MUS421 Lecture 9:
Spectral Audio Modeling Applications

Julius O. Smith III (jos@ccrma.stanford.edu)
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

June 27, 2020

Outline

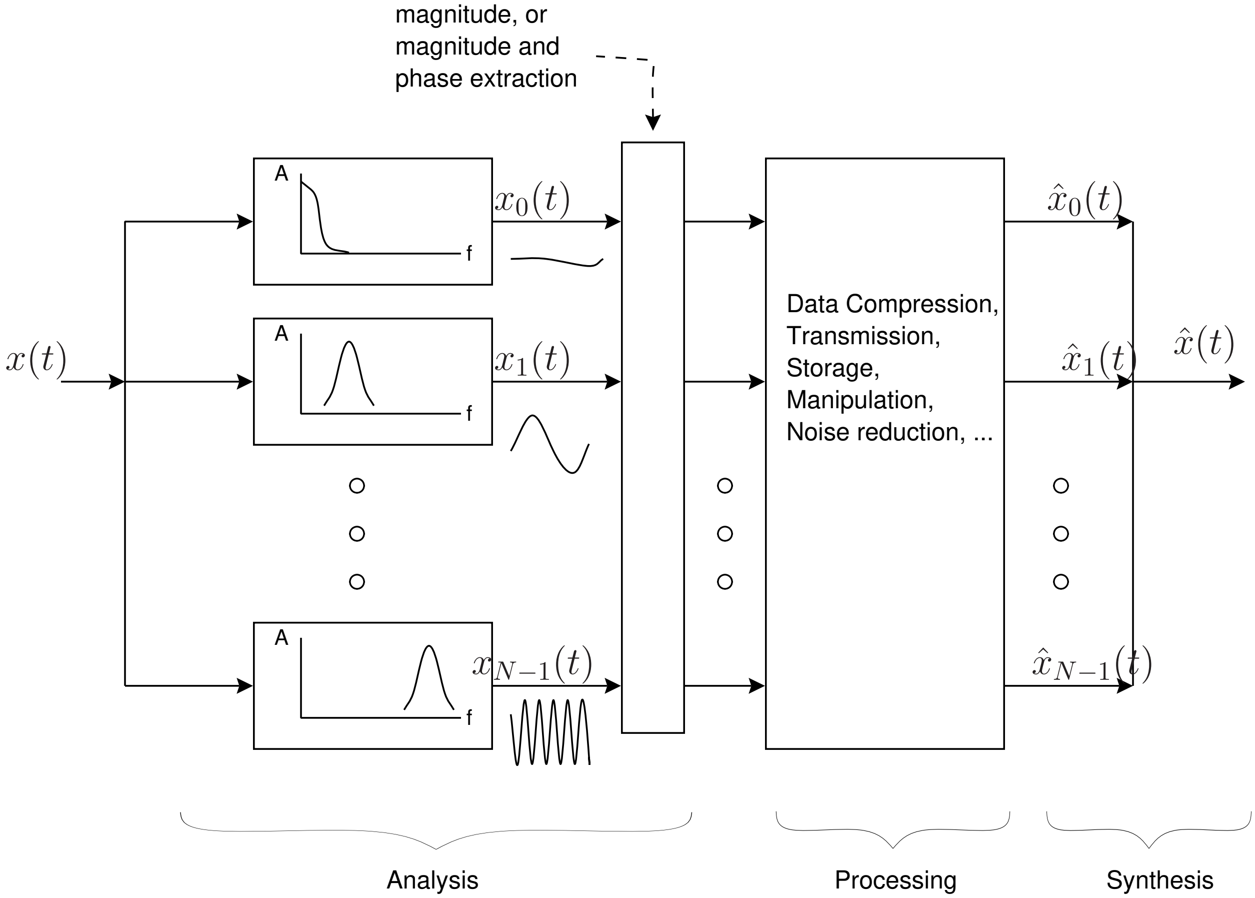
- *Analysis* for Additive Synthesis
  (in "recent historical order"):
  - Channel Vocoder
  - Phase Vocoder
  - Tracking Spectral Peaks across Time Frames
  - Sines + Noise Modeling
    "Spectral Modeling Synthesis (SMS)"
  - Sines + Noise + Transients

Spectral Modeling Overview

A large class of musical sounds can be modeled efficiently as sums of sinusoidal components ("tonals") and noise bands. It usually boils down into the following steps:

- Analysis in Frequency Bands over Time (determine the components)
  - Bandpass Filter Bank, or
  - Short Time Fourier Transform (STFT)
- Data Reduction (optional)
- Modification (optional)
- Synthesis
  - Bank of Oscillators (traditional additive synthesis), or
  - Inverse Fourier Transform

Filter-Bank Analysis/Modification/Resynthesis



1. Input signal is decomposed into subbands
2. Bands are processed (compressed, transformed, ...)
3. Bands are summed to form the output signal

The last step is called "filter-bank summation"'

Applications

Applications of spectral modeling include:

- Sinusoidal modeling of "tonals" for audio compression
- Colored noise modeled as filtered white noise
- "Easily transformable" audio representation
  - Time Scale Modification (TSM)
  - Frequency scaling (dual of TSM)
  - Cross-synthesis (*e.g.*, "talking rain")
- Pitch detection (detect "harmonic" relationships)
  - Pitch to MIDI conversion
  - Source separation
- Automatic transcription from sound to score
  (hard in general - nowadays a good problem for *neural networks*)
- Music synthesis and sound composition
  including powerful transformation techniques

Additive Synthesis

Overview

Additive synthesis is a technique in which a signal is reconstructed from a summation of "sinusoids". Each "sinusoid" has a time varying amplitude and frequency:

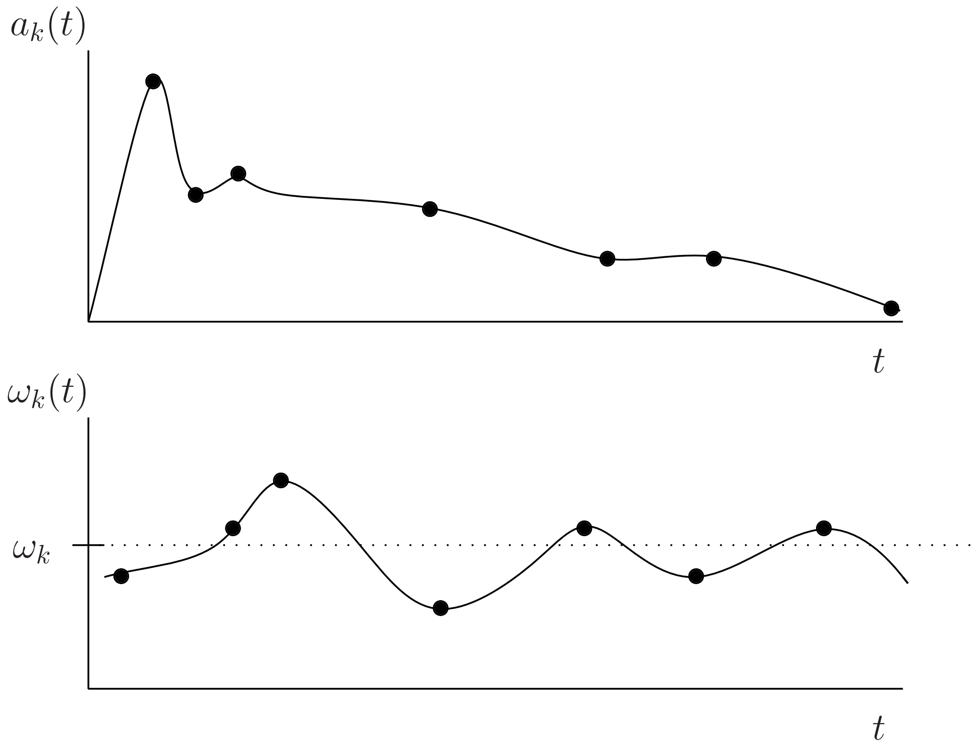
$$\boxed{y(t) = \sum_{i=1}^{N} A_i(t) \sin[\theta_i(t)]}$$

where

$$\begin{aligned}
A_i(t) &= \text{Amplitude of } i\text{th partial over time } t \\
\theta_i(t) &= \int_0^t \omega_i(t)dt + \phi_i(0) = \text{inst. phase} \\
\omega_i(t) &= \text{Inst. frequency of } i\text{th partial vs. time} \\
\phi_i(0) &= \text{Initial phase of } i\text{th partial at time 0}
\end{aligned}$$

and all quantities are real.

