**Nick Bailey[*] and David Cooper[†]**
[*]Department of Electronic and Electrical Engineering
[†]Department of Music
The University of Leeds
Leeds LS2 9JB, UK
n.j.bailey@leeds.ac.uk
d.g.cooper@leeds.ac.uk

# Perceptually Smooth Timbral Guides by State-Space Analysis of Phase-Vocoder Parameters

Sculptor is a phase-vocoder-based package of programs that allows users to explore timbral manipulation of sound in real time. It is the product of a research program seeking ultimately to perform gestural capture by analysis of the sound a performer makes using a conventional instrument. Since the phase-vocoder output is of high dimensionality—typically more than 1,000 channels per analysis frame—mapping phase-vocoder output to appropriate input parameters for a synthesizer is only feasible in theory. For example, a lookup table of dimension 1,024 with only 16 points on each axis contains more than $10^{1,200}$ entries. Even using simplex-based storage schemes (Bowler et al. 1990), it is clear that the number of dimensions must be reduced if direct gestural capture is to become practical.

State-space models promise to reduce the dimensionality of the problem, but the coefficients they produce are not well correlated to the acoustic input in the way that a spectrum generated by a phase vocoder is. The need arose to integrate within a real-time framework the facility to manipulate signals in both the frequency and Laplace domains.

Sculptor uses state-space techniques to model the formant characteristics of a sound. It allows the user to use both spectrum- and filter-based resynthesis models, which can be edited and auditioned in real time. The source code of the package is freely available, allowing researchers to develop further functionality. Because the package combines and simultaneously presents the interfaces associated with filter-based and spectral methods, new algorithms can be tested in the appropriate domain.

Computer Music Journal, 24:1, pp. 32–42, Spring 2000
© 2000 Massachusetts Institute of Technology.

## Background

The phase vocoder has become widely used as a tool in computer music over the two decades since Michael Portnoff's article describing its digital implementation using FFTs (Portnoff 1976; Moorer 1978; Dolson 1986). The phase vocoder offers a convenient means of analyzing and resynthesizing sound in terms of perceptual correlates. In particular, it mitigates against the inherent limitation of Fourier analysis, namely, the dependency of frequency resolution on window length. The frequency separation of FFT analysis bins is equal to the sampling frequency divided by the number of samples in the analysis window; thus, the longer the window, the finer is the effective frequency resolution. In order to have a frequency resolution of, for example, 0.1 Hz, a 10-sec window is required, making a real- or near-real-time system impossible, given the intrinsic delay and the considerable computational load.

The phase vocoder makes use of the phase as well as the magnitude information provided by the Fourier transform. Given two overlapping analysis frames, it is possible to calculate the phase of any output bin from the presented frequency and the time delay between the two frames. As long as only a single partial lies within any output bin, and the spectral content is effectively constant across the overlapping frames ("quasi-stationary"), any advance or retard of the phase value of the bin in the second frame relative to the same bin in the first one will provide a much more accurate estimate of the frequency of the partial the analysis bin contains. For example, given a sampling rate of 44,100 Hz and a Fourier bin with a center frequency of 441 Hz, it takes 100 samples to complete a single cycle, during which the phase will have advanced by $2\pi$ radians. If the equivalent Fourier bin is examined at an overlap of 100 samples (by moving the sampling window forward by 100 samples) and the frequency is exactly 441 Hz, the phase angle should have advanced by $2\pi$ radians relative to the same bin in the previous frame. Since $2\pi$ radians is a full cycle, the phase of that bin will appear to be the same in both analysis frames because of phase wrapping. If the phase angle is greater than 0, then the actual frequency must be higher than 441 Hz, and vice versa. Within the constraints of at most one partial to a bin and acceptable signal-to-noise ratio, the frequency of that partial can be found to a degree of accuracy, which exceeds the FFT bin spacing (Puckette and Brown 1998).

Several packages are currently available for phase-vocoder-based modification of sound. Csound, the widely used audio-processing system developed by Barry Vercoe of MIT, includes a range of tools developed by Richard Karpen to manipulate soundfiles using phase-vocoder analysis (Vercoe 1992; Fitch 1998). These include time-stretching without pitch change, transposition maintaining original duration, and spectral interpolation and rescaling. The Moore-Loy CARL phase vocoder is implemented as part of the UK Composers' Desktop Project (CDP), with a set of command-line tools for manipulation of analysis files (Moore and Loy 1983). Some 90 operations are available, including frame interleaving, spectral tracing, spectral sweeping, spectral shifting and stretching, and conventional vocoding (in which the magnitude data from one analysis file is superimposed on that of another) (Fischman 1997). The IRCAM Super Phase Vocoder provides graphical facilities through the AudioSculpt program for the manipulation of the phase-vocoder files, e.g., by allowing users to superimpose filter characteristics on the spectrogram representation of a sound.

The interfaces of the CDP and Csound implementations undeniably provide the greater flexibility and power to users, but also place greater demands on them, essentially requiring a computer program to be written. The AudioSculpt family of applications is certainly an impressive suite, but is not open-source, thus denying the academic community the opportunity to modify or extend its capabilities in a particular direction.

