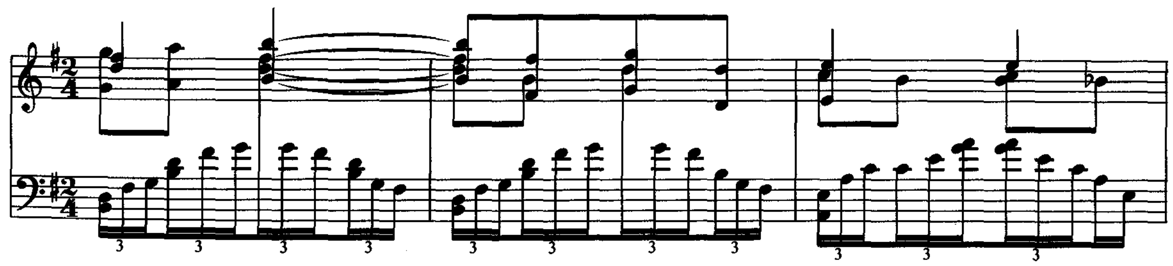


Figure 5j: A More Suitable Density

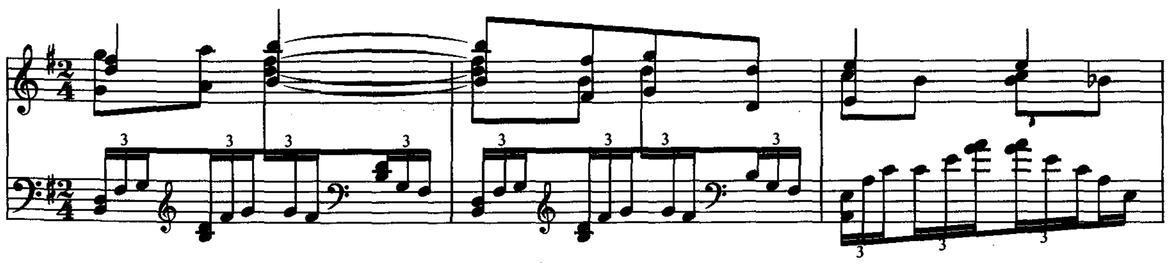


Figure 5k: After Final "Hand" Touch-Up by the User

At this point, the user can go through the score, manually assigning time values to the notes and designating chords and beams. Figure 5e shows this process partially completed. However, Mockingbird has a number of heuristics to help with these tasks. After the user assigns notes to their proper voices and gives a time signature, he may ask Mockingbird to guess the time values of the notes, group them into chords, and assign beams. Although the heuristics used are only about 80% accurate, they save the user a lot of work. In addition, what remains is easier for the user to deal with, since it is in a more familiar form. Figure 5f shows the result of the heuristics applied to figure 5d. (If we had started with 5e, the heuristics would have taken advantage of the information that the user had already provided.) Mistakes made by the heuristics can be found by inspecting the score or listening to it through the synthesizer. The user then fixes the mistakes and adds more structure. Figure 5g shows the resulting score. The combination of simple heuristics and easy editing is central to the concept of Mockingbird as an amanuensis.

By the time these steps are completed, the piano roll has become a score containing all the basic information. The only thing that remains to be done is some tidying up as discussed below.

Justification

A particularly powerful command is the one that "justifies" a sequence of measures within the score. (The user selects some area of the score, and the justifier locates the nearest measure lines.) Justification involves several things: making the voices consistent relative to one another, laying out the graphical elements of the score in an aesthetic arrangement, and making sure that each line of the score contains complete measures. Justification is concerned only with the *horizontal* placement of objects; details such as the height and tilt of the beams are outside of its domain. Furthermore, it doesn't disturb structural elements such as stem directions and staffing.

The justifier starts by going through each measure and making sure that all of the voices are consistently ordered relative to one another. Two voices are inconsistent if each adds up to the time signature, but when taken together, allowing for alignments and spacing, they appear to add up to more than the time signature. Figure 6a shows an example of such a situation. The justifier moves notes around to correct matters, as shown in Figure 6b.