Typical Looking Amplitude and Frequency Envelopes

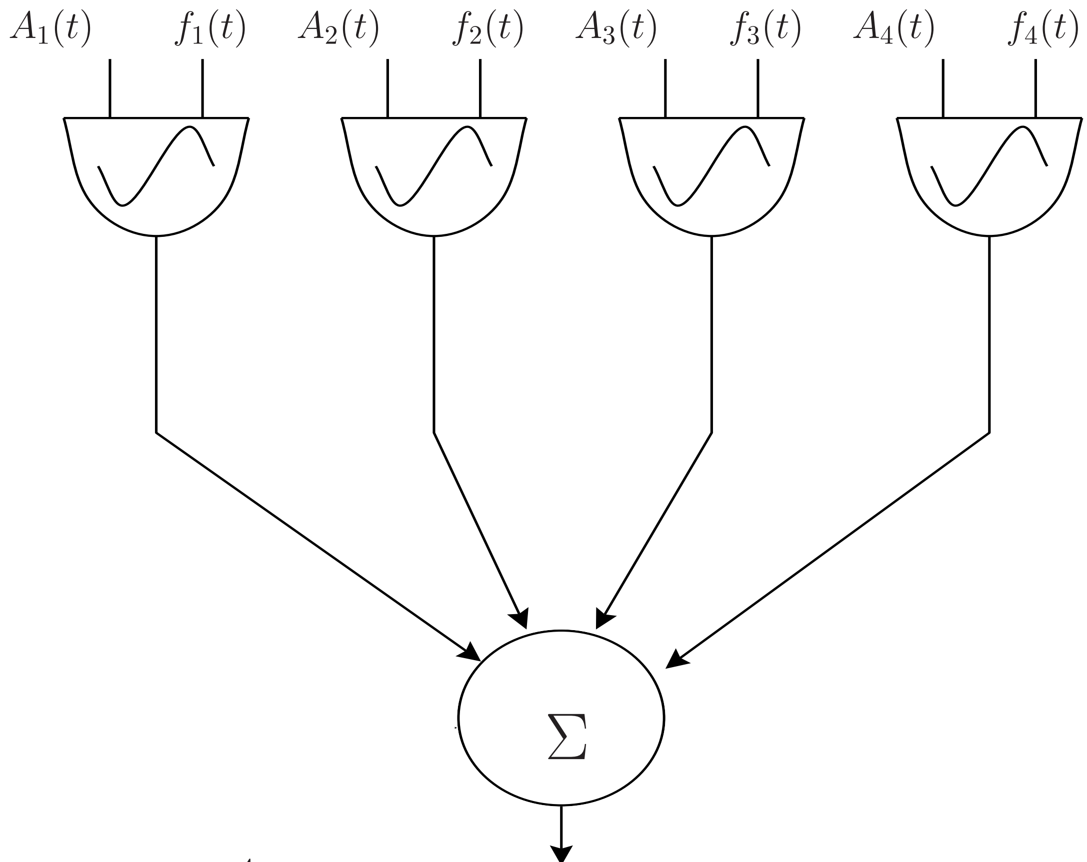


Applicability of Additive Synthesis

$$y(t) = \sum_{i=1}^{N} A_i(t) \sin \left[\int_0^t \omega_i(t) dt + \phi_i(0) \right]$$

- The sinusoidal signal model is efficient for *tonal* signals, such as voiced speech, steady-state wind instrument tones, plucked/struck strings, etc.
- *Inefficient* for *noise-like* signals, such as unvoiced speech, and the "chiff" portion of flute/organ tones $\Rightarrow$ *Sines+Noise* modeling (discussed later)
- *Inefficient* also for *attacks*, (sharp time-domain transients) such as in percussion, note onsets $\Rightarrow$ *Sines+Transients* modeling (discussed later)
- Most efficient when $A_i(t)$ and $\omega_i(t)$ are *slowly varying* (i.e., we really have a sum of quasi-sinusoidal components) and when $\phi_i(t)$ can be *neglected* altogether
- It has been well known since Helmholtz that modifications to the phases ϕ_i in a sum of sinusoids are usually not audible

Additive Synthesis Oscillator Bank



$$y(t) = \sum_{i=1}^{4} A_i(t) \sin\left[\int_0^t \omega_i(t)dt + \phi_i(0)\right]$$

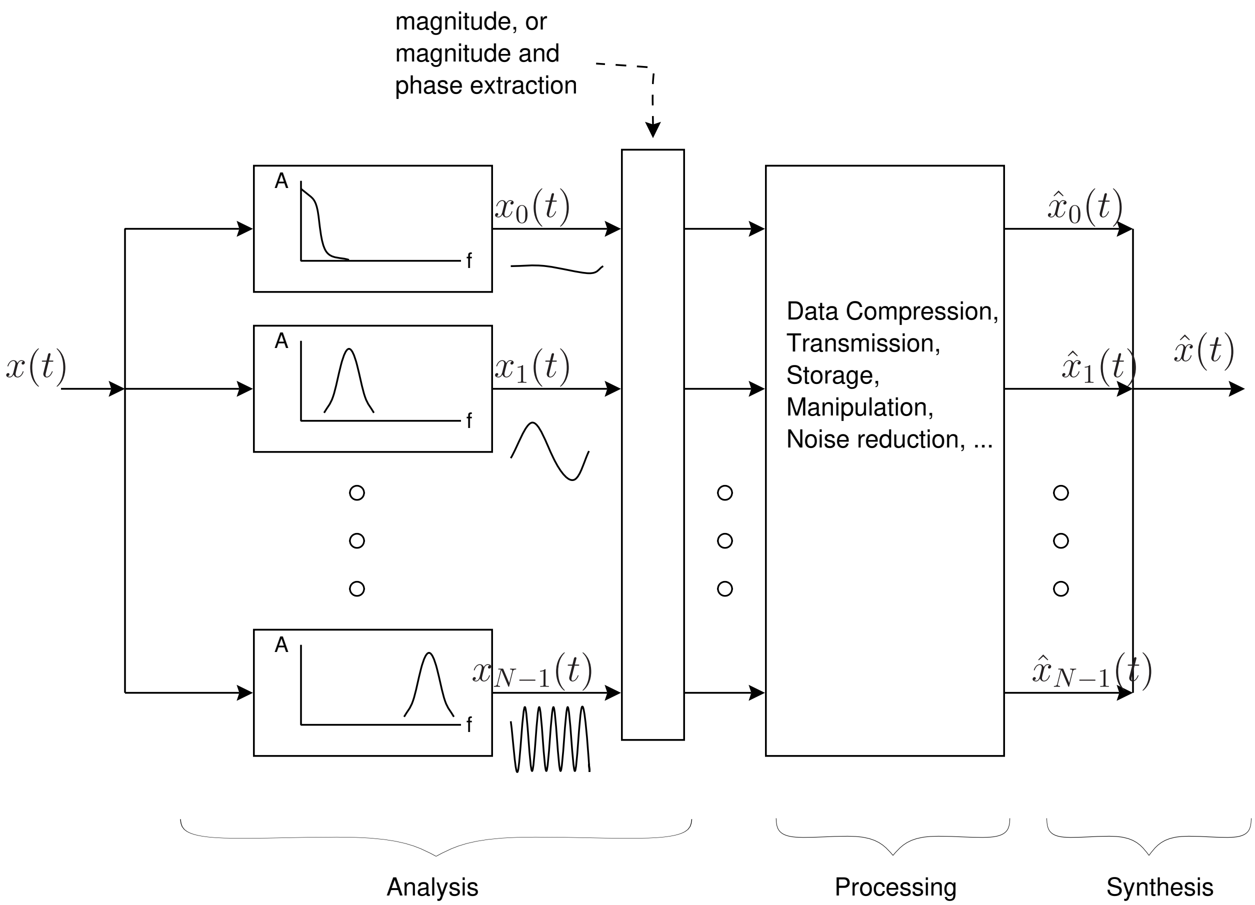
- In order to reproduce a signal, we must first *analyze* it to determine the amplitude and frequency *trajectories* for each sinusoidal component. We may or may not want the phase information.
- Vocoder often used for analysis (before STFT)