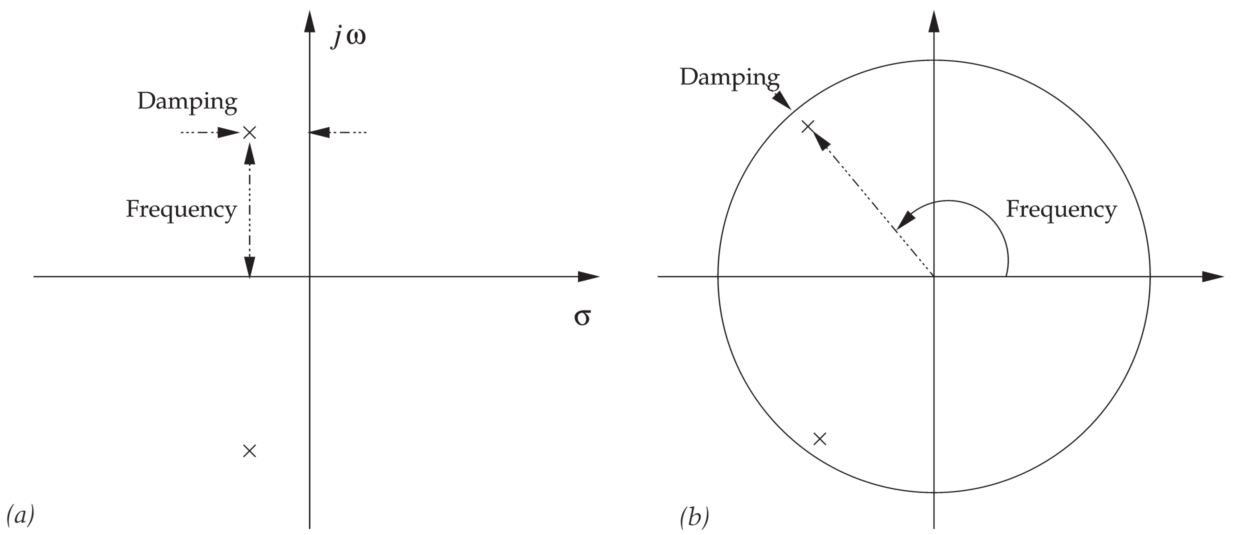
## Sculptor

Sculptor is an open-source package developed in C on the Linux operating system, and is easily portable to any system that supports X Windows and Sun's xview widget set. It enables the analysis, parameterization, and editing of sounds in real time using the phase vocoder, and it simultaneously provides graphical representation and spectral manipulation. A major feature of the software is its ability to derive all-pole models from quasi-stationary spectral samples using state-space techniques—elements that are explained in detail below. Models derived by the algorithm can be edited and auditioned in real time. This provides a particularly useful tool for the electroacoustic composer, in that it can generate perceptually smooth transitions between different timbres. Because the software is open-source and is written in a popular language, it also provides a framework within which users can add their own desired functionality.

Computer musicians are familiar with the concept of a spectrum. In formal terms, the spectrum can be regarded as the output of a Fourier transformation of a time series, or a sequence of time samples. The spectrum is useful for representing certain perceived properties of a sound using easily understood correlates, but it is in fact just one of many possible *transformations*.

In the same way that a Fourier transformation transforms a time series into a spectrum, the Laplace transform can be used to display certain other properties of a sound, starting from its time series. The Laplace transform is only applicable to continuous-time systems, whereas digital filters work in discrete time. We have chosen the Laplace representation to reinforce the idea that the system under analysis—the musical instrument or voice—operates in continuous time. Another program in the Sculptor system, Prism, generates a predictor model from spectral information from the phase-vo-

*Figure 1. A simple resonator, in the* s*-domain (left) and in the* z*-domain (right).*



coder analysis. This eliminates the need to compensate for the time quantization at the analysis stage.

The Laplace transform contains a few salient parameters that characterize the timbral information of a sound. While its output is highly concise and perceptually relevant, it still possesses meaningful physical analogs relating to the vibrations and resonance of a physical system. Such physical systems are essentially characterized by their *transfer functions*, which describe how an input, such as impulses of air from an opening and closing reed, work through the system to produce its output, e.g., by stimulating a column of air to vibrate.

Figure 1a represents a Laplace transform of the transfer function of a simple system. The transformation itself is in fact a surface above the graph. We are only interested in the places where the surface touches infinity or zero. In the simple system above, there are two infinities (*poles*), labeled with crosses. In any real system, these poles will occur in *conjugate pairs* at equal distances above and below the horizontal (*real*) axis. One can think of the horizontal axis as representing the amount of damping in the system. In Figure 1a, the poles are near the vertical (*imaginary*) axis and on the left-hand side. This tells us that the system is lightly damped (because of proximity to the imaginary axis) and stable (because the poles occur on the left-hand side). If this system were in a quiescent state and received an impulse, like the tapping of a wine glass, it would oscillate for some time before decaying. Moving the poles to the left increases the damping, from, say, that of a wine glass, through the sound of a beer glass, to a decay so rapid that only a dull thud would remain. Increasing the vertical distance between the poles broadly has the effect of changing the natural frequency of oscillation: a small wine glass with a high pitch is described by poles that are further separated than those for a larger glass. The timbral characteristics of the system are contained in the number and position of these poles. The Laplace transformation is linear, so that if the stimulation is doubled in intensity, the height of the surface away from the graph in the direction of the reader doubles also. However, the position of the poles remains constant. In short, the glass struck with a greater force sounds louder, but it has the same fundamental timbre.