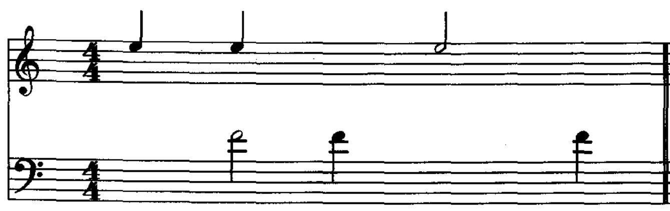


Figure 6a: Two Voices Improperly Aligned

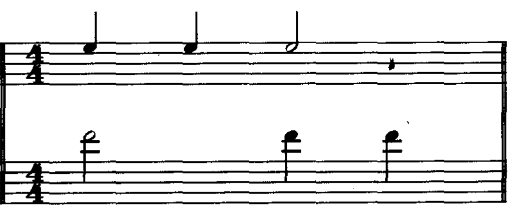


Figure 6b: After Correction by the Justifier

Next, the justifier redetermines the horizontal placement of the graphical elements of the score. The horizontal spacing is based on the types of elements (measure line, note, clef sign, accidental, etc.), the voicing, and the need to keep things from overlapping. The user can also give a parameter that determines how "dense" the justification should be. The justifier then squeezes things together as close as possible based on these constraints. Figure 5h shows the results of this step as applied to the section of score from Figure 5g.

Finally, the justifier stretches out the spacing in the material to make an integral number of measures fit on each line. The user can justify various sections of a piece with different densities as appropriate. Figure 5i shows the consequence of using a low density parameter on this section. Figure 5j shows a more reasonable density. The last step is to tilt the beams and make clef changes as shown in figure 5k.

If at any point the user is dissatisfied with the results of justification, he can manually move items around in the score to improve the appearance. However, the justifier produces a surprisingly good layout, so that usually the only things left for the user to do are adjustments that enhance readability (such as grouping and tilting beams, adding of clef switches, etc.). In fact, given the power of the justifier, one common style of use is to enter music one voice at a time, not worrying at all about the spacing between notes. Each time a line of material has been entered, the user justifies that and then goes on to the next line. The justifier takes care of aligning the voices properly and producing a suitable layout.

The justifier is also helpful in determining page layout and page breaks. The user can indicate that specific measures are to fall at the end of a line. The justifier takes this into account when deciding how many measures to put on a line. With this feature, the user can control how many pages the score will fill and can assure that the end of the score falls at the end of a page.

Data Structures

We felt that it was important to include a section on data structures because it took us such a long time to arrive at a final design. We will describe this design and explain why we think that it is the most suitable one for our purposes.

Our first design was a structured, hierarchical data structure that closely matched the formal structure we saw in music. But we ran into numerous problems with it because it didn't match the needs of an editor. After much discussion, we settled on an unstructured, sequential data structure. This surprised us, because in the beginning we had thought that the hierarchical design was the obvious choice. However, experience has convinced us that the sequential design is vastly superior.

There are three considerations in designing a data structure: representational power, programming convenience, and performance. Representational power concerns how much of the domain is covered by the data structure and how easy it is to represent different aspects of the domain in the structure. Programming convenience concerns how easy it is to write algorithms that deal with the data structure. This depends a great deal on what the algorithms do. In Mockingbird we are mostly concerned with playing, editing, and displaying the score (as opposed to structural analysis or automatic composition). Performance concerns how rapidly the data may be accessed. Even if a data structure is convenient, it may not be efficient. Sometimes there is a trade-off between structural complexity, memory utilization, and speed. In Mockingbird, memory was plentiful and speed was critical; for an editor to be useful, the display must respond crisply to user actions.

There are many possible hierarchical data structures that might be used to represent music. We use the term loosely to describe a class of data structures that implicitly incorporate musical structure in the design. Thus one might imagine a data structure that had a separate part for each measure or for each voice. A "sequential" data structure, on the other hand, is simply a sequence of undifferentiated entities. No attempt is made to incorporate musical structure into the design. Instead, it is up to the algorithms to determine the structure from the entities.

On the surface, the hierarchical design seems obviously better. If the musical structure is built into the data structure, then one can guarantee a uniform interpretation over all of the algorithms. Not only that, but algorithms won't have to derive the built-in information.

Unfortunately, basing the data structure on the musical structure was too constraining. We wanted a uniform representation for both common music notation and piano rolls so that the user could mix both types of music freely. Although it might be possible to keep a separate data structure for piano rolls, it would make the algorithms for editing, displaying, playing, and justifying much more difficult. All of these algorithms need to know what things are near one another. A simple example is redisplaying the score after a small edit has been made. For efficiency, we would like to redisplay as little of the score as possible. But that requires knowing what objects are near the entity that was edited. In the hierarchical design, an entity that is close physically may be logically far away. It might be in another measure, voice, or chord, or on a different staff. Enumerating all of the possibilities is incovenient and time-consuming.

In addition to all this, there is the problem of exceptions. Many of the rules of notation that are presumably inviolable turn out to be violated when the composer finds the notation too constraining. Figure 7 shows several examples of this. A design that has fixed rules about the structure of music built into it won't be able to handle such exceptions. Even if the exceptions were ruled out, one would still have problems with the inconsistent structures that arise temporarily during editing. We wanted our design to be tolerant of such exceptions and inconsistencies.

A sequential design doesn't have these problems. It allows piano rolls to be mixed with standard music notation because both are represented as ordered sequences. Finding things that are nearby is easy, because things that are near one another on the screen are near one another in the data structure. And finally, since the data structure is so unstructured, it is flexible enough to handle a wide range of exceptions.

Mockingbird's "sequential" data structure is simply a sequence of events ordered by time. An event might be a measure line, a collection of notes, a time signature, a clef change, or a change in the number of staves per line. The events that contain notes are called "syncs" because they synchronize all of the notes in the event. (That is, all of the notes in the sync are played or displayed together.) The notes may belong to different voices or chords, but they all have the same "time." The editor automatically synchronizes notes that are very close to simultaneous whenever notes are entered or moved. Occasionally this will introduce an error, which can be fixed by the user.

Syncs are important because they keep simultaneous notes together while the score is being edited. Usually, if the composer plays several notes at the same time, he wants them to stay together unless he explicitly says otherwise. Inserting a note before a sync shouldn't break up the sync, even if one of the notes in the sync belongs to the same voice as the inserted note. If any of the notes in a sync move, they should all move. (The justifier sometimes violates this rule, but only when it is obvious that the notes have been incorrectly synchronized.)

There are three ways of measuring the "time" of an event: as seconds from the start of play, as beats from the start of the score, and as inches from the first measure line. Although these notions of time are very different, they coexist nicely because the order of one is usually the order of the other. Thus, if note A is displayed to the left of note B, it is most likely played before note B. In general, we can use the order of the notes as they appear on the display to determine the order in which they should be played. There are a few exceptions which must be handled properly—embellishments such as trills and grace notes are not always played in the order that they are displayed. Conversely, notes that are logically simultaneous may be separated slightly on the display in the interest of visual clarity.

Beams and chords are ancillary to the main data structure, since they act as horizontal and vertical parentheses grouping notes together; they are visual aids for the human performer and aren't otherwise crucial to the score. Removing all of the beams and chords from a score would affect its readability but not its playing. In Mockingbird, each beam and chord knows what notes belong to it, and each note knows what beam or chord it belongs to. In addition, chords have a stem direction and beams have a tilt and vertical position.

The linear data structure we have been discussing treats a score as a single, long sequence of measures, but since music is printed on rectangular sheets of paper, this sequence must be broken up into a succession of lines. Rather than complicate the main data structure, we use a separate data structure to map between the one-dimensional score and the two-dimensional sheet of paper. This sheet data structure keeps track of how long each line is, how many lines fit on each page, how much of the line must be devoted to a key signature, and which section of the score goes on which line. Only the displayer and the justifier need to make use of this representation of the sheet; all of the other algorithms manipulate the sequential data structure directly.

Automatic Recognition of Piano Scores

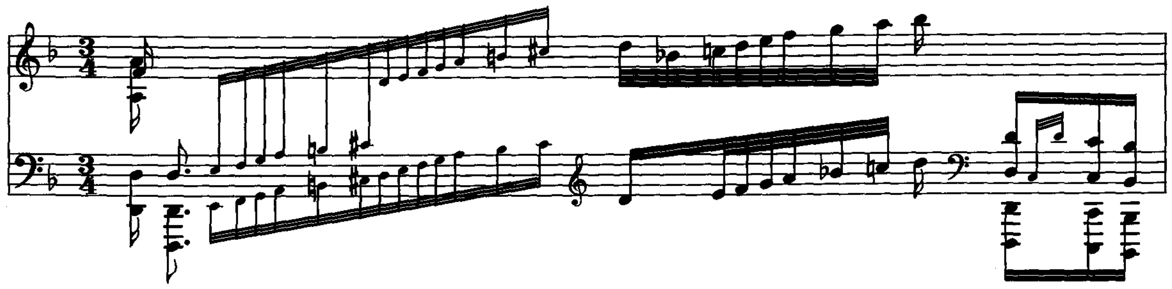
We decided not to try to write a program which would attempt to deduce the score automatically from piano roll input because we thought that job exceedingly difficult. Why? It has been done for some simple pieces; can it not be done more generally? We believe that the relevant question is: can it be done for polyphonic piano music where the voicing is not known in advance?

To produce a proper score involves, among other things, determining the time value of all notes. Although one traditionally thinks of "holding a note," say, for a quarter, one doesn't mean literally holding the key down. What is meant is that the next note (or rest) *in that voice* is to commence one quarter after the beginning of the note in question. Time values thus measure intervals rather than durations. (Although often the two are almost the same, in staccato playing it is not at all the case.) So, before one can assign time values to notes, it is first necessary to understand how the music is separated into its component voices. Voicing is partly dependent on thematic and harmonic relationships within the piece, but it may also be used by a composer simply to indicate how he wishes the music to be parsed by the reader, for emphasis or for division between the hands. Sometimes a single note may participate in two different voices, possibly even with two different time values. Rests and ties further complicate the problem, as they are elements that appear in the score to help complete rhythmic structuring, but they are absent from the actual playing.

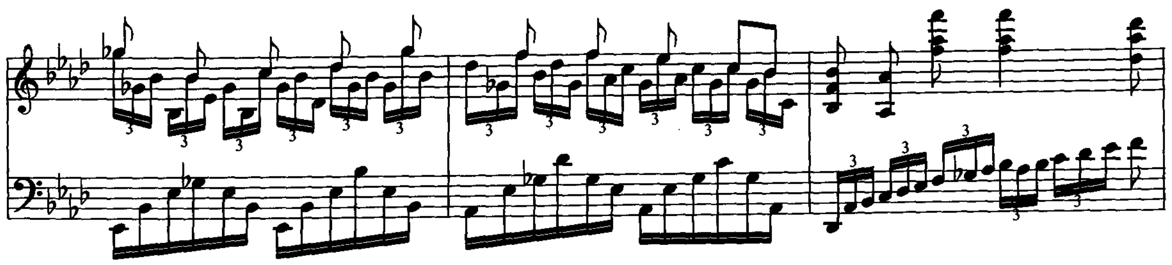
The assignment of notes to staves forms a further structuring element in piano music. This assignment is a complex function of voicing, fingering, division between hands, and aesthetic judgment. There are many other problems: identifying complex n-tuplets, distinguishing grace notes, determining rhythm and detecting rhythmic changes, identifying measure lines, determining how many staves to use, how to combine notes into beams, when to switch clefs or use ottava, and in short, determining *all* of the complex structural notational devices that composers utilize to render their music readable.

By studying more and more scores, we found that complexity often gave way to ambiguity—that decisions about notation were often a matter of personal taste. We invite the reader to study the examples shown in Figure 7 and consider the problems of deducing these scores algorithmically from even the most precisely played, but unvoiced input. After studying many such examples, we finally concluded that deducing, in general, how a piece should be structured was not merely a difficult problem; it was in many cases undefined. In any case, it was certainly beyond the scope of our interests or ambitions.

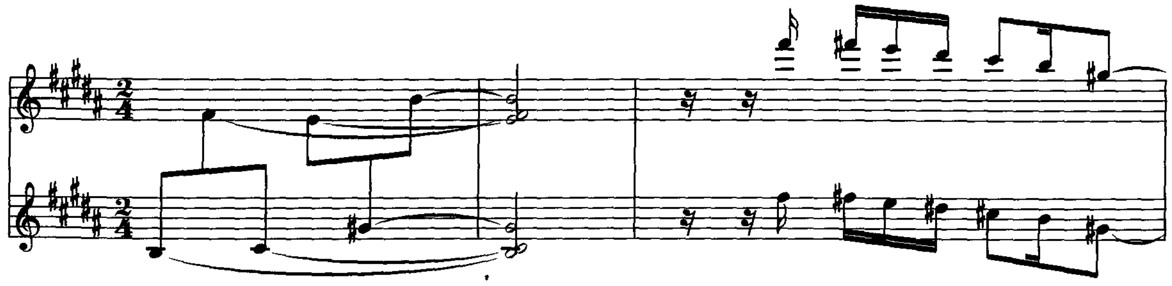
7a)



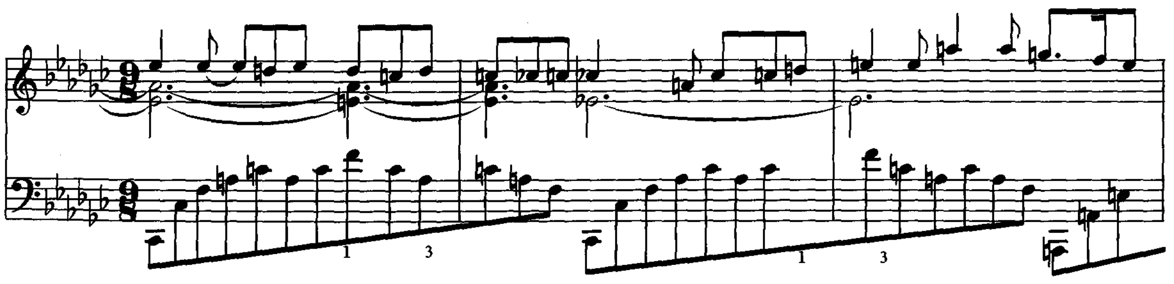
7b)



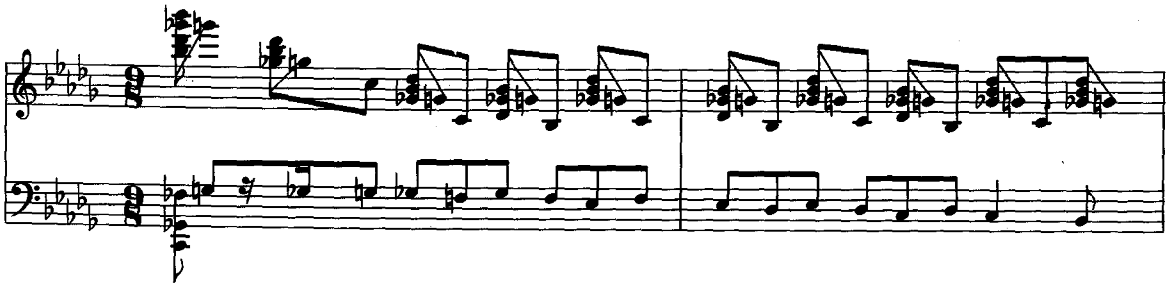
7c)



7d)



7e)



Figures 7a through 7e: Difficult Cases for Automatic Score Production

Conclusion

Our intention in presenting this material is to encourage others to pursue similar endeavors. Music editing is already being done on home computers and while it will be some years before machines as powerful as a Dorado are found in every living room, useful tools will become feasible soon—even on home machines of modest cost. It would seem that a display of reasonable resolution and a mouse (or some similarly convenient pointing device) are prerequisites. But if one avoids the temptation to make "pretty" scores and sticks to providing a simple "cut and paste" editor of piano roll material, then a reasonable composing tool should soon become practical. The problem of storage and retrieval of snatches of material and full pieces would have to be addressed, but this seems quite tractable.

Acknowledgments

Mockingbird was made possible by a fortuitous convergence of people, interests, and facilities. The environment at Xerox PARC in general, and within the Computer Science Laboratory (CSL) in particular, provided a hospitable environment for this work. The existence of and access to a Dorado was absolutely essential. Robert W. Taylor, director of CSL, provided us with this and all other necessary facilities and support. Will Crowther worked with us closely on the initial design and helped us get started in the right direction. John Warnock and Doug Wyatt provided us with the Cedar Graphics software package which Mockingbird uses. Gene McDaniel wrote special Dorado microcode for handling the Synthesizer and Mike Overton built the hardware interface. Last, and most gratifying of all, has been the enthusiastic support that we received from our colleagues, whose vicarious pleasure in seeing Mockingbird come to life cheered us along the way.

References

1. Geschke, C. M., Morris, J. H., and Satterthwaite, E. H. Early experience with Mesa. Report CSL-76-6 Xerox PARC, Palo Alto, CA, October 1976.

2. Mitchell, J.G., Maybury, W., and Sweet, R. E. Mesa language manual, version 5.0. Report CSL-79-3 Xerox PARC, Palo Alto, CA, April 1979.

3. Lampson, B. W., Pier, K. P., Ornstein, S. M., Clark, D. C., and McDaniel, G. The Dorado: a high performance personal computer (three papers). Report CSL-81-1 Xerox PARC, Palo Alto, CA, January 1981.

4. Metcalfe, R. M., Boggs, D. R., Crane, R. C., Taft, E. A., Shoch, J. F., and Hupp, J. A. The Ethernet local network (three papers). Report CSL-80-2 Xerox PARC, Palo Alto, CA, February 1980.

5. Warnock, J. and Wyatt, D. A device independent graphics imaging model for use with raster devices. *Computer Graphics 16*, 3 (July 1982), 313-320. (Siggraph '82)