Vocoders

- 'Voice Coder'
- Example of an analysis / synthesis system
  - Analysis done by filterbanks
  - Synthesis = additive
- Developed at Bell Labs (late 1930s)
- Used for speech coding and transmission
  - Data compression
  - Noise reduction
  - Reverberation suppression
- Channel Vocoder
  - Determines only the magnitude of the signal in each filter band (historically an *analog* filter bank)
- Phase Vocoder
  - Determines both magnitude and phase in each band slice (STFT *digital* filter bank)
- A bit of history[1]

[1] https://ccrma.stanford.edu/~jos/sasp/Dudley_s_Channel_Vocoder.html

Vocoder Block Diagram



Input signal is decomposed into subbands.

- *Channel Vocoder:*
  Form *amplitude envelope* in each band
- *Phase Vocoder:*
  Compute *amplitude and phase* envelopes versus time

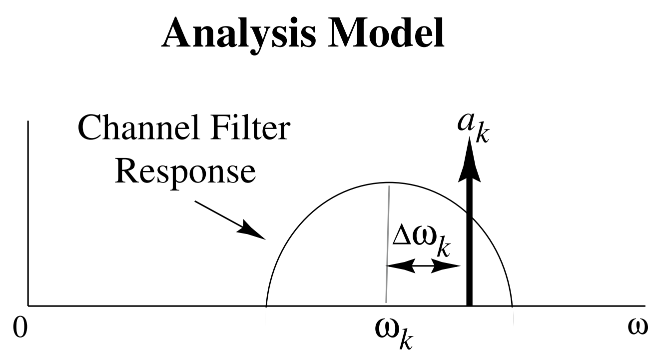
Vocoder Channel Model

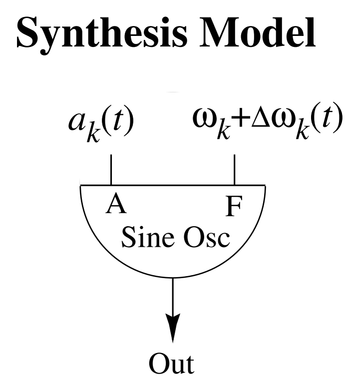
If we assume that we have *at most 1 sinusoid* with time varying parameters in each channel, then we can write down the following expression for $x_k(t)$, the signal in the k^{th} subband:

$$x_k(t) = a_k(t)\ \cos[\omega_k t + \phi_k(t)]$$

where

- $a_k(t)$ = amplitude modulation
- ω_k = fixed channel center frequency
- $\phi_k(t)$ = phase (or frequency) modulation

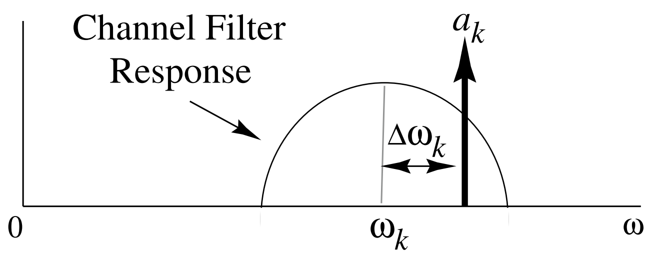




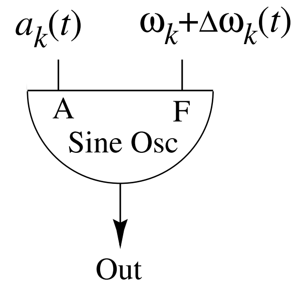
- Using these parameters, we can re-synthesize the signal via oscillator summation
- A *nonparametric* filter-bank signal representation is replaced by a *parametric* (sum-of-sinusoids) signal model