Laplace transformations describe the behavior of continuous-time systems where the forces that cause oscillations are continuously variable as the system moves; *z-transforms* describe sampled-time systems in which updates are made at regular discrete intervals based upon observations of the values at a previous time, as is the case with digital filters. The Laplace transform and the *z*-transform are closely related by the substitution $z = e^{st}$, because of the effect of a sample unit delay in

Laplace space. The transfer function of the system associated with Figure 1a in $z$-space is shown in Figure 1b. Note that the entire left-hand side of the $s$-plane is mapped inside the unit circle in the $z$-plane. The Laplace transformation is used by Prism because of the way the analysis is performed, starting with the spectrum and not with the time series. Conversion to $z$-space is made after the parameters have been manipulated in the $s$-domain, although this is purely for programming convenience, and not for any theoretical reason.

Knowing the Laplace transformation of a system's response, it is a relatively simple matter to construct a digital filter that behaves in the same way. The $Q$ of the filter (its center frequency divided by its bandwidth) is related to the pole's characteristic frequency and damping. In a real system, several pole pairs will exist, and the overall response is the sum of the contribution of each simple filter.

## Placing the Poles

The positions of the poles of a system characterize its response to excitation (i.e., its impulse response). The poles essentially define a model for an instrument, albeit one which is not strongly correlated to the instrument's physical operation, as is the case with physical models. This is true because, while each system posses a unique transfer function, it is not true that each transfer function describes a unique system. In the Laplace model, one can think of the poles as characterizing the system resonance, in the same way that the position of the formants in the spectrum of a spoken sound determines the vowel. The resonances of an instrument determine the spectral envelope, but the absolute position of the partials is also related to the excitation.

One method of extracting the pole positions from a time series is to use linear prediction (Makhoul 1975). The linear predictor is a filter that attempts to "best guess" the next sample in a time series based on several previous samples (typically in the order of 10–20). A *predictor filter* is constructed that takes the statistics of a signal into account, and the subsequent sample is predicted on the basis that it will minimize the mean-squared error of the predicted sample against the real sample for any sequence of samples that have occurred in the recent past. The process essentially separates the system's resonances from its excitation, so that if the poles of the filter can be found, then the system can be modeled to an arbitrary degree of accuracy.

However, Prism uses the Levinson-Durbin algorithm (of which Makhoul [1975] provides an excellent overview) to find the poles recursively, stopping when the model error has reached a predefined level. By setting the termination threshold appropriately, the poles thus calculated describe the formants present in the spectrum; if the predictor were allowed to continue to much higher orders, it would probably identify the partials as individual formants or else fail because of numerical inaccuracy. The Levinson-Durbin algorithm enables the predictor filter to be determined from the short-term autocorrelation of a time sequence, and this is easily available to the Sculptor suite because it is simply the inverse Fourier transformation of the signal's power spectrum. Algorithms exist with superior performance, such as Discrete All-Pole Modeling (El-Jaroudi and Makhoul 1991), but these include considerable associated computational overhead. Because the formants are intended to be edited interactively by the user, the simpler Levinson-Durbin algorithm was adopted.

By considering $n$ autocorrelation values, one can generate an $n$th-order predictor filter to guess the next sample. The predictor filter is a recursive filter, as shown in Figure 2.

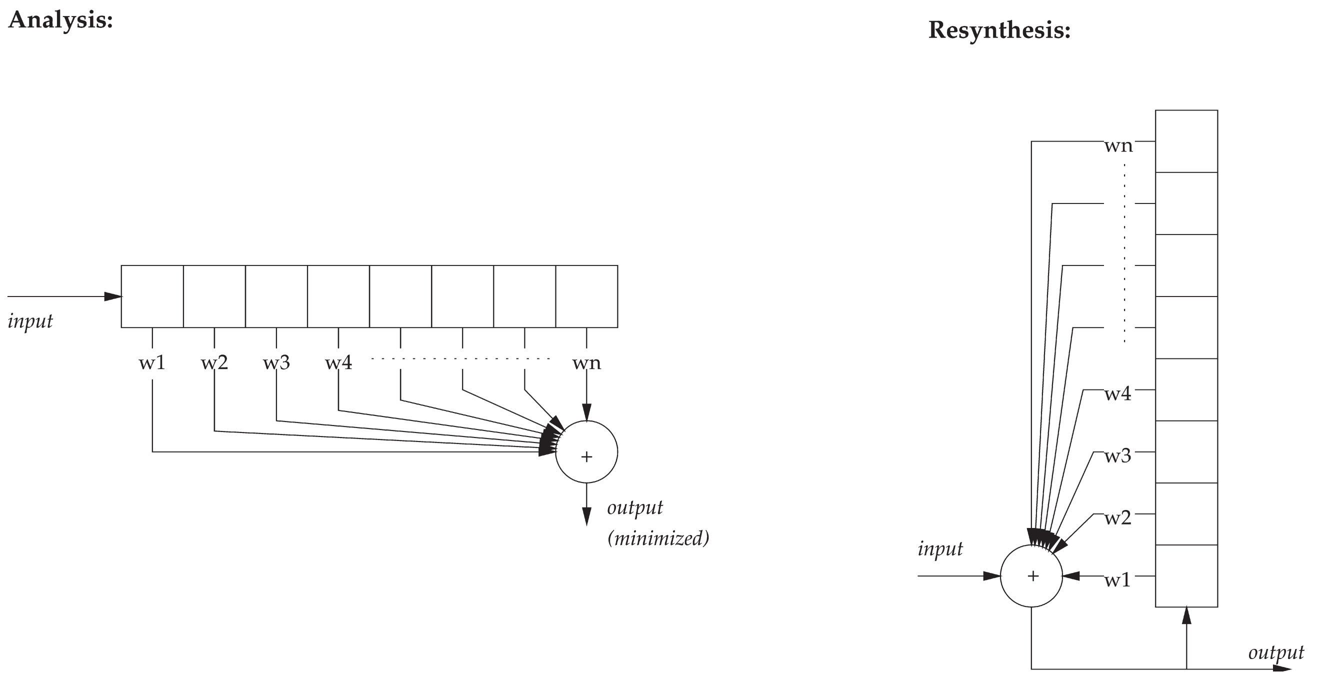
The $n$th-order predictor will have $n$ poles in its Laplace transform, each of which represents a simple resonator contributing to the system response. A tenth-order predictor, for example, can locate five formant regions in Laplace space. However, this is of little immediate use in attempting to decompose the system into the simplest possible model.

Because the model realization is carried out in discrete time, the transfer function of the filter is a polynomial in order $n$ expressed in the $z$ domain:

$$H(z) = \frac{1}{a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}}$$

where $z^{-1}$ represents a sample delay. The presence

Figure 2. Linear prediction architecture.



of a pair of poles in the Laplace transform maps onto a pair of poles in the $z$-domain, as previously described. However, in order to extract useful information from the system, we would prefer to express the transfer function as the sum of many simple second-order filters, each representing the frequency and significance of a formant region. This will enable us to represent the spectral envelope using a series of formants, and, using Prism, manipulate the parameters of each of these formants while hearing the consequences of doing so in real time. There are two significant advantages to this approach over direct access to the polynomial coefficients $a_n$: adjusting the formant positions individually generates perceptually smooth timbral guides; and the user interface is easily designed to prevent the user from placing poles in regions of instability (the right-hand side of the $s$-plane, or outside the circle in the $z$-plane).

Using state-space techniques, it is possible to perform the decomposition into second-order filters. First, it is necessary to represent the predictor filter in *state-space form*. This will involve considering the filter as a state machine and writing down a matrix equation describing the transition between the current state and the next state. Consider once more the resynthesis filter in Figure 2 and its behavior when a new excitation sample is presented. The output of the filter is still given by the weighted sum of the current and delayed inputs, but all of the stored values will "march up" the shift register, making space for the previous output. The ones in the trailing diagonal perform the shifting operation, while the top row calculates the next output sample. If the filter is considered to be a vector of predictor coefficients $[v_1. . .,v_n]^T$ and the new input sample is $x[n]$, then the state equation can be written as follows:

$$s_{n+1} = \begin{pmatrix} v_1 & v_2 & \cdots & \cdots & v_n \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix} S_n + \begin{pmatrix} x[n] \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

This is known as the *companion form* of the state-space matrix.

The system is described by the state-space matrix, but the matrix itself describes more than the response of the predictor filter: it also describes the filter's structure. It is possible to build other filters with identical responses but with completely different structures. Information about the response of the system is contained entirely within the state-space matrix's *eigenvalues*. Subject to certain conditions, a matrix has a set of vectors which, when operated on by the matrix, do not change their directions. These are called the *eigenvectors* of the matrix. Multiplying a matrix and one of its eigenvectors simply scales the eigenvector; this scale factor is referred to as an eigenvalue of the matrix. Because the predictor filter's response is defined completely by the eigenvalues, any rearrangement of the state-space matrix which retains the eigenvalues will not change the characteristics of the predictor filter. This is equivalent to changing the organization of a filter without changing its response.

One such rearrangement of the state-space matrix yields the so-called *Jordan canonical form*:

$$\begin{pmatrix} \lambda_1 & 0 & \cdots & & \\ 0 & \lambda_2 & 0 & \cdots & \\ \vdots & \ddots & \ddots & \ddots & \\ & & 0 & \lambda_{n-1} & 0 \\ & & & 0 & \lambda_n \end{pmatrix}$$

where λ are the (complex) eigenvalues of the state-space matrix in its companion form. This is, to say the least, useful, because the independent eigenvalues are also the positions of the poles in Laplace space, and the structure of the Jordan canonical form implies decomposition into a set of *n/2* independent second-order filters, the outputs of which are summed together to produce a response equivalent to that of the original predictor. In fact, the matrix is not guaranteed to be diagonal: it may be that the second-order sections are not independent, but for most spectra arising from sampled audio input, experimentation shows that the matrix practically always reduces to this form. A more rigorous treatment of this issue is given elsewhere (Bailey and Cooper in press).

## A Practical Implementation

Sculptor is written in C, and makes use of Fortran math library calls. It compiles and runs "as is" on Linux platforms with xview libraries installed. Machine-specific routines (mostly audio output) are segregated for easy porting. As of version 3.2, the Sculptor package contains five programs. The first, Analyse, is a batch-mode, command-driven program that generates phase-vocoder data files in the appropriate format, given a Sun/NeXT audio file as input. Prism, a playback program which resynthesizes the phase-vocoder data file, permits simple speed variation and various other effects using command-line flags. Xprism is a playback program with interactive facilities for pitch, rate, and formant manipulation. Finally, two simple programs, Fermions and Timbres, demonstrate how Prism can be used as a real-time, general-purpose additive synthesizer.

Figure 3 shows a screen shot of Prism. The upper section of the screen provides a spectrogram view, while the lower portion shows the spectral content on the right-hand side with tracked formants superimposed. (In this example, four main formant regions have been identified.) An additional dialog box (not shown in the example) allows the user to edit individual formant regions by adjusting the center frequency, damping factor, and normalization gain of each of the second-order filters. The graph at the lower left-hand side of the screen shot plots the pole positions; the vertical axis indicates center frequency ($\omega/2\pi$), and the horizontal axis displays damping ($\zeta$). This representation is preferred over a simple mapping in the *z*-plane, because the latter representation groups all poles of timbral significance near the unit circle, yielding a poor display. Even with our alternative representation, which is more akin to the *s*-plane, nonlinear axes had to be used to present the results effectively.

The source code for the Sculptor package has been made publicly available for other researchers

Figure 3. An editing session using Prism.

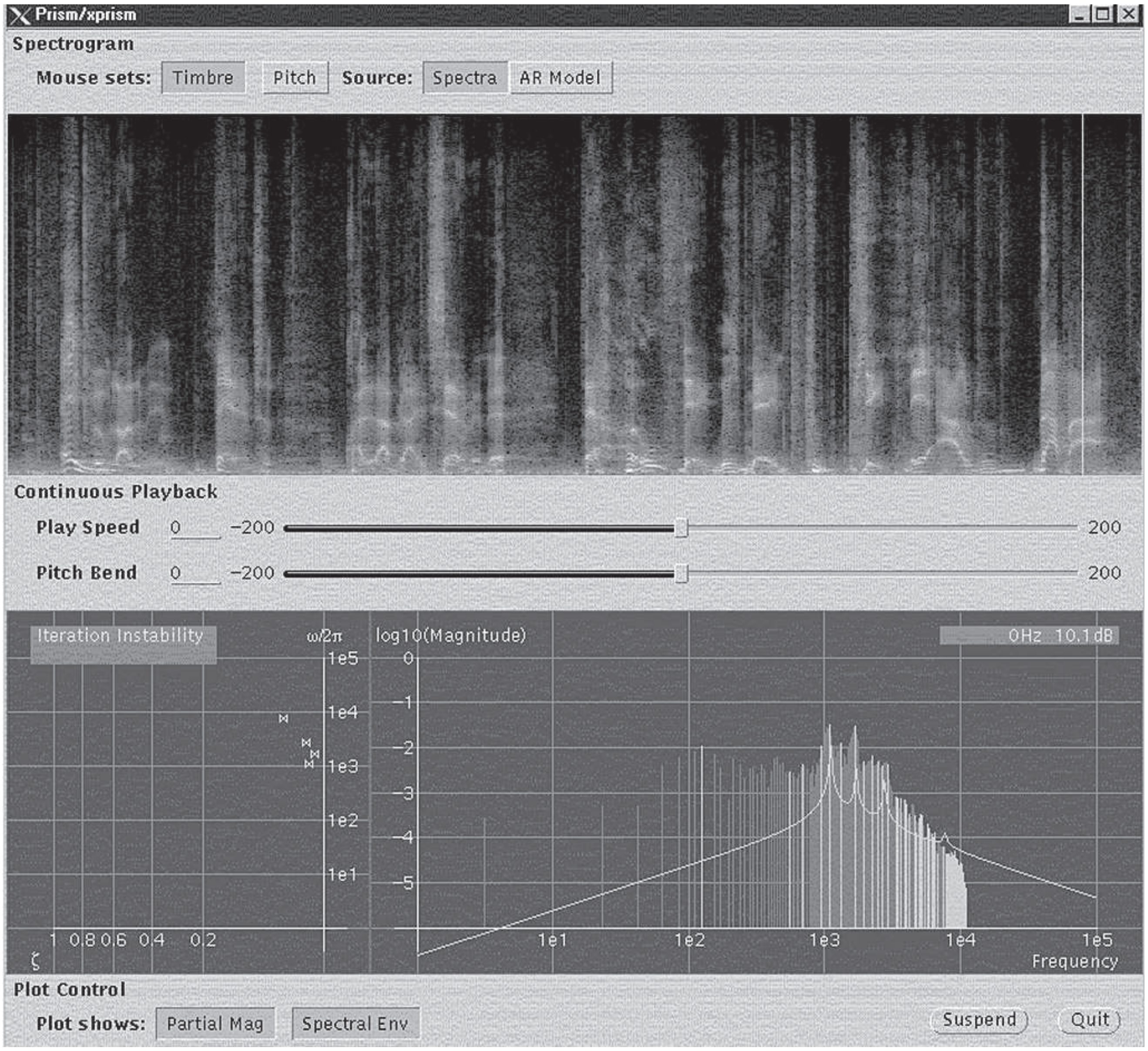
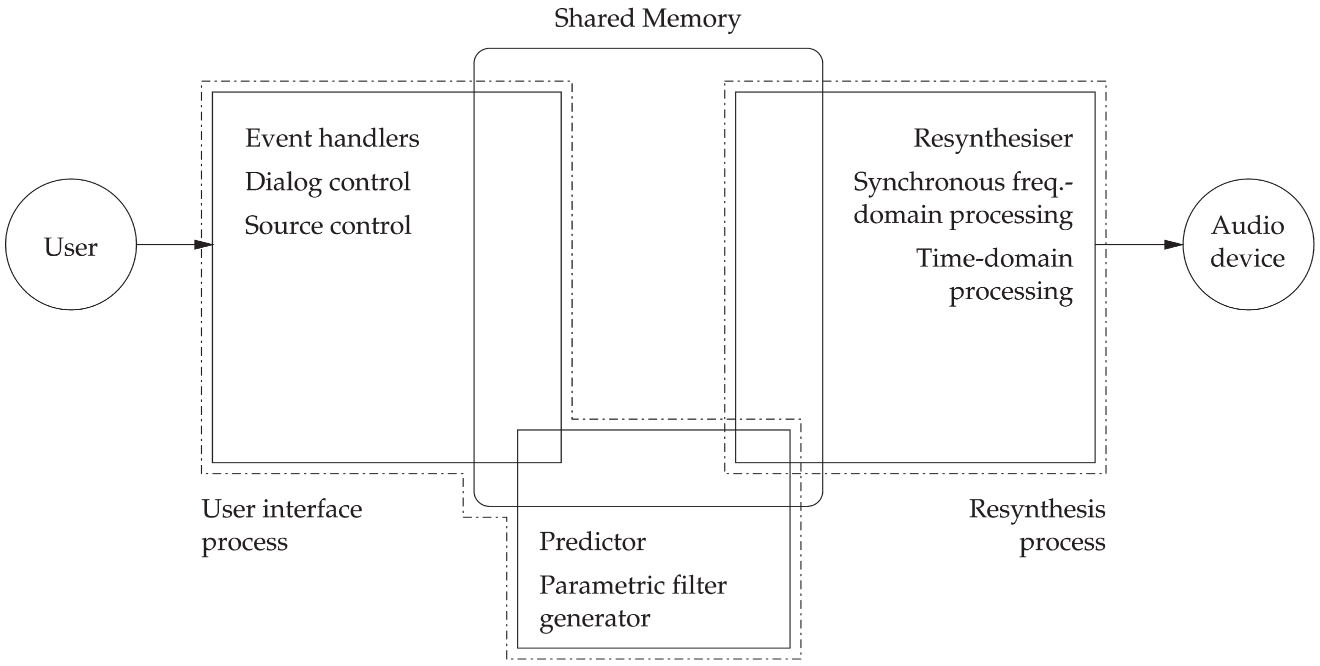


Figure 4. Interprocess communication in Prism.



using a Linux platform with xview who wish to experiment in an environment where both spectral and Laplacian representations of sampled audio are available concurrently. The structure of the most interesting program, Prism, is shown in Figure 4.

Analyse reads a Sun/NeXT format audio file to build a file of phase-vocoder data. Even though UNIX operating systems are generally not noted for real-time performance ability, Prism is capable of moderate real-time interaction, even on very-low-power processors. Indeed, the choice of the Linux platform was made because of its efficient implementation of concurrent solutions and its superior software-development environment. When the project began, a 40-MHz 80386 computer without a floating-point coprocessor was found to be capable of real-time 8-bit resynthesis at sampling rate of 8,000 Hz. The same algorithms compiled using Microsoft operating systems and compilers returned such low benchmark results as to make the project infeasible. Table 1 shows the results of repeatedly running the core FFT algorithms on a 50-MHz 486DX machine with 8 MB of memory; the fastest time is commensurate with real-time output at a sampling rate of approximately 21 kHz. The difference in performance was so pronounced that the cause was not investigated, although a significant factor was likely to be that Microsoft systems of the time were still producing 16-bit Intel binaries.

**Table 1. Benchmarks for Prism's FFT algorithm.**

| | |
|---|---|
| Linux: gcc with -O2 optimization | 9 min, 43 sec |
| Microsoft DOS EXE "fastest" code-generation option | 40 min, 11 sec |
| Microsoft QuickWin, "fastest" code-generation option | 29 min, 14 sec |

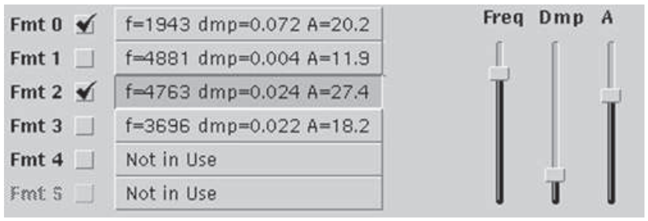
Having chosen the implementation platform, the issue of obtaining a sufficiently prompt real-time response was addressed. On startup, Prism checks to see whether it is running in interactive mode, and if so, starts two processes communicating via a shared memory block. The entire analysis file is read into this block. The resynthesis process begins to produce a stream of phase-vocoder output using parameters taken from the shared memory block. As it does so, it overlaps, windows (with a raised cosine window function), and adds the output before writing it directly to the audio device. Movement through the spectral data is achieved by incrementing the spectrograph index at the end of each resynthesis call by an amount the control process sets asynchronously.

Using this method, it is possible for the control process, as a result of user-interface events, to set a playback rate through the analysis data, or, by setting an increment of zero, to manipulate the spectrogram index directly to force resynthesis from data at an absolute position in the spectral array. The user interface supports the former technique by providing a "play speed" slider; it also permits the user to click and drag on the spectrogram itself. This strategy delegates problems associated with the real-time output requirement to the Linux kernel, and in practice it performs particularly well. The machine on which Prism is currently being developed has a twin processor architecture, for which the two-process shared-memory paradigm is particularly well suited.

In addition to allocating sufficient shared-memory resources for the storage of the spectrogram and the shared control block, additional buffers of a size equal to the FFT used in the resynthesis process are also reserved. These can be used for the synthesis of arbitrary spectra, such as those produced by the linear predictor module.

Because the spectral information is always available and is the basis for the continuous resynthesis, a somewhat nonstandard route has been adopted for the calculation of the prediction coefficients. In a speech-linear-predictive-coding (LPC) system, a *residual* or error signal is calculated, which helps determine whether the segment of waveform under consideration represents a voiced or unvoiced sound. When the waveform is resynthesized, an appropriate excitation signal is used to ring the filter according to the state of the residual and the particular system used (generally either Codebook-Excited Linear-Predictive Coding or the simpler Residual-Excited Linear Prediction) (Roads 1996). However, the system described in this article does not calculate a residual signal at all, and a repeated-pulse excitation (RPE) stimulus is used to ring the synthesis filter.

Figure 5. Manipulating a predictor filter.



| | | |
|---|---|---|
| Fmt 0 | ✓ | f=1943 dmp=0.072 A=20.2 |
| Fmt 1 | ☐ | f=4881 dmp=0.004 A=11.9 |
| Fmt 2 | ✓ | f=4763 dmp=0.024 A=27.4 |
| Fmt 3 | ☐ | f=3696 dmp=0.022 A=18.2 |
| Fmt 4 | ☐ | Not in Use |
| Fmt 5 | ☐ | Not in Use |

Clicking at any point on the spectrogram causes the AR algorithm to use that spectrum to generate a predictor filter. Selecting the filter as the resynthesis source causes the displayed filter response to be used as an envelope for partials separated by the pitch period. This is equivalent to the RPE method familiar from speech processing, and the "pitch-bend" slider presented by the user interface is used to set the fundamental frequency (see Figure 5).

At this point, a second dialog box is presented that permits interactive variation of the parameters associated with the predictor filter. The user thus exercises some timbral control over the resynthesized sound by manipulating a few parameters. The formants are also explicitly ordered so that when interpolation is required, formants moving from one position to another may be distinguished from formants that fade to zero and are replaced by new formants of steadily increasing amplitude.

A simple extension of the basic model allows the user to save timbral snapshots which can be kept as an ordered series, providing a means of perceptually smooth morphing from one timbre to another. In this case, the formant parameters derived from analysis provide the set of timbral guides alluded to in this article's title.

## Extending Sculptor Further

Because Sculptor is an open-source package, it may be extended by anyone in the computer music community with the appropriate programming skills, without the need for "forum" membership. We hope that the facility to easily add algorithms that can operate synchronously or asynchronously in the frequency, time, or Laplacian parametric domains will offer a genuinely useful resource to other researchers.

The authors have already identified a number of possible extensions of the core functionality of the software. In particular, consideration has been given to the placing of *zeros* as well as poles. The current model is prone to some extent to oversmoothing of the spectral envelope, and recent advances in the area of the so-called rational covariance extension problem suggest that this tendency may be ameliorated (Byrnes, Gusev, and Lindquist 1998). In fact, zeros arising from the vibrational nulls of the system generating the sound are already implicitly included in the user interface presented here. In addition to controlling the center frequency and $Q$ of each pole, users can also control another parameter, $A$, which determines the maximum gain of the resonator. Readers familiar with the dynamics of vibrating systems know that the quality factor $Q$ not only relates center frequency to bandwidth, but it also determines the maximum gain of the filter. The Sculptor package is able to isolate $Q$ and $A$ adjustments, because the $A$ slider changes the amplitude of the spectral peak without modifying its center frequency or bandwidth. This can be considered in two equivalent ways: the prefiltering of the RPE stimulus to modify the energy available to the resonator; and the addition of a zero of similar characteristic frequency to but different damping factor from the pole. Depending on forthcoming results which should indicate the sufficiency or otherwise of the parameter set presented here, it may be necessary to modify Prism to deal with zeros more explicitly.

A second obvious area for further investigation is that of formant-corrected pitch shifting. The failure to formant-correct transposed samples results in the so-called chipmunk effect (named for American cartoon characters whose voices were created by speeding up adult speech). The chipmunk effect becomes noticeable with transpositions as small as a minor third for many musical signals. Current implementations of formant-corrected transposition in the frequency domain tend to involve interpolation after the multiplication of input frequencies by an appropriate ratio (for example by 3/2 for a perfect fifth higher) and scaling the magnitudes of the output bins so that the original spectral envelope is superimposed (Bristow-Johnson 1995). It is probable that the method of location and categorization of formant frequencies described above may be useful in the development of an algorithm for smooth formant-corrected pitch shifting over extended frequency ratios. It is also hoped that Sculptor could provide potentially useful tools in the search for an objective analysis and classification of timbre involving the transition between steady-state portions of tones (Grey 1977; Wessel 1979; Pollard and Jansson 1982).

## References

Bailey, N. J., and D. Cooper, In press. "Sculptor: Exploring Timbre Spaces in Real Time." *Journal of the Audio Engineering Society.*

Bowler, I. W., P. D. Manning, A. Purvis, and N. J. Bailey. 1990. "On Mapping *N* Articulations onto *M* Synthesiser Control Parameters." *Proceedings of the 1990 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 181–184.

Bristow-Johnson, R. 1995. "A Detailed Analysis of a Time-Domain Formant-Corrected Pitch-Shifting Algorithm." *Journal of the Audio Engineering Society* 43(5):340–352.

Byrnes, C. I, S. V. Gusev, and A. Lindquist. 1998. "A Convex Optimization Approach to the Rational Covariance Extension Problem." *SIAM Journal of Control Optimization* 37(1):211–229.

Dolson, M. 1986. "The Phase Vocoder: A Tutorial." *Computer Music Journal* 10(4):14–27.

El-Jaroudi, A., and J. Makhoul. 1991. "Discrete All-Pole Modeling." *IEEE Transactions on Signal Processing* 39:411–423.

Fischman, R. 1997. "The Phase Vocoder: Theory and Practice." *Organised Sound* 2(2):127–145.

Fitch, J. P. 1998. "Release Notes for Csound Version 3.44." Program documentation, contained in the MS Help file covering Csound 3.48, distributed with the Winsound shell.

Grey, J. M. 1977. "Multidimensional Perceptual Scaling of Musical Timbre." *Journal of the Acoustical Society of America* 61:1270–1277.

Makhoul, J. 1975. "Linear Prediction: A Tutorial Review." *Proceedings of the IEEE* 63(4):561–580.

Moore, F. R., and D. G. Loy. 1983. *CARL Release Notes, CARL Start-up Kit*. Program documentation. La Jolla, California: Computer Audio Research Lab, Center for

Research in Computing and the Arts, University of California, San Diego.

Moorer, J. A. 1978. "The Use of the Phase Vocoder in Computer Music Applications." *Journal of the Audio Engineering Society* 26(1/2):41–45.

Pollard, H. F., and E. V. Jansson. 1982. "A Tristimulus Method for the Specification of Musical Timbre." *Acustica* 51:162–171.

Portnoff, M. R. 1976. "Implementation of the Digital Phase Vocoder Using the Fast Fourier Transform." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24(3):243–248.

Puckette, M. S., and J. C. Brown. 1998. "Accuracy of Frequency Estimates Using the Phase Vocoder." *IEEE Transactions on Speech and Audio Processing* 6(2):166–176.

Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, Massachusetts: MIT Press.

Vercoe, B. 1992. *Csound User's Manual*. Cambridge, Massachusetts: MIT Media Lab.

Wessel, D. L. 1979. "Timbre Space as a Musical Control Structure." *Computer Music Journal* 3(2):45–52.