Proceedings of The 1980 International
Computer Music Conference

John Strawn
Center for Computer Research in Music and Acoustics
Stanford University
Stanford, California 94305

Approximation and Syntactic Analysis of Amplitude and Frequency Functions for Digital Sound Synthesis

1. Introduction

Of the various models proposed and used for analyzing and synthesizing musical sound, additive synthesis is one of the oldest and best understood. In recent years, time-variant Fourier methods implemented as computer programs have made it possible to analyze digitally a large variety of tones from traditional musical instruments. Such research has led to a better understanding of the physical and perceptual nature of musical sound as well as improvements in techniques for digital sound synthesis.

The heterodyne filter (Moorer 1973) and the phase vocoder (Portnoff 1976; Moorer 1978) provide time-varying amplitude and frequency functions for each harmonic of a tone being analyzed. This quickly leads to an almost unmanageable increase in the amount of data used to represent the original tone. The question of reducing the amount of data without sacrificing tone quality is thus of potential interest to hardware and software designers as well as musicians and psychoacousticians.

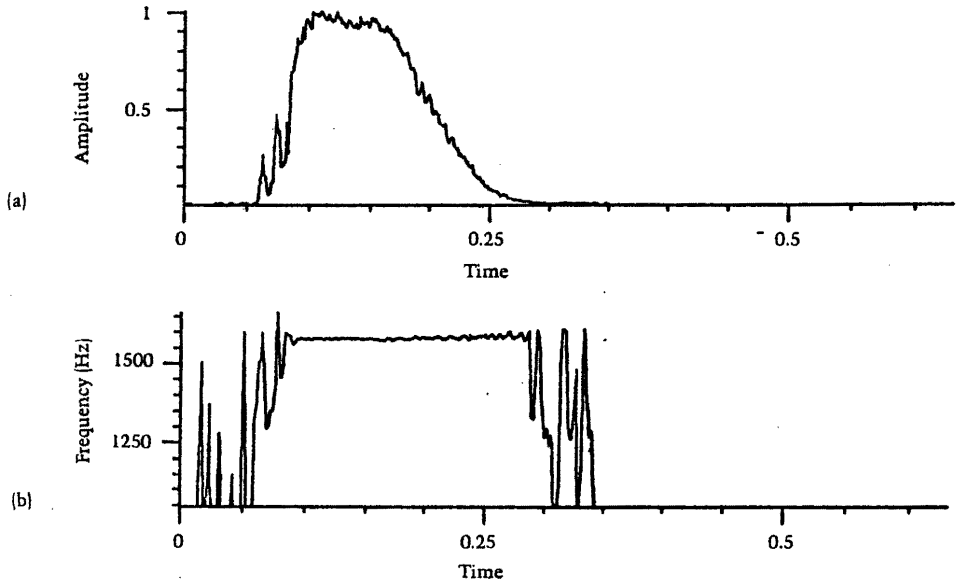
Computer Music Journal, Vol. 4, No. 3, Fall 1980,

One approach to data reduction has involved the use of line-segment approximations (Risset 1969; Beauchamp 1969; Grey 1975), in which an amplitude or frequency envelope is represented by a relatively small number of line segments. Depending on the degree of reduction, tones resynthesized using line-segment approximations often sound as though they were produced by the original instrumental source and in many cases cannot be distinguished perceptually from the original tone.

There is still no definitive answer to the question of how much data can be omitted without changing the tone significantly. The ultimate goal would be to use the smallest possible amount of data to produce a tone that would be perceptually indistinguishable from the (digital) recording of the original tone (Strong and Clark 1967). Grey was unable to explore the question of the degree of acceptable data reduction because at that time only analog tape recordings could be digitized for analysis by computer at the Center for Computer Research in Music and Acoustics (CCRMA). Thus the resynthesized tone could be discriminated from the original tone merely by the absence of tape hiss. Using the digital recording facility (Moorer 1977) it is now possible to record traditional musical instruments at CCRMA in digital form.

An important problem in data reduction concerns the selection of features to be retained in the

Fig. 1. (a) The amplitude-versus time function of the fifth harmonic of a single trumpet tone, analyzed by the heterodyne filter and normalized to a maximum amplitude of 1.0. The y-axis represents amplitude on an arbitrary linear scale. This function has been chosen as an interesting test case for investigating methods of approximation because of the large amount of noise in the trace and the presence of blips in the attack. Presumably such blips play an important role in the timbre of the note from which this function was derived. (b) The frequency-versus-time function for the same harmonic of the same tone as in (a). Both functions contain 230 points.



simplified representation of the amplitude and frequency waveforms. In this article, *feature* will be used in a very narrow sense to mean components of functions that are presumably important perceptually. An example of this would be the so-called blips which typically occur in brass tones, such as those shown in Fig. 1 (cf. Strong and Clark 1967; Moorer, Grey, and Strawn 1978). The central hypothesis of the work discussed here might be formulated as follows: it is possible to reduce time-varying amplitude and frequency functions derived from traditional instrument tones to some minimum number of line segments such that a digitally resynthesized tone using such line segments is perceptually indistinguishable (according to some suitable measure) from a digital recording of the original tone. Reducing the number of line segments further, that is, omitting some features, results in tones which can be distinguished from the original.

Various manual and automatic algorithms for generating line-segment approximations were used in previous research. In the first half of this paper we will review several algorithms from the literature on pattern recognition which have been developed for analyzing such diverse data as coastlines on maps (Davis 1977), electrocardiograms (Pavlidis and Horowitz 1974), chromosomes (Fu 1974), outlines of human heads (Kelly 1971), and gasoline blending curves (Stone 1961), but which have not yet been applied to musical problems. After discussing the difficulties inherent in such "low-level" techniques for the problem at hand, preliminary results from a syntactic, hierarchical scheme for analyzing amplitude and frequency functions will be presented. Since Grey concluded that it was necessary to retain time-varying information for both frequency and amplitude functions (1975), a method for analyzing both will be discussed.

The algorithms which will be outlined below draw extensively from the literature on approxima-

tion theory and pattern recognition. Considerations of time and space prevent a review of these topics here, except the mention of some standard texts (Davis 1963; Rosenfeld 1969; Duda and Hart 1973; Fu 1974; Tou and Gonzales 1974) which have proved useful.

The work presented in this article forms part of a larger research project investigating the role of timbre in the context of a musical phrase. For the purposes of this report, only individual tones from traditional orchestral wind instruments will be presented. The trumpet tone shown in Fig. 1 was analyzed using the heterodyne filter, but the phase vocoder will be used exclusively in future work. No matter which of the two analysis techniques are applied, the issues of approximation remain the same.

2. Line Segments

Formally speaking, the various approximations will be treated as first-degree splines. The following definition has been adapted from that of Cox (1971). Let $f(t)$ be a sampled function, where $t = \{t_0, t_1, t_2, \ldots t_{(n-1)}, t_n\}$ is a sequence of real numbers representing time defined at $t = qT$; T is some sampling period and $q = 0, 1, 2, \ldots n$. Then a first-degree spline *approximation* $a(t)$ to $f(t)$ is given by

$$a(t) = \begin{cases} a_1(t) = g_1 + h_1(t - t_{a0}), & t \in \{t_{a0}, t_{a1}\} \\ a_2(t) = g_2 + h_2(t - t_{a1}), & t \in \{t_{a1}, t_{a2}\} \\ \quad \ldots \\ a_m(t) = g_m + h_m(t - t_{a(m-1)}), & t \in \{t_{a(m-1)}, t_{am}\} \end{cases} \quad (1)$$

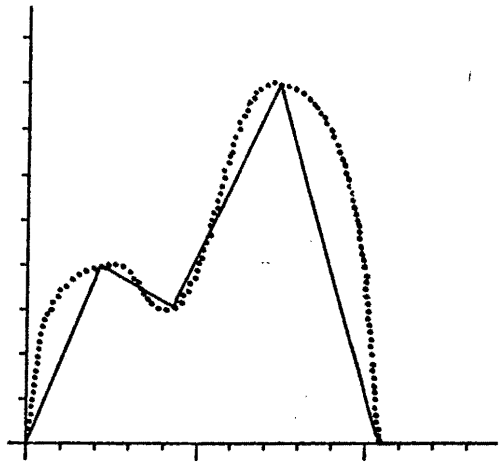
where $m \leq n$, $t_0 = t_{a0}$ and $t_n = t_{am}$. $a(t)$ is also required to be continuous across $t_{a0} \leq t \leq t_{am}$, with continuity defined as

$$a_i(t_{ai}) = a_{i+1}(t_{ai}). \quad (2)$$

Any a_i is obviously continuous across $t \in \{t_{a(i-1)}, t_{ai}\}$.

Furthermore, one important restriction has been adopted and concerns the endpoints of the line segments used to approximate a function. Each endpoint is required to be the same as some data point in the original waveform (Fig. 2). In terms of the definition given above:

Fig. 2. The solid line represents a first-degree spline function a(t) as defined in Eq. (1), with m = 4. The function f(t) being approximated is shown as a dotted line. Each breakpoint of a_i is the same as a point in the original function, as specified in Eq. (3).



For every $i \in \{0 \leq i \leq m\}$ there exists $j \in \{0 \leq j \leq n\}$ such that

$$t_{ai} = t_j \text{ and } a(t_{ai}) = f(t_j). \quad (3)$$

The reasons for this restriction will become clear only at the very end of this paper. In the meantime, it will merely be mentioned again when appropriate.

Since the functions being approximated exist only in sampled form, such requirements as the existence of the nth-order derivative are satisfied by definition (see also Pavlidis and Maika 1974).

There is a considerable body of literature on the use of higher-order approximations, for example cubic splines. Line segments, however, have the advantage of being conceptually simple. The effect of changing the slope or intercept of a straight line is easy to conceptualize, but it is more difficult to correlate changes in higher-order polynomial coefficients with changes in the appearance of the approximation to some waveform. However, just such a one-to-one correspondence is essential in the feature-oriented work presented here. Cubic splines are also more complicated in that a change in any $a(t_i)$ will change all of the coefficients across the entire approximation since the slopes at each t_i

are required to be continuous. This criterion of continuity at a breakpoint is in fact ignored when working with line segments. Finally, the line-segment approach is currently implemented in most general-purpose music compilers as well as in most hardware devices for digital sound synthesis.

3. Error Norms

The most common measure of error is the square of the vertical distance (i.e., the distance parallel to the y-axis) between a function and an approximation to it. Duda and Hart (1973, p. 332) discuss a variation of this, in which the error is the *perpendicular* distance from a point $f(t)$ of the original function to the approximating line; Dudani and Luk (1977) present an algorithm for fitting a line to a set of points using this error norm. But according to Moorer (1980), experience has shown that using this error criterion does not improve the results of approximation for the class of waveforms under discussion here. Since it involves a considerable increase in computation time, this variation has not been tested in the work to be presented below.

Before various methods for approximating functions can be examined, three error norms need to be defined.

3.1 Maximum Error

The maximum error E_∞ is given by

$$E_\infty = \max_t [f(t) - a(t)]^2, \tag{4}$$

with $a(t)$ defined in Eq. (1). This is sometimes called the *uniform error norm* (Davis 1963, p. 133).

3.2 Sum-of-Squared Error

Pavlidis (1973) also calls this norm the *integral square error*:

$$E_n = \sum_{t=t_0}^{t_n} [f(t) - a(t)]^2. \tag{5}$$

3.3 Mean Squared Error

The mean squared error across each a_i is given by

$$E_m = \frac{\sum_{t=t_{a(i-1)}}^{t_{ai}} [f(t) - a(t)]^2}{t_{ai} - t_{a(i-1)}}. \tag{6}$$

The E_m norm is more tolerant of error across long line segments than the E_n and E_∞ norms.

4. Algorithms for Line-Segment Approximation

There are two basic approaches to solving the problem of approximation using splines. In the first, the number of segments is specified in advance and the algorithm is required to minimize some measure of error. The number of splines is changed in the other method until the measure of error lies as close as possible to, but still under, some predetermined threshold.

4.1 Minimizing Error: ADJUST

One method for minimizing error with a given number of line segments is presented by Pavlidis (1973). Since two typographical errors occurred in Eq. (3) of Pavlidis's article, (which should read $P_j = t_{i(j)} - M^k$), and since it forms an integral part of the procedures discussed in the next section, Pavlidis's algorithm will be discussed in some detail.

The algorithm, called *ADJUST* in the rest of this paper, is given in Fig. 3. For each iteration, the endpoints of successive segments (first the odd-numbered segments, then the even-numbered ones) are moved by some number M, always set to 1 in the work discussed in this paper; larger values of M could be specified for the first few iterations if the initial approximation were thought to be significantly different from the expected final solution. If the error of the approximation using the trial breakpoint is less than the original error, then the trial

Fig. 3. The algorithm ADJUST, modified from Pavlidis's algorithm (1973), in a quasi-ALGOL notation. The algorithm accepts as input some function to be approximated, some initial approximation a(t) *in the form given in Eq. (1),* M *(the number of points to move a breakpoint for each trial), and* iterationLimit *(the maximum number of iterations allowed). When the algorithm has finished, either iterationLimit has been exceeded or the approximation has been adjusted so that any further changes in the breakpoints will cause some (local) error to increase.* Error *in this algorithm refers to one of the three error norms defined in Section 3 of the text, although Pavlidis originally designed this algorithm for the* E_x *and* E_n *norms.*

```
INTEGER iterationCount, start, stop, i;
BOOLEAN odd, breakPointChanged;

FOR iterationCount ← 1 STEP 1 UNTIL iterationLimit DO
  BEGIN "iteration loop"
  breakPointChanged ← FALSE;
  FOR odd ← TRUE, FALSE DO
    BEGIN "odd/even loop"
    IF odd THEN start ← 1 ELSE start ← 2;
    FOR i ← start STEP 2 UNTIL n DO
      BEGIN "step through segments"
      τ1 ← t(ai-1);
      τ2 ← tai; COMMENT τ1, τ2 are the breakpoints for segment ai,
      τ3 ← t(ai+1); COMMENT τ2, τ3 are the breakpoints for segment ai+1;
      ei ← error across [τ1, τ2];
      ei+1 ← error across [τ2, τ3];
      IF ei ≠ ei+1 THEN
        BEGIN "try moving point"
        IF ei > ei+1 THEN τ2' ← τ2−M ELSE τ2' ← τ2+M;
        ei' ← error across [τ1, τ2'];
        ei+1' ← error across [τ2', τ3];
        IF MAXIMUM (ei, ei+1) > MAXIMUM (ei', ei+1') THEN
          BEGIN "accept moved point"
          tai ← τ2';
          breakPointChanged←TRUE;
          END "accept moved point";
        END "try moving point";
      END "step through segments";
    END "odd/even loop";
    IF breakPointChanged = FALSE THEN DONE;
END "iteration loop";
```

breakpoint replaces the original breakpoint. Pavlidis presents a proof to show that the algorithm will converge in a finite number of steps, and (more importantly) that no cycling is possible. It must be emphasized that this algorithm is useful for finding local minima, not some globally optimal solution. Pavlidis states that the maximum error norm is to be preferred for most applications, although the sum-of-squared error and mean squared error norms have also been used successfully in the work presented here.

A note on the implementation of ADJUST: if both endpoints of some a_i are points from the function being approximated, then e_i, for example, only needs to be calculated using points τ_1+1 through τ_2-1, because the points τ_1 and τ_2 cannot contribute to the error. This represents a slight improvement over Pavlidis's algorithm, which points out this computational saving only for the beginning endpoint.

The results of applying ADJUST to two test cases are given in Fig. 4 and Table 1. The "worst" case (in terms of computational time) is given if the breakpoints for the original approximation are all gathered at one end of the function, in which case ADJUST must spread the points out across the function in the process of finding the optimal approximation.

Fig. 4. Results of applying the algorithm ADJUST (given in Fig. 3) to the approximation of two test cases; further data is given in Table 1. The E_n norm was used in all of the cases illustrated here.

(a) This test case consists of two diagonal lines, each corresponding to 24 units on the x-axis, separated by a horizontal line one unit in length. The original smooth diagonal lines have been modified by adding quasi-random variations, the amplitude of which depends on the y-value of the original line.
(b) Arbitrary, initial placement of four line segments approximating test case (a).
(c) Approximation to test case (a), with four line segments, after algorithm ADJUST has reached a solution.
(d) As in (c), but with 11 line segments. Perhaps if the breakpoints were distributed more evenly, the error could be reduced further and the blip on the left-hand side might be avoided. But ADJUST converges onto a locally *optimal solution, which is not necessarily optimal* globally.
(e) One-half of a sine wave, again spread across 49 units of the x-axis and with noise added as in (a).
(f) Approximation to (e), with four line segments, after ADJUST has reached a solution.
(g) As in (f), with 11 line segments.

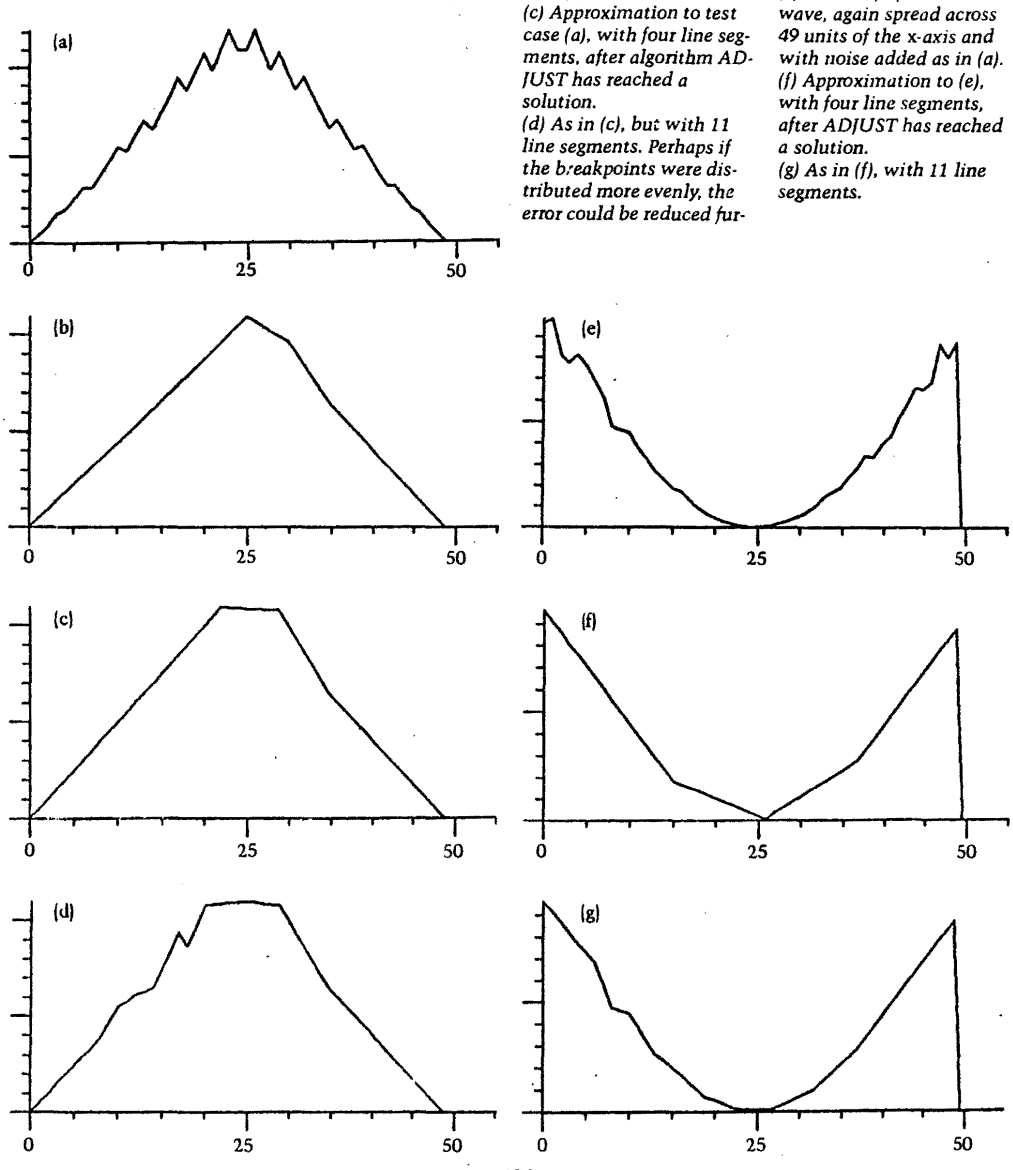


Table 1. Performance of the algorithm ADJUST for the test cases shown in Fig. 4

Corresponding Illustration in Fig. 4	Error (E_n) Initial	Final	Number of Iterations
b, c	122.9	43.21	4
d	43.2	25.5	2
f	0.45	0.07	11
g	0.0205	0.0190	6

NOTE: *Initial error* refers to the error for the initial, arbitrary placement of the approximating line segments. E_n is calculated across the entire test case. Note that the function in Figs. 4(a)–4(d) varies from 0 to 24, whereas the half-period of the sinusoid in Figs. 4(f) and 4(g) does not exceed 1.0.

4.2 Initial Segmentation

Three algorithms (and a variant of one of them) for approximating a function with line segments will be discussed. The first two represent solutions to the problem, discussed above, of find ng an approximation such that the error does not exceed some threshold, with no a priori restrictions on the final number of segments.

One widely used method, in which each line segment provides the least mean squared error approximation to all or part of a function, is not considered here because of the restriction of Eq. (3). Another interesting approach (Stone 1961; Phillips 1968; Pavlidis 1971) will be omitted here for the same reason.

4.2.1 Thresholding Sum-of-Squared Error

The sum-of-squared error norm was defined in Eq. (5). The method, as suggested by Pavlidis (1973), is quite simple. The endpoint for the approximating segment is incremented until the sum-of-squared error exceeds some threshold. The endpoint is then decremented by 1, a segment is established, and the process is repeated using the endpoint just found as the new initial point.

4.2.2 Split-and-Merge

Pavlidis and Horowitz (1974) developed a highly efficient algorithm, presented in Fig. 5. In the version to be discussed here, the error across each segment of the approximating function must not exceed some threshold. The initial line segments are first split into smaller line segments until the threshold requirement is satisfied (or the line segment consists of only two points). Neighboring line segments are then joined when possible by MERGE, that is when joining them will not violate the same threshold conditions. Finally, ADJUST is applied to the segmentation. The algorithm repeats until no changes are made to the breakpoints during an iteration.

In its original formulation, the split-and-merge algorithm returned to the SPLIT procedure ("start:" in Fig. 5[a]) after every iteration. But given the restriction of Eq. (3), SPLIT only needs to be invoked once (during the first pass) for the following reason. After SPLIT, the error across each segment is less than or equal to the threshold. MERGE likewise can only result in segments with error less than or equal to the threshold. For τ_2' in ADJUST (cf. Fig. 3) to be accepted as a breakpoint, e_i' and e_{i+1}' must be less than or equal to e_i and e_{i+1} respectively. But when ADJUST is invoked after MERGE, e_i and e_{i+1} must be less than or equal to the threshold. Thus MERGE and ADJUST cannot result in segments with an error greater than the threshold, so that there is no need to invoke SPLIT again.

4.2.3 "Case 2"

There is another version of the algorithm, called "Case 2" by Pavlidis and Horowitz (1974), in which the error across *all* of the segments must not exceed some threshold. For the maximum error norm, both cases are identical. When using the sum-of-squared error norm, however, the sum of the squared error across the *entire* function is examined. Modifications to the split-and-merge algorithm are necessary. This variation has been tested but will not be discussed here for reasons to be summarized in Section 4.3.

4.2.4 Curvature

There is yet another method, based on a measure of curvature (Symons 1968; Shirai 1973; 1978). In a study of human visual perception, Attneave (1954)

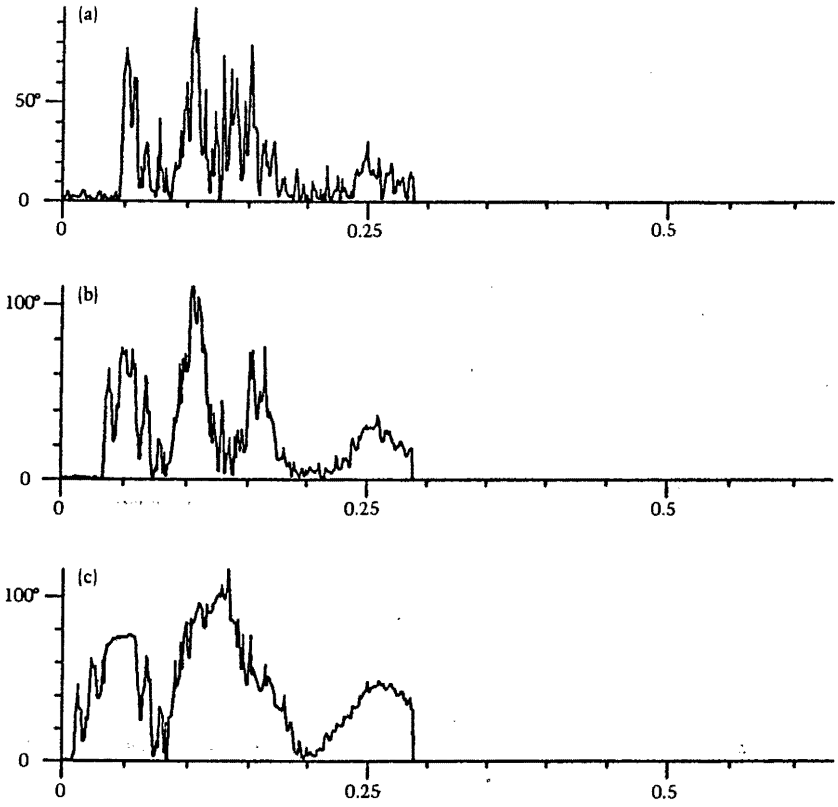
Fig. 5. The split-and-merge algorithm in a quasi-ALGOL notation, after Pavlidis and Horowitz (1974). Procedure ADJUST is presented in Fig. 3. (a) Outline of the algorithm. (b) Procedure SPLIT. Assume that SPLIT has been initialized so that $t_{a1} = t_{an}$, that is, there is initially one line segment which covers the entire function f(t). (c) Procedure MERGE.

```
a) Split-and-Merge Algorithm
     start:  breakPointChanged ← FALSE;
             invoke SPLIT
     loop1:  breakPointChanged ← FALSE;
             invoke MERGE
             invoke one iteration of ADJUST
             IF (breakPoint changed by ADJUST or MERGE) THEN GOTO loop1, ELSE DONE;

b) Procedure "SPLIT"
   i ← 1
   numberSegments ← 1;
   loop2:
   τ1 ← t(a(i-1));
   τ3 ← t(ai); COMMENT τ1, τ3 are the breakpoints for segment ai;
   ei ← error across (τ1, τ3);
   IF ei > threshold THEN
     BEGIN "split ai"
     IF maximum squared error across (τ1, τ3) occurs only once
       THEN τ2 ← point halfway between (τ1, τ3)
       ESLE τ2 ← point halfway between (first) two error maxima across (τ1, τ3);
     redefine segment ai to extend from τ1 to τ2
     after ai, insert a new segment to extend from τ2 to τ3
     renumber segments
     numberSegments ← numberSegments+1;
     breakPointChanged ← TRUE;
     END "split ai"
   ELSE
     BEGIN
     COMMENT note that ai may be split more than once if necessary;
     i ← i + 1
     IF i > numberSegments THEN DONE "SPLIT";
     END;
   GOTO loop2;

c) Procedure "MERGE"
     i ← 1;
   loop3:
   IF numberSegments = 1 THEN DONE "MERGE";
   τ1 ← t(a(i-1));
   τ3 ← t(a(i+1)); COMMENT segments ai and a(i+1) lie between τ1 and τ3;
   ei ← error across (τ1, τ3);
   IF ei ≤ threshold THEN
     BEGIN
     redefine segment ai to extend from τ1 to τ3 (i.e. remove tai as a breakpoint)
     renumber segments
     breakPointChanged ← TRUE;
     numberSegments ← numberSegments − 1;
     END
   ELSE
     BEGIN
     i ← i + 1;
     IF i ≥ numberSegments THEN DONE "MERGE";
     END;
   GOTO loop 3;
```

Fig. 6. (a) Curvature (in degrees) of the waveform shown in Fig. 1, with M *(see text) set to correspond to approximately 12.5 msec. (b) As in (a), but with* M ≈ *25 msec. (c) As in (a), with* M ≈ *50 msec. In (a), even small variations in the original function result in large values of curvature.* M *in (c) is so large that the points of high curvature at* t ≈ *0.1 sec and* t ≈ *0.15 sec are missed.*

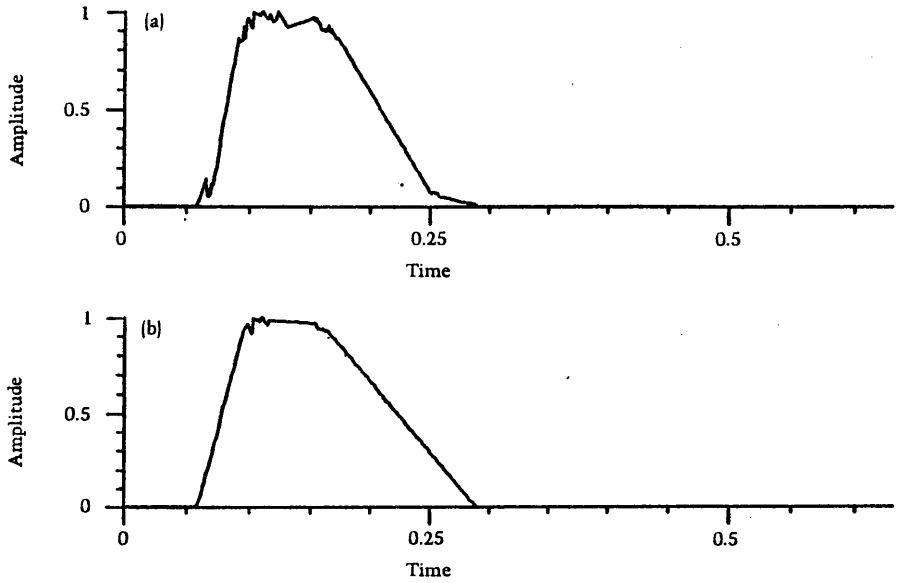


found that the points on a curved line where its direction changes most rapidly were extracted by test subjects as endpoints for drawing an outline of the curve. Conversely, he found that an illustration can be satisfactorily reproduced by drawing straight lines between points of high curvature; the cat-drawing in his article has been widely reprinted (e.g., Duda and Hart 1973, p. 339).

Shirai defines curvature at a point P along a digitized function as the angle α between RP and PQ, where Q and R are points of the function a constant number of samples (called M by Shirai) away from P. In this method, then, the curvature at each point of the function to be approximated is calculated (Fig. 6), and breakpoints are assigned at points of high curvature. If RPQ is a straight line, $\alpha = 0°$; for a highly acute angle RPQ α approaches 180°.

It is important to choose an appropriate value for M. Too small a value will result in distortions in the approximation because features which are too small will cause large variations in curvature and thus be assigned breakpoints. If M is too large, sig-

Fig. 7. (a) Approximation to the waveform of Fig. 1, derived from the curvature shown in Fig. 6(b), with the threshold for establishing a breakpoint set at 60°. The breakpoint which presumably should occur near t = 0.25 sec is missed. (b) As in (a), with a threshold of 30°. The breakpoint omitted in (a) is included, along with a large amount of presumably extraneous points. The approximation in (a) comprises 32 segments; (b) has 77, both with a long segment for the initial silence. Choosing an appropriate threshold would be just as hard, if not harder, for the plots of curvature in Figs. 6(a) and 6(c).



nificant points of high curvature will be missed (cf. Fig. 7).

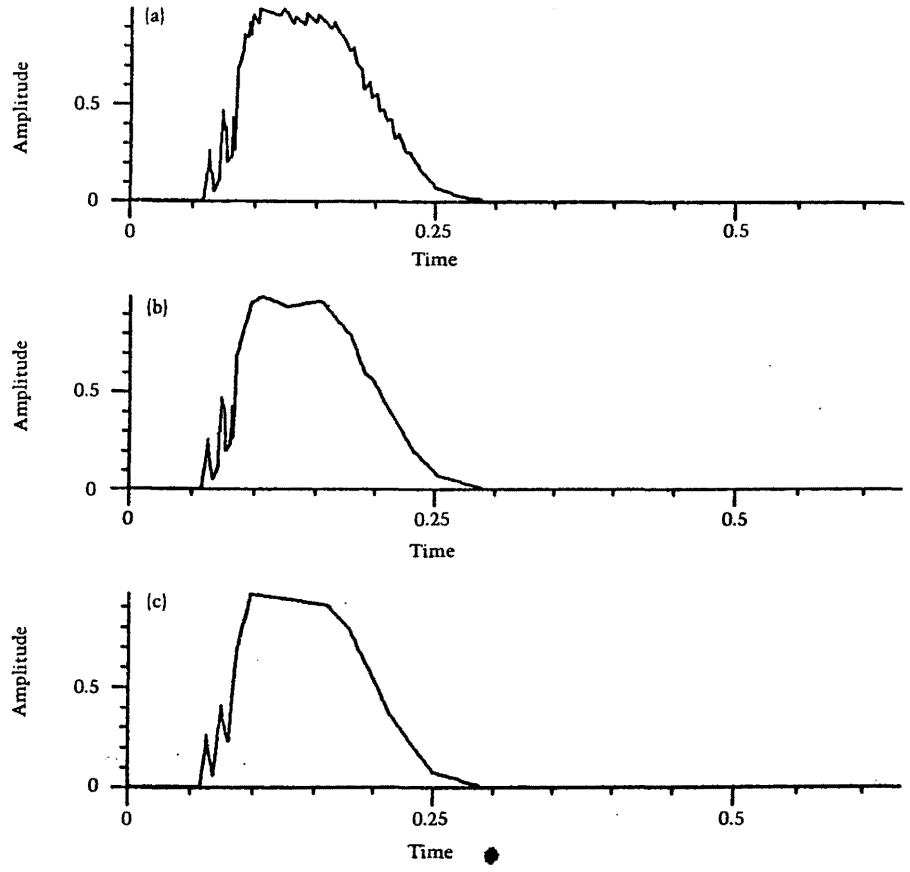
Shirai suggests a thresholding scheme for assigning breakpoints according to curvature. This method undoubtedly works for the cases cited by Shirai, in which fairly smooth lines change direction only occasionally. But as is shown in Fig. 7, it seems difficult to find a single threshold which provides a useful approximation for the kinds of functions under consideration here. This method has also been found to be sensitive to the absolute values used to represent $f(t)$, especially for small M. Another problem occurs because the curvature near a probable breakpoint often does not reach a peak, but rather stays at a plateau; this happens, for example, at $t \approx 0.1$ sec in Fig. 6(b). Obviously only one point needs to be assigned—but which one? I have spent considerable time and effort in an attempt to derive heuristics for selecting only appropriate peaks of curvature, with no notable success to date. Perhaps a varying curvature threshold could be applied, but this has not yet been explored.

4.3 Summary

Three different analyses using the split-and-merge algorithm are presented in Fig. 8. As one would expect, the approximation using a very small threshold captures many details; as the threshold is increased, some details are lost and the overall shape of the waveform is emphasized.

Similar analyses have been conducted using the "Case 2" form of the split-and-merge algorithm as well as the thresholding algorithm of Section 4.2.1. The mean squared error and maximum squared error norms have likewise been explored. In general, using any combination of these algorithms and error criteria it is possible to produce results similar to those shown in Fig. 8. Each combination admittedly reacts to changes in the threshold in a unique

Fig. 8. Results of the split-and-merge algorithm in the form given in Fig. 5, using the sum-of-squared error norm given by Eq. (5). Recall that the maximum amplitude has been normalized to 1.0. (a) A threshold of 0.001 yields 61 segments, some of which contain only two points of the original data. (b) Threshold = 0.01, resulting in 22 segments, a few of which are still only two points long. (c) Twelve segments are found when the threshold = 0.05; the shortest segment now covers five points of the original data. When the threshold is raised to 0.1 (not shown), the first blip in the attack is missed completely. See Section 4.3 for further discussion.



way, but it seems impossible to reach any general conclusion about the superiority of any algorithm or error norm for the purposes of the work outlined in the introduction to this article.

Rather, it has proven impossible to find a paradigm for controlling the threshold for the one waveform shown here so that the projected perceptually salient features (e.g., blips in the attack) can be retained or removed without concomitant, significant changes in the rest of the waveform. If the threshold is large, then the analysis delivers the overall waveshape; with a small threshold, supposedly necessary details are retained along with presumably superfluous ones. (Moorer [1980] has suggested that the threshold could be dynamically adjusted according to some measure of randomness, but this approach has not yet been explored.)

In general, our experience has led to the con-

clusion that no single algorithm of this type will ultimately be sufficient for the systematic exploration of timbre and data reduction. None of these algorithms can take into account both global and local considerations, which seems to be a major drawback. Still, these methods will probably be useful in a wide variety of musical applications where such control is not a prime consideration.

5. Syntactic Analysis

Hierarchical syntactic analysis would seem ideal for mediating between global and local considerations. In the rest of this paper, then, such an approach will be presented.

5.1 Introduction

The method to be discussed draws extensively from the literature on pattern recognition through syntactic analysis, which is based on the similarities between the structures of patterns and (formal) languages. There are, of course, many other methods of pattern recognition, such as template matching, but they will not be discussed here.

In syntactic pattern recognition, a relatively small vocabulary of terminals, called *primitives*, is "parsed" according to the rules (productions) of a grammar to form higher-level nonterminals known as "subpatterns" and "features." Characteristic features are further grouped into patterns (or objects). A successful parse is equated with "recognition" of the pattern. Fortunately, many of the problems involved in pattern recognition by computer can be ignored here. A considerable body of literature is devoted, for example, to the question of finding lines in digitized pictures.

5.2 Primitives

One major advantage of syntactic analysis lies in the fact that a relatively small vocabulary of primitives can be used for constructing a wide variety of patterns. The choice of primitives is thus an important issue. It would seem reasonable, for example, to require that the same primitives be used in the analysis of both amplitude and frequency functions even if it turned out that analysis of the two at higher levels followed different rules.

A generalized approach to primitive selection has not yet been found. Fu, however, gives the following two requirements to serve as guidelines:

> (i) The primitives should serve as basic pattern elements to provide a compact but adequate description of the data in terms of the specified structural relations (*e.g.*, the concatenation relation).
> (ii) The primitives should be easily extracted or recognized by existing nonsyntactic methods, since they are considered to be simple and compact patterns and their structural information not important (Fu 1974, p. 47).

Various kinds of primitives have been developed for different pattern-recognition applications. Fu gives many examples, such as Freeman's chain code and half-planes in the pattern space for representing arbitrary polygons. However, in light of the restriction (still to be justified) imposed in Eq. (3), it seems reasonable to use very small line segments connecting points of the original data as primitives for syntactic analysis.

For reasons discussed by Moorer (1976), the implementation of the phase vocoder used at CCRMA performs an averaging of the amplitude and frequency functions. At a sampling rate of, for example, 25.6 kHz, 32 samples of the original output (corresponding to 1.25 msec) might be averaged to form one output point. These averaged output points are perfectly suited to serve as breakpoints for line-segment primitives in syntactic analysis, and will be used as such in this paper.

5.3 Grammar

The choice of primitives having been made, the next step is to decide on a set of subpatterns and features, and to specify a grammar for parsing the primitives accordingly. Davis and Rosenfeld (1978) provide a grammar based on hierarchical relaxation

methods. Briefly, a line-segment primitive is classified as having length x or $2x$, and as being horizontal, sloped upward, or sloped downward. The relaxation algorithm parses the primitives into longer line segments at a first hierarchical level; at the second level, peaks and valleys are formed from the line segments of the first level, and the final level expresses the whole function as a concatenation of valleys and peaks. There are productions in the grammar for combining not only two adjacent primitives, but also two primitives separated by another.

Pavlidis (1971) provides examples of a grammar which can be used to remove line segments of very short duration and to substitute in their place a single straight line. Grammars for finding more complex features such as the so-called "hats" (up-horizontal-down) or "cups" (down-horizontal-up) are given as well.

Having examined a large number of amplitude and frequency functions, it seemed reasonable to the author to specify three hierarchical levels for syntactic analysis. The first level attempts to remove very small features which one would ascribe to noise in the waveform, artifacts of the analysis procedure, and so on. The second reduces the waveform to its overall shape in terms of fairly long line segments, and the third classifies those line segments into parts of a note. The analysis system, which incorporates elements of the two methods just reviewed, will be presented in detail.

The primitives have already been specified as being very short line segments. Associated with each primitive is its duration and its slope, which are used to classify the line as *up*, *down*, or *horizontal*. The only relational operator between primitives is, coincidentally, the concatenation mentioned in the quote from Fu. Pattern recognition can often involve other operators such as *above* and *below* or *to the right*, which complicate the analysis significantly, but these will not be considered here.

5.3.1 *Level 1a: lineSeg*

The grammar for the first two hierarchical levels is presented in Fig. 9. There are two subdivisions of Level 1. In the first, successive *microLineSeg* primitives (as they will be called) are combined into *lineSeg* as long as (1) the next microLineSeg is within the thresholds *DurThresh* and *YThresh* (defined in Fig. 9), or the direction of the first microLineSeg in the lineSeg being formed is the same as the direction of the next microLineSeg; and (2) including the next microLineSeg will not cause the duration of the lineSeg being formed to exceed some threshold $\Sigma YThresh$.

After a new lineSeg has been found, its duration is calculated as the sum of the durations of the constituent microLineSegs. The slope for the new lineSeg a_i is likewise calculated using the values of $f(t_{a(i-1)})$ and $f(t_{ai})$ at its beginning and endpoints. (This is known as a *synthesized attribute* of a_i [Knuth 1968]). A similar process takes place at the other hierarchical levels.

5.3.2 *Level 1b: macroLineSeg*

It proved advisable to insert a second subdivision into this hierarchical level in order to avoid occasional irregularities at Level 2. This is accomplished by the introduction of *macroLineSeg*, which are formed of one or more lineSeg, all with the same direction (up, down, or horizontal).

5.3.3 *Level 2: featureLineSeg*

The second hierarchical level uses a syntax with productions and conditions corresponding exactly to those of Level 1a. At this second level, macroLineSeg are combined into *featureLineSeg*, with thresholds chosen so that only the most striking elements of the function remain.

5.3.4 *Level 3: Note*

At the third level, featureLineSeg are assigned to the various parts of a note: attack, steady-state, and decay, as well as to any silence which might occur before the attack. Software has been written to search out the longest horizontal segment for which the amplitude (in an amplitude function) exceeds some silence threshold and the duration exceeds some minimum time threshold. Anything

Fig. 9. Grammar for the first two levels of hierarchical syntactic analysis. The conditional form is based on a model used by Pavlidis (1971). The complete hierarchy is shown in Fig. 11. At Level 1 the primitives (microLineSeg) are parsed into lineSeg and then the latter are combined when possible into macroLineSeg. The same syntax, but with different thresholds and corresponding changes in nonterminals, is used in Level 2 for combining macroLineSeg into featureLineSeg. A less formal version of the grammar is given in section 5.3 of the text.

Vocabulary:
V_N = lineSeg, macroLineSeg, featureLineSeg
V_T = microLineSeg

Abbreviations:

τ_L, $f(\tau_L)$	beginning point of lineSeg (featureLineSeg in Level 2)
τ_{i-1}, $f(\tau_{i-1})$	beginning point of microLineSeg a_i (macroLineSeg in Level 2)
τ_i, $f(\tau_i)$	endpoint of microLineSeg a_i (macroLineSeg in Level 2)
D_{M1}	direction of first microLineSeg in lineSeg
D_M	direction of microLineSeg
D_{L1}	direction of first lineSeg in macroLineSeg
D_L	direction of lineSeg
D_{MA1}	direction of first macroLineSeg in featureLineSeg
D_{MA}	direction of macroLineSeg

Thresholds:

DurThresh
: duration of microLineSeg (macroLineSeg in Level 2)

ΣYThresh
: threshold for $\text{abs}[f(\tau_i)-f(\tau_L)]$, constrained to be ≥ YThresh

YThresh
: threshold for $\text{abs}[f(\tau_i)-f(\tau_{i-1})]$

Level	Condition	Production
1a	$\tau_i-\tau_{i-1} > \text{durThresh} \vee$ $\text{abs}[f(\tau_i)-f(\tau_{i-1})] > \text{YThresh} \vee$ $\text{abs}[f(\tau_i)-f(\tau_{i-1})] > \Sigma\text{YThresh}$	lineSeg ::= microLineSeg
	$\tau_i-\tau_{i-1} \leq \text{durThresh} \wedge$ $\text{abs}[f(\tau_i)-f(\tau_{i-1})] \leq \text{YThresh} \wedge$ $\text{abs}[f(\tau_i)-f(\tau_L)] \leq \Sigma\text{YThresh}$	lineSeg ::= lineSeg,microLineSeg
	$D_{M1}=D_M \wedge$ $\text{abs}[f(\tau_i)-f(\tau_L)] \leq \Sigma\text{YThresh}$	lineSeg ::= lineSeg,microLineSeg
1b	$D_{L1}=D_L$	macroLineSeg ::= macroLineSeg,lineSeg
	(none)	macroLineSeg ::= lineSeg
2	$\tau_i-\tau_{i-1} > \text{durThresh} \vee$ $\text{abs}[f(\tau_i)-f(\tau_{i-1})] > \text{YThresh} \vee$ $\text{abs}[f(\tau_i)-f(\tau_{i-1})] > \Sigma\text{YThresh}$	featureLineSeg ::= macroLineSeg
	$\tau_i-\tau_{i-1} \leq \text{durThresh} \wedge$ $\text{abs}[f(\tau_i)-f(\tau_{i-1})] \leq \text{YThresh} \wedge$ $\text{abs}[f(\tau_i)-f(\tau_L)] \leq \Sigma\text{YThresh}$	featureLineSeg ::= featureLineSeg,macroLineSeg
	$D_{MA1}=D_{MA} \wedge$ $\text{abs}[f(\tau_i)-f(\tau_L)] \leq \Sigma\text{YThresh}$	featureLineSeg ::= featureLineSeg,macroLineSeg

between the initial silence and the steady-state is classified as *attack*; everything after the steady-state is *decay*. If no steady-state is found, then as many successive up featureLineSeg as possible are grouped together following the initial silence to form the attack, and the remaining featureLineSeg are assigned to the decay. It should be emphasized that the *attack—steady-state—decay* terminology is used merely for convenience in labeling part of the syntactic analysis, with no semantic implications, at least at this stage.

5.4 Analysis of Amplitude Functions

Figure 10 shows an analysis of the waveform of Fig. 1 at every stage of the hierarchical analysis. None of these plots represents the ultimate form of the output from this method of analysis; further processing is envisioned, as will be discussed. However, by adjusting the thresholds properly, data reduction at, say, the lineSeg level (Fig. 10[a]) can be achieved which will probably be useful in a wide variety of cases.

The model for the analysis of an amplitude waveform assumes that the note will consist of the parts discussed above. This attack—steady-state—decay model has been widely used in commercial analog synthesizers. The fact that it appears here is merely coincidental. The class of waveforms for which this software has been optimized is restricted to waveforms derived from a limited set of test tones lasting, typically, one-quarter of a second or longer.

Ultimately these programs will be used to analyze two or three such notes played in succession, either separated by silence or connected in some way (legato, portato, etc). In fact, the software has already been implemented to handle such groups of notes. Briefly, the output of a given channel of the phase vocoder is first compared with an amplitude threshold. Notes are initially defined within the entire function as being separated by periods of silence. The hierarchical analysis is then performed on a note-by-note basis. Each period of silence is assigned to the note immediately following. The entire function is thus represented as a linked list of notes followed by an optional silence. Each note, in turn, includes an optional initial silence, attack, decay, and optional steady-state. Within each note, the line segments at a given hierarchical level are represented as a linked list of records. A line segment at one hierarchical level also has pointers to one or more line segments at the next lower hierarchical level which the line segment at the higher level encompasses.

5.5 Analysis of Frequency Functions

This preliminary division of a function into notes incorporates the notion of *planning* originally formulated by Minsky (1963) but used here as developed by Kelly (1971). Planning is further applied to the analysis of the frequency functions produced by the phase vocoder, which is especially problematical at low amplitudes where the frequency traces varies widely (Fig. 1[b]). There is also the problem of *phase wraparound*, which occasionally produces characteristic spikes in the frequency trace. Before the frequency traces are submitted to syntactic analysis, these spikes are removed from the regions of the frequency trace bounded by the attack—steady-state—decay parts of the notes in the amplitude function. These same portions of the frequency trace are then analyzed syntactically using the grammar already given in Fig. 9. Details of the assignment of featureLineSeg to the parts of a note vary slightly for the frequency functions but the basic approach is the same. Fig. 11 shows the results of syntactic, hierarchical analysis of the frequency trace belonging to the same harmonic as the tone shown in Fig. 10. Obviously the thresholds are different for amplitude and frequency functions. Given the diverging abilities of the ear to discriminate frequency and amplitude, it would not be surprising if the higher-level analysis of frequency functions could eventually be simplified or modified somewhat.

An entire musical phrase is thus represented as a linked list of phase vocoder channel outputs. Each channel consists of an amplitude and frequency function. Each function in turn points to a linked

Fig. 10. Syntactic hierarchical analysis of the amplitude waveform shown in Fig. 1. (a) lineSeg, resulting from analysis of microLineSeg, with DurThresh = 0.01 sec, YThresh = 0.01, and ΣYThresh = 0.1. (b) lineSeg of the same direction have been combined into macroLineSeg. (c) featureLineSeg, with DurThresh = 0.1 sec, YThresh = 0.2, and ΣYThresh = 0.4. (d) Analysis of featureLineSeg into parts of a note. The "direction" of the straight line from t ≈ 0.1 sec to t ≈ 0.175 sec in (c) is analyzed as being diagonal (pointing downward) and therefore is not classified as belonging to a steady-state. There are 65 line segments in the approximation of (a), 19 line segments in (b), and 7 in (c).

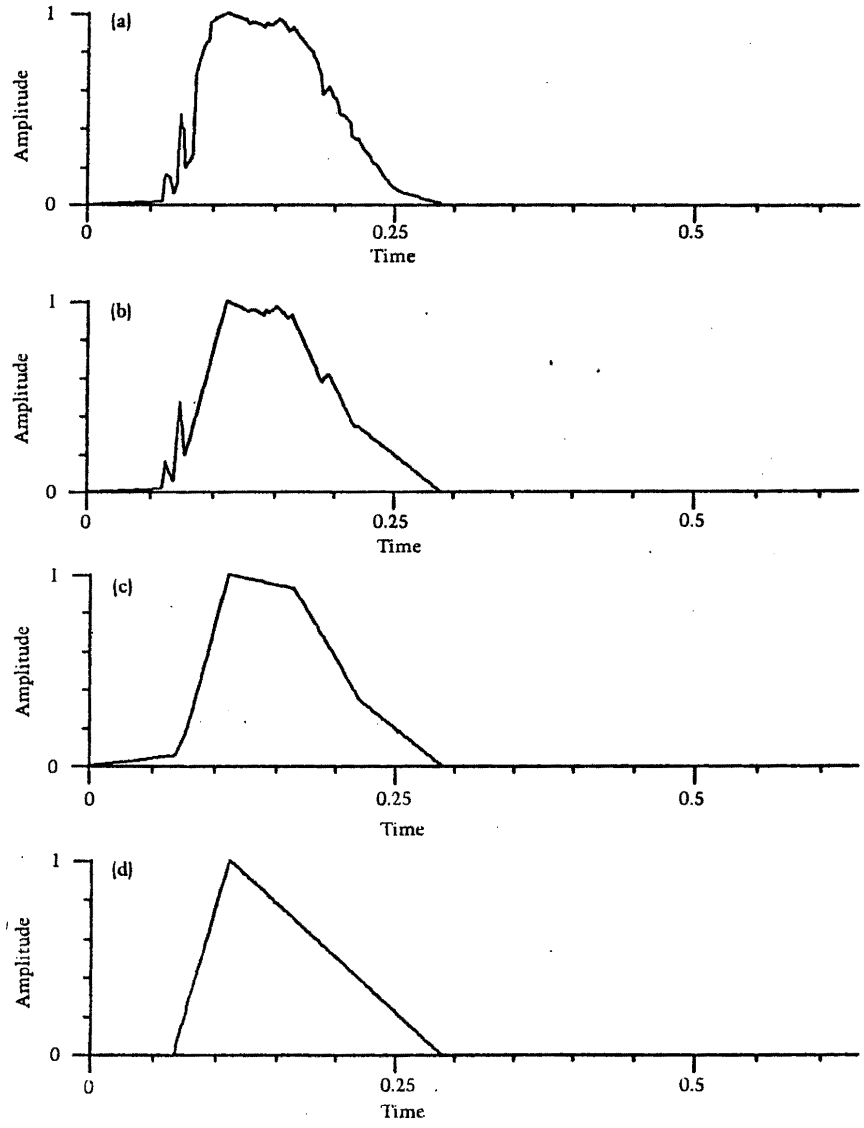


Fig. 11. Syntactic hierarchical analysis of the frequency function of Fig. 1(b). The vertical line extending to the x-axis on the right-hand side is an artifact of the display routine used for plotting; compare this with Fig. 1(b). (a) lineSeg, resulting from analysis with DurThresh = 0.01 sec, YThresh = 2, and ΣYThresh = 5 (YThresh and ΣYThresh in Hertz). (b) lineSeg of the same direction have been combined into macroLineSeg. (c) featureLineSeg, with DurThresh = 0.1 sec, YThresh = 10, and ΣYThresh = 30. (d) Analysis of featureLineSeg into parts of a note. The "decay" slopes slightly downward because it is constrained to end at a point which occurs at the same time as the final point of the note found in the amplitude function. As stated in the text, however, use of the terms attack, steady-state, *and* decay *is merely for the sake of convenience. The top-down parse allows such preliminary analysis to be refined considerably before arrival at a final set of breakpoints for such a function. There are 80 line segments in the approximation of (a), 46 line segments in (b), and 14 in (c).*

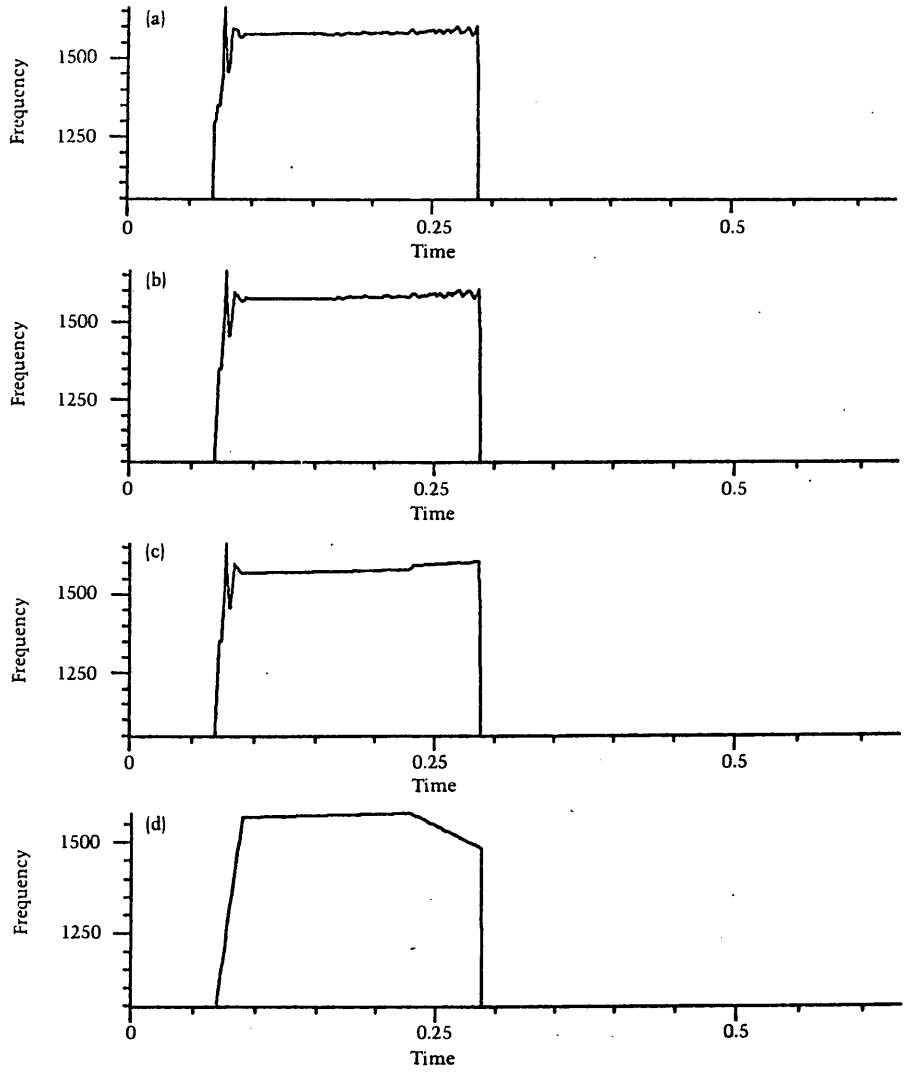
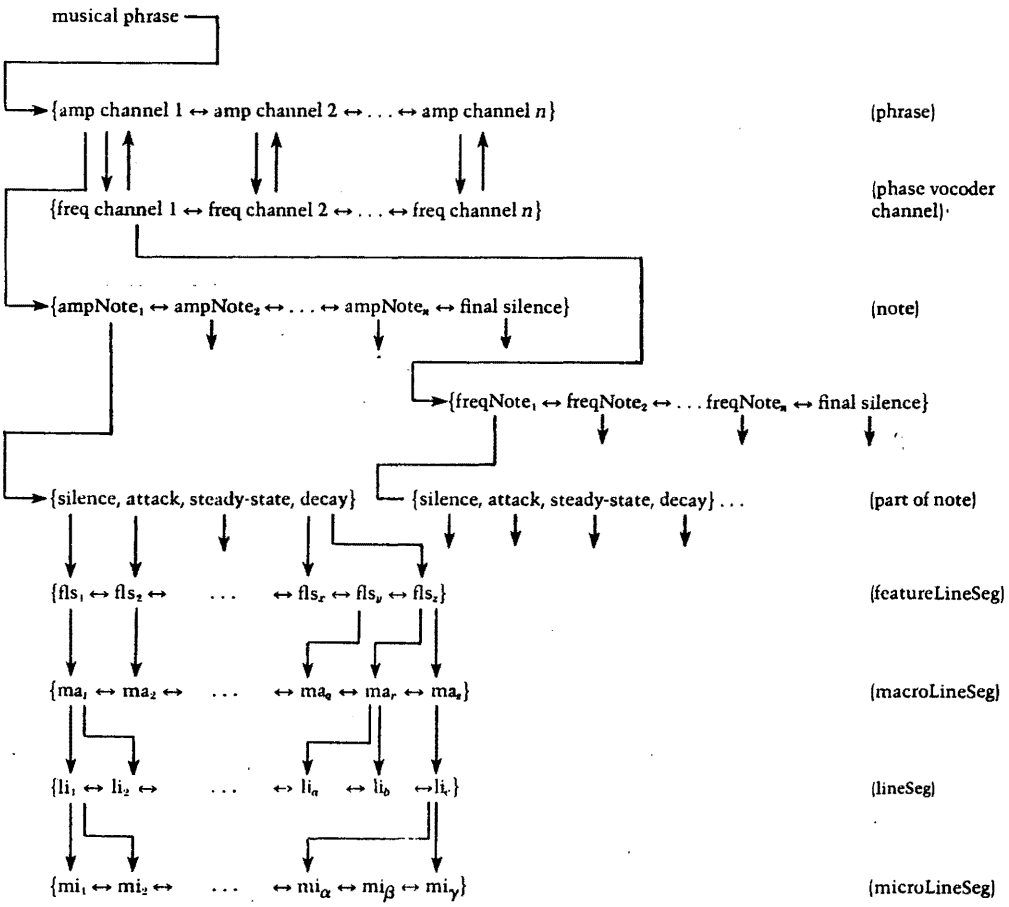


Fig. 12. Overview of the data structure used in the hierarchical syntactic analysis. Curly brackets are used to delineate a two-way linked list of records; the arrangement of pointers between hierarchical levels indicates one of many possible configurations. Names of the hierarchical levels are enclosed in parentheses.



list of notes which are individually analyzed in terms of the grammar presented here. The complete structure is summarized in Fig. 12.

5.6 Refinement: A Top-Down Parse

This "bottom-up" parse is only the beginning of a projected system, shown in Fig. 13. Once the attack, steady-state, and decay portions of the amplitude function have been found, they can be utilized in terms of planning as a guide for a "top-down" directed search for features to be included in the final set of breakpoints. The output would thus no longer be grouped according to attack, steady-state, or decay. Rather, it would consist of the breakpoints found and confirmed by the top-down analysis.

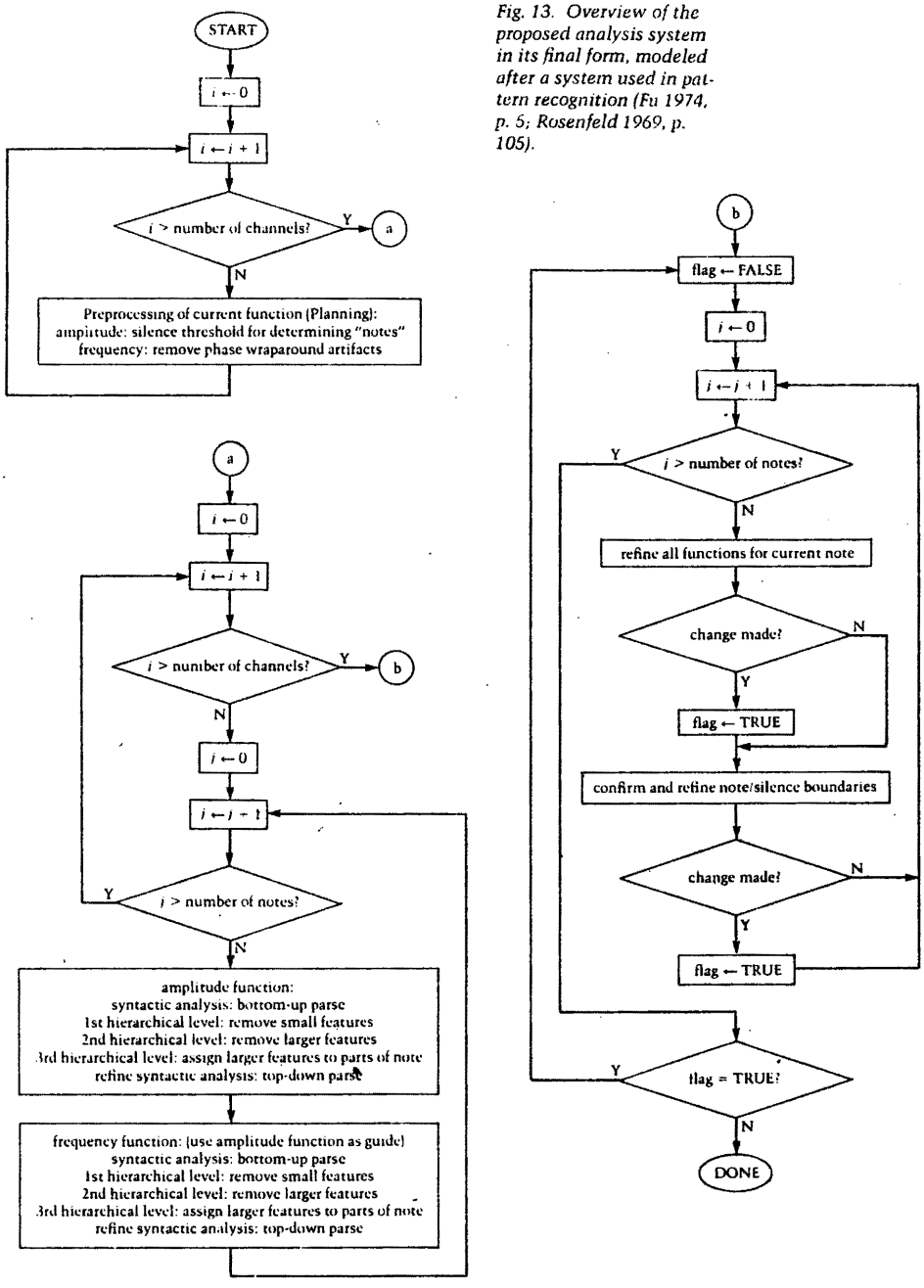
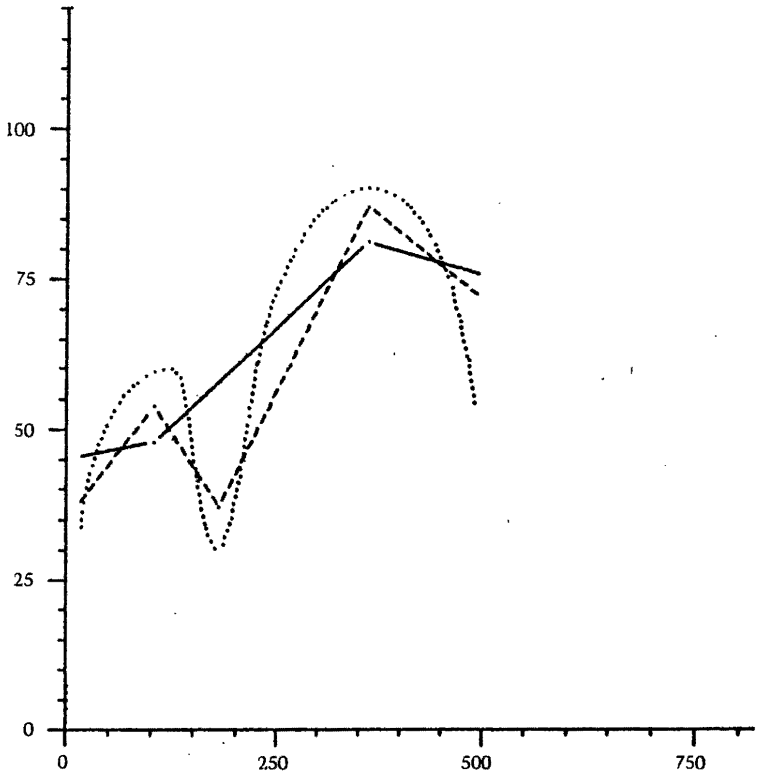


Fig. 13. Overview of the proposed analysis system in its final form, modeled after a system used in pattern recognition (Fu 1974, p. 5; Rosenfeld 1969, p. 105).

Fig. 14. In this figure, the endpoints of an approximating line segment are not constrained to be points of the original data. The solid lines represent some line-segment approximation to a function (dotted line). The parser has decided that an additional breakpoint is to be inserted near the bottom of the "valley," as indicated by the dashed lines. But in order to do so, the original endpoints for the solid line must be recalculated. This additional computational overhead is reduced in the current implementation by the requirement of Eq. (3).



Consider the blips in the attack portion of the amplitude waveform analyzed in Fig. 10. A blip can be represented as a characteristic succession of line segments (e.g., up-down-up, or up-horizontal-down-up) at the lineSeg level. Such a pattern can easily be found using syntactic analysis. The juncture of "attack" and "steady-state" can be refined similarly, for example, by "tracking" to extend each part as far as possible (Shirai 1973) or, to include an initial "overshoot" at the beginning of the steady-state.

After each function is refined in this manner, the entire family of functions can be scanned to confirm the acceptability of various features for each note (cf. Fig. 13). For example, a flag can be set to retain a blip in an amplitude function only if the blip occurs at the same place in, say, more than two harmonics. The initial segmentation of the phase vocoder channel outputs into notes can likewise be confirmed by comparing all of the functions. It might happen that a spurious "silence" detected in some amplitude function would result in an erroneous initial division of one note into two. Such an error can be corrected at this stage.

This also provides, at long last, the justification

for requiring that all breakpoints in the approximation be identical to points of the original waveform (Eq. [3]). If the approximations at the various hierarchical levels could include other points, then a large amount of recomputation would be necessary every time a breakpoint were modified (Fig. 14). In other words, it seems reasonable at this stage to require that the analysis retain some of the low-level information, in the form of the original data points, at higher levels of the parse. This requirement may have to be modified later, however, if it turns out that a significant reduction in the final number of segments can be achieved by doing so.

Other methods from pattern recognition will probably prove useful in this work. Perhaps stochastic syntax analysis (Fu 1974) in the bottom-up parse would reduce the amount of refining to be done later. The Hough transform (Duda and Hart 1973; Iannino and Shapiro 1978) might also prove useful for deciding, for example, that four points defining two separated line segments a_i and a_{i+k}, $k > 1$, were actually colinear and that the entire function between them could be represented as a single line.

6. Summary and Conclusion

Various algorithms from the literature on approximation theory and pattern recognition have been shown to be useful for approximating waveforms in digital sound synthesis. A syntactic analysis scheme has also been presented which will ultimately be extended to a generalized parser for amplitude and frequency functions.

This work represents one aspect of the growing application of artificial intelligence (AI) techniques to musical problems. The syntax and data structure here could easily be extended to a system for instrument recognition similar to the speech-recognition system Hearsay (Erman and Lesser 1975; Erman 1976). A scheme for automatic transcription of music could also be derived which in turn could be used to drive a music-analysis system such as that developed by Tenney (1978). The syntactic analysis presented here will also be useful for approximating other time-varying waveforms in a wide variety of musical applications.

7. Acknowledgments

Dexter Morrill (Colgate University) provided the trumpet tone used as an example in this paper. In addition to assisting at every stage of this work, James A. Moorer (CCRMA) contributed considerably to the design and implementation of the syntactic analysis. I would also like to express my appreciation to my teachers and colleagues for their time, patience, and many helpful suggestions: John Chowning, John Grey, and Julius Smith (CCRMA); Barry Soroka (Stanford AI Project); and Curtis Roads.

References

Attneave, F. 1954. "Some Informational Aspects of Visual Perception." *Psychological Review 61:183–193.*

Beauchamp, James W. 1969. "A Computer System for Time-Variant Harmonic Analysis and Synthesis of Musical Tones." In *Music By Computer*, ed. Heinz von Foerster and James Beauchamp. New York: John Wiley and Son.

Cox, M. G. 1971. "An Algorithm for Approximating Convex Functions by Means of First-Degree Splines." *Computer Journal* 14:272–275.

Davis, Philip J. 1963. *Interpolation and Approximation.* New York: Blaisdell.

Davis, Larry S. 1977. "Shape Matching Using Relaxation Techniques." *Proc. 1977 IEEE Conf. on Pattern Recognition and Image Processing.* Rensselaer, New York, pp. 191–197.

Davis, Larry S., and Rosenfeld, Azriel. 1978. "Hierarchical Relaxation for Waveform Parsing." In *Computer Vision Systems*, ed. Allen R. Hanson and Edward M. Riseman. New York: Academic Press, pp. 101–109.

Duda, Richard O., and Hart, Peter E. 1973. *Pattern Classification and Scene Analysis.* New York: John Wiley and Son.

Dudani, Sahibsingh, and Luk, Anthony L. 1977. "Locating Straight-Line Edge Segments on Outdoor Scenes." *Proc. 1977 IEEE Conf. on Pattern Recognition and Image Processing.* Rensselaer, New York, pp. 367–377.

Erman, Lee D. 1976. "Overview of the Hearsay Speech Understanding Research." *SIGART Newsletter* 56:9–16.

Erman, Lee D., and Lesser, Victor R. 1975. "A Multi-Level Organization for Problem Solving using Many, Diverse, Cooperating Sources of Knowledge." In *Advance Pa-*

pers. Fourth International Conference on AI, Tbilisi, USSR, pp. 483–490.

Fu, K. S. 1974. *Syntactic Methods in Pattern Recognition*. New York: Academic Press.

Grey, John M. 1975. "An Exploration of Musical Timbre." Ph.D. thesis, Stanford University. Distributed as Department of Music Report No. Stan-M-2.

Iannino, Anthony, and Shapiro, Stephen D. 1978. "A Survey of the Hough Transform and its Extensions for Curve Detection." In *Proc. IEEE Conf. Pattern Recognition and Image Processing*. Chicago, 1978, pp. 32–38.

Kelly, M. D. 1971. "Edge Detection in Pictures by Computer Using Planning." *Machine Intelligence* 6:397–409.

Knuth, D. E. 1968. "Semantics of Context-Free Languages." *J. Mathematical Systems Theory* 2:127–146.

Minsky, Marvin. 1963. "Steps toward Artificial Intelligence." In *Computers and Thought*, ed. Edward Feigenbaum and Julian Feldman. New York: McGraw-Hill, pp. 406–450.

Moorer, James A. 1973. "The Heterodyne Method of Analysis of Transient Waveforms." Artificial Intelligence Laboratory Memo AIM-208. Stanford: Stanford University.

Moorer, James A. 1976. "The Use of the Phase Vocoder in Computer Music Applications." *Journal of the Audio Engineering Society* 26(1/2):42–45.

Moorer, James A. 1977. "Signal Processing Aspects of Computer Music: A Survey." *Proceedings of the IEEE* 65(8):1108–1137.

Moorer, James A. 1980. Personal communication.

Moorer, James A.; Grey, John M.; and Strawn, John. 1978. "Lexicon of Analyzed Tones. Part 3: The Trumpet." *Computer Music Journal* 2(2):23–31.

Pavlidis, Theodosios. 1971. "Linguistic Analysis of Waveforms." In *Software Engineering*, vol. 2, ed. Julius T. Tou. New York: Academic Press, pp. 203–225.

Pavlidis, Theodosios. 1973. "Waveform Segmentation through Functional Approximation." *IEEE Transactions on Computers* C-22(7):689–697.

Pavlidis, Theodosios, and Horowitz, Steven L. 1974. "Segmentation of Plane Curves." *IEEE Transactions on Computers* C-23(8):860–870.

Pavlidis, Theodosios, and Maika, A. P. 1974. "Uniform Piecewise Polynomial Approximation with Variable Joints." *Journal of Approximation Theory* 12:61–69.

Phillips, G. M. 1968. "Algorithms for Piecewise Straight Line Approximations." *Computer Journal* 11:211–212.

Portnoff, M. R. 1976. "Implementation of the Digital Phase Vocoder Using the Fast Fourier Transform." *IEEE Transactions on Acoustics, Speech, and Signal Processing* ASSP-24:243–248.

Risset, Jean-Claude, and Mathews, Max V. 1969. "Analysis of Musical-Instrument Tones." *Physics Today* 22(2):23–30.

Rosenfeld, Azriel. 1969. *Picture Processing by Computer*. New York: Academic Press.

Shirai, Yoshiaki. 1973. "A Context-Sensitive Line Finder for Recognition of Polyhedra." *Artificial Intelligence* 4(2):95–119.

Shirai, Yoshiaki. 1978. "Recognition of Real-World Objects Using Edge Cue." In *Computer Vision Systems*, ed. Allen R. Hanson and Edward M. Riseman. New York: Academic Press, pp. 353–362.

Stone, Henry. 1961. "Approximation of Curves by Line Segments." *Mathematics of Computation* 15:40–47.

Strong, William, and Clark, Melville. 1967. "Synthesis of Wind-Instrument Tones." *Journal of the Acoustical Society of America* 41(1):39–52.

Symons, M. 1968. "A New Self-organising Pattern Recognition System." *Conference on Pattern Recognition*. Teddington, pp. 11–20.

Tenney, James (with Polansky, Larry). 1978. *Hierarchical Temporal Gestalt Perception in Music: A "Metric Space" Model*. Toronto: York University.

Tou, Julius T., and Gonzales, Rafael C. 1974. *Pattern Recognition Principles*. Reading, Massachusetts: Addison-Wesley.