Vocoder Channel Model, Cont'd

Analysis Model



Synthesis Model



$$x_k(t) = a_k(t)\,\cos[\omega_k t + \phi_k(t)]$$

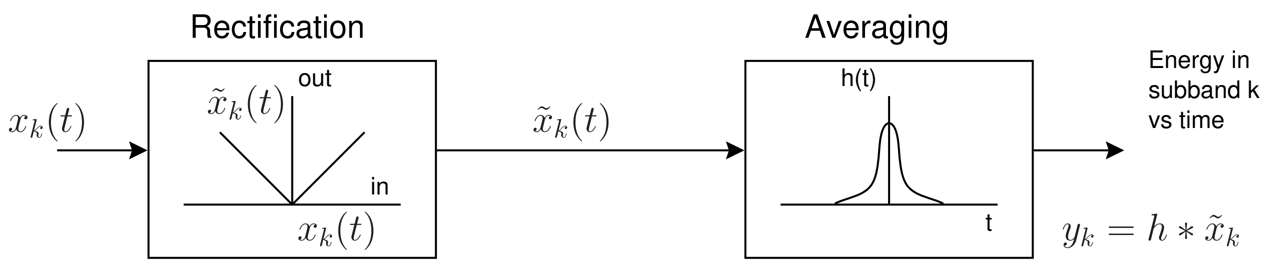
Typically, the instantaneous phase modulation $\phi_k(t)$ is differentiated to obtain instantaneous frequency deviation:

$$\Delta\omega_k(t) \;\stackrel{\Delta}{=}\; \frac{d}{dt}\,\phi_k(t)$$

Computing Vocoder Parameters

The *channel vocoder* determines only the *amplitude envelope* $a_k(t)$ of the signal in the kth band for each k

Classic analog *envelope follower:* Rectification followed by lowpass filtering (average magnitude):



- This envelope follower is often used to recover amplitude modulation
- It also used in analog (non-switching) power supplies to convert ac electricity to dc
- Audio envelope followers often use a nonlinear first-order lowpass filter which follows faster going up (capacitor charging) than coming down (capacitor discharging)

Phase Vocoder

In the case of the *Phase Vocoder*, we need to determine both the amplitude $a_k(t)$ and the phase $\phi_k(t)$ of the signal in each subband. Under the assumption of no more than one varying sinusoid in each subband, we can compactly represent the signal in each channel as

$$x_k(t) = a_k(t)\cos[\omega_k t + \phi_k(t)]$$

where ω_k is the fixed channel center frequency. This gives us two real signals for each vocoder channel:

- $a_k(t) =$ *instantaneous amplitude*
- $\phi_k(t) =$ *instantaneous phase modulation*

$a_k(t)$ is also called the *amplitude envelope*.
$\Delta\omega_k(t) = \frac{d}{dt}\phi_k(t)$ is often called the *frequency envelope*.

Note that the amplitude/phase modulation decomposition is *nonlinear* and *not unique*. For example, we could simply set $\phi_k(t) = -\omega_k t$ and $a_k(t) = x_k(t)$. (This would not be interesting.)

Analytic Signal Processing

In order to determine these signals, it is helpful to express the channel signal $x_k(t)$ in its complex "analytic" representation. We will denote this by

$$x_k^a(t) = \mathsf{re}\{x_k^a(t)\} + j \cdot \mathsf{im}\{x_k^a(t)\} \stackrel{\Delta}{=} a_k(t) e^{j[\omega_k t + \phi_k(t)]}$$

Hence,

$$\begin{aligned} a_k(t) &= |x_k^a(t)| \\ \phi_k(t) &= \angle x_k^a(t) - \omega_k t \;=\; \textit{instantaneous phase} \\ &= \tan^{-1}\left[\frac{\mathsf{im}\{x_k^a(t)\}}{\mathsf{re}\{x_k^a(t)\}}\right] - \omega_k t \end{aligned}$$

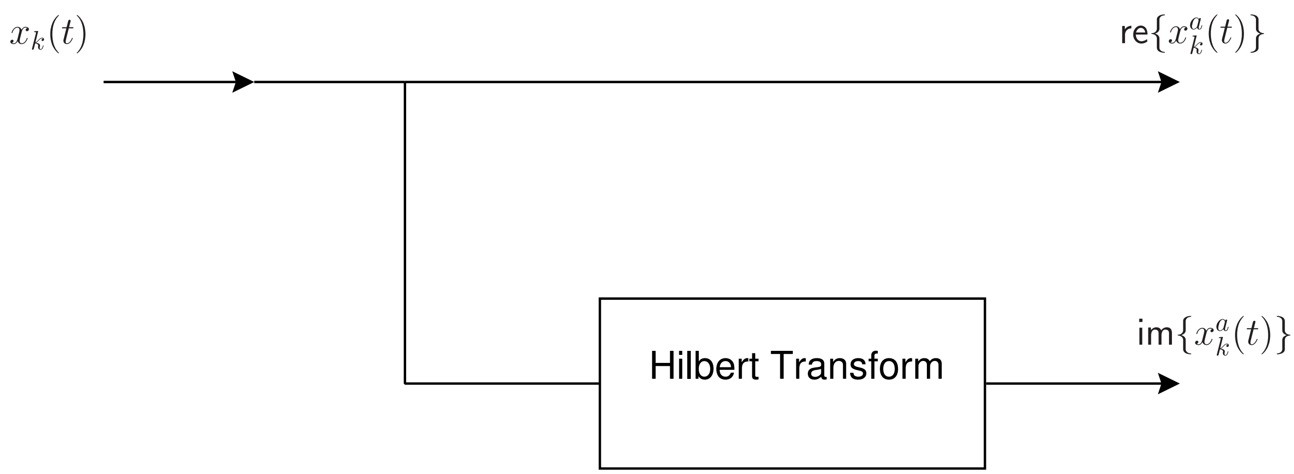
- We normally work in practice with *instantaneous frequency deviation* in place of phase:

$$\Delta\omega_k(t) \stackrel{\Delta}{=} \frac{d}{dt}\phi_k(t)$$

- Since the kth channel of an N-channel uniform filterbank has nominal bandwidth given by f_s/N, the frequency deviation usually does not exceed $\pm f_s/(2N)$ in vocoder analysis

Hilbert Transform

Ideally, the imaginary part of the analytic signal is obtained from its real part using the *Hilbert transform*:



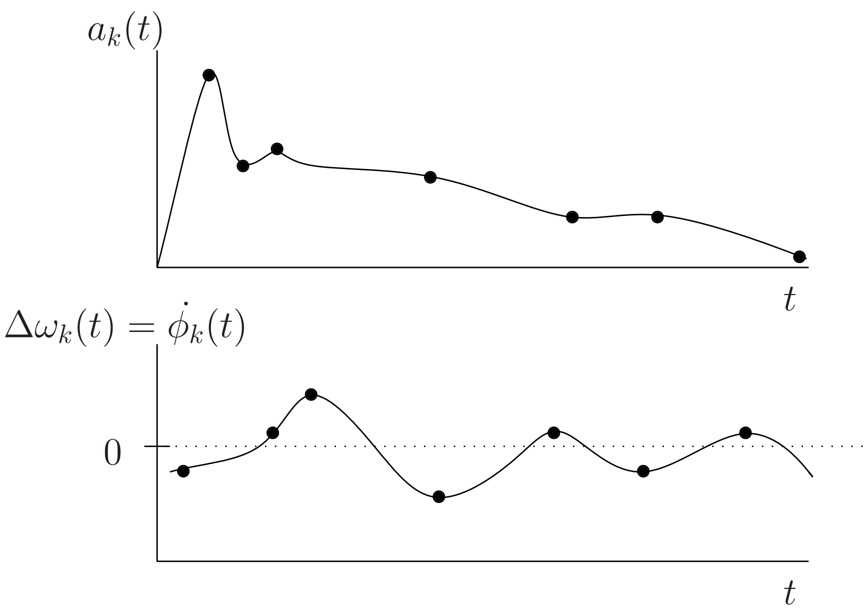
Practical Hilbert transformers may be designed as FIR filters (e.g., `firpm` in the Matlab Signal Processing Toolbox). (See FIR Hilbert-Transform Design in the lecture on the Window Method for FIR Filter Design)

Baseband Processing

Note that $x_k^a(t)$ is a narrowband signal centered about the channel frequency ω_k. It is common to *heterodyne* the channel output signal to "base band" by shifting its spectrum by $-\omega_k$ so as to center the channel bandwidth about zero. This is accomplished by modulating the analytic signal by $\exp(-j\omega_k t)$ to get

$$x_k^m(t) \stackrel{\Delta}{=} e^{-j\omega_k t} x_k^a(t) = a_k(t) e^{j\phi_k(t)}$$

For each of the subbands, we get data which typically looks like the following:



Once we have data in this form, we can compress it using, e.g.,

- Piecewise linear approximation
  - Large compression ratios are possible for “tonal” signals like oboe notes
  - Compression ratio depends on the nature of the signal
- Downsample each channel (MPEG)
  - Each subband is bandlimited to the channel bandwidth
  - Actually, this just gets us back to the original number of samples
    - N channels
    - Downsample by N
- Requantize the signal (MPEG)
  - Allocate bits depending on the amount of energy in each subband

Instantaneous Frequency Computation

Working with the baseband channel signals, we may compute the frequency deviation more easily as simply the derivative of the instantaneous phase:

$$\Delta\omega_k(t) \;\stackrel{\Delta}{=}\; \frac{d}{dt}\angle x_k^m(t) \;=\; \dot{\phi}_k(t)$$

Let, $x \stackrel{\Delta}{=} \text{re}\{x_k^m(t)\}$ and $y \stackrel{\Delta}{=} \text{im}\{x_k^m(t)\}$. Then we have

$$\begin{aligned}\dot{\phi}_k(t) \;&=\; \frac{d}{dt}\tan^{-1}\left(\frac{y}{x}\right) \;=\; \frac{\frac{d}{dt}(y/x)}{1+(y/x)^2} \\ &=\; \frac{x^2[\dot{y}/x - y\dot{x}/x^2]}{x^2+y^2} \;=\; \frac{x\dot{y}-y\dot{x}}{x^2+y^2}\end{aligned}$$

Vocoder Demos, 26 Channels

1. Original[2]
2. Resynthesis[3] preserving amplitude envelopes but discarding frequency deviations
3. $= 2$ with Channel Frequency-Inversion.[4]
   That is, the vocoder channels are reversed in frequency order, which obscures the formants.
4. Noise Substitution[5]
   Each original channel amplitude-envelope is applied to a narrowband noise with bandwidth equal to that of the analysis channel (instead of a sinusoid).
5. Noise Substitution and Frequency Inversion[6]

Parameters:

- $f_s = 8$ kHz sampling rate
- 26 vocoder channels, auditory spaced

[2] http://ccrma.stanford.edu/~jos/wav/SteveJobs.wav
[3] http://ccrma.stanford.edu/~jos/wav/SteveJobs_sine_n_26.wav
[4] http://ccrma.stanford.edu/~jos/wav/SteveJobs_sine_i_26.wav
[5] http://ccrma.stanford.edu/~jos/wav/SteveJobs_noise_n_26.wav
[6] http://ccrma.stanford.edu/~jos/wav/SteveJobs_noise_i_26.wav

Vocoder Demos, 5 Channels

1. Original[7]
2. Resynthesis[8] preserving amplitude envelopes but discarding frequency deviations
3. = 2 with Channel Frequency-Inversion.[9]
   That is, the vocoder channels are reversed in frequency order, which obscures the formants.
4. Noise Substitution[10]
   Each original channel amplitude-envelope is applied to a narrowband noise with bandwidth equal to that of the analysis channel (instead of a sinusoid).
5. Noise Substitution and Frequency Inversion[11]

Parameters:

- $f_s = 8$ kHz sampling rate
- 5 vocoder channels
- Center frequencies at 148 Hz, 392 Hz, 825 Hz, 1.6 kHz, and 3 kHz

[7] http://ccrma.stanford.edu/~jos/wav/SteveJobs.wav
[8] http://ccrma.stanford.edu/~jos/wav/SteveJobs_sine_n_5.wav
[9] http://ccrma.stanford.edu/~jos/wav/SteveJobs_sine_i_5.wav
[10] http://ccrma.stanford.edu/~jos/wav/SteveJobs_noise_n_5.wav
[11] http://ccrma.stanford.edu/~jos/wav/SteveJobs_noise_i_5.wav

Vocoder Limitations

There are some inherent problems with the vocoder:

- We required a maximum of one quasi-sinusoid per subband
  - This means we need lots of filters
- Poor model for signal transient or sharp attack
- Inconvenient for inharmonic signals
- Inefficient model for signals with *noise*-like qualities (e.g., flute)
- Not an *identity* system
  (unless *phase* retained and no data reduction done)
- Computationally expensive

Tracking Sinusoidal Peaks in a Sequence of FFTs

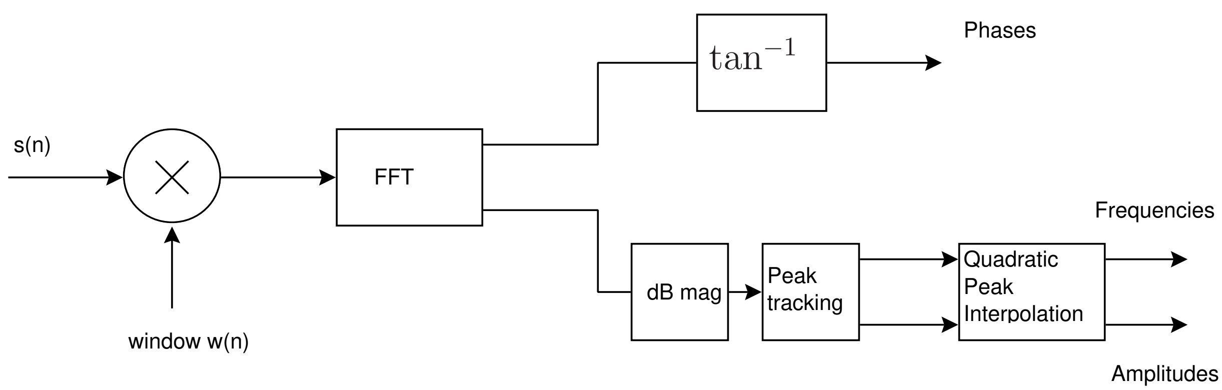
This technique addresses some of the problems inherent in vocoders.

Important points:

- Applicable to inharmonic sounds (e.g., piano)
- Analysis only near spectral peaks, not in every filter band
- "Non-coherent" sinusoidal parameter estimation from magnitude spectrum (peak amplitude, center-frequency, and sometimes phase)
- Quadratic interpolation and zero-padding may be used to accurately find spectral magnitude peaks
- The Short Time (fast) Fourier Transform (STFT) is used for analysis
  - STFT can be interpreted as a filterbank (more on this later)
  - FFT makes it computationally feasible to implement filter banks with a large number of analysis filters

- Resynthesis using oscillator bank or IFFT
- Original signal is replaced by oscillator amplitude and frequency *envelopes*
- When a signal is converted entirely to envelopes, *time-scale modification* and *frequency scaling* become easy (simply resample the envelopes)

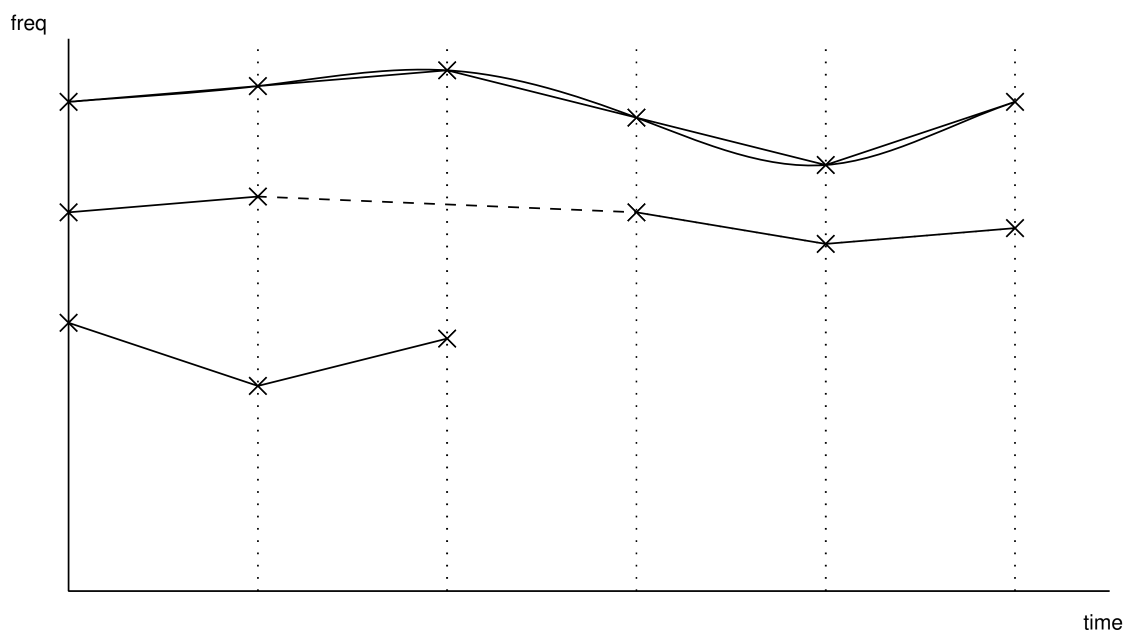
The following diagram depicts the general analysis system:



- For steady-state signals, phase is usually discarded
- Phase is normally needed for frames containing a *transient*, or to provide a phase-locked transition to a transient frame

Peak Tracking across Frames

- Sinusoidal peaks must be *associated* across frames
- *Linear interpolation* may be used to define the instantaneous amplitude and frequency *between* frames, when phase is discarded (`PARSHL`[12]).
- When phase is retained, cubic phase interpolation can be used from frame to frame (McAulay and Quatieri).



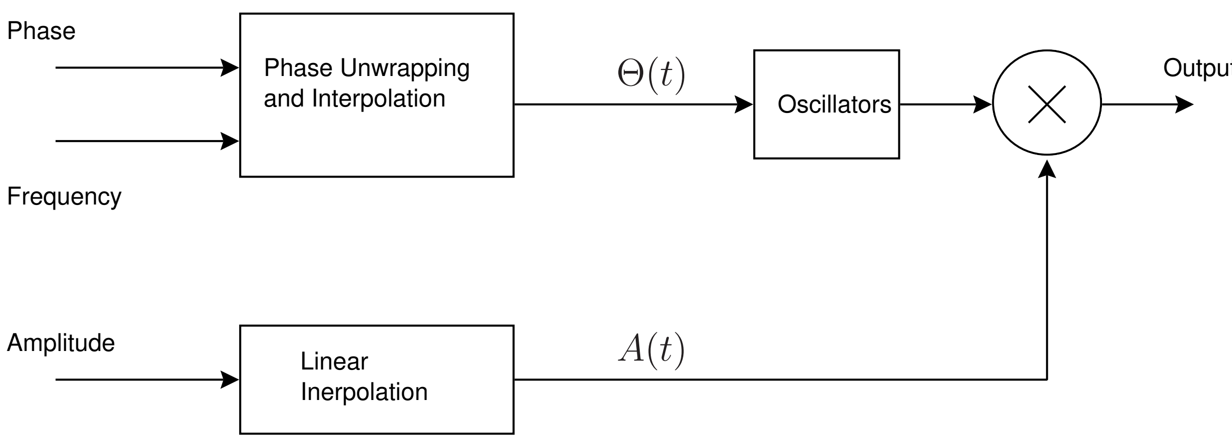
[12] `http://ccrma.stanford.edu/~jos/parshl/`

A *transient detector* on the side can be used to indicate when the peak phases should be retained:

- Differentiated amplitude envelope of high-passed time-domain signal
- Linear prediction error

See Scott Levine CCRMA thesis[13]

Synthesis is performed using a bank of amplitude- and phase-modulated oscillators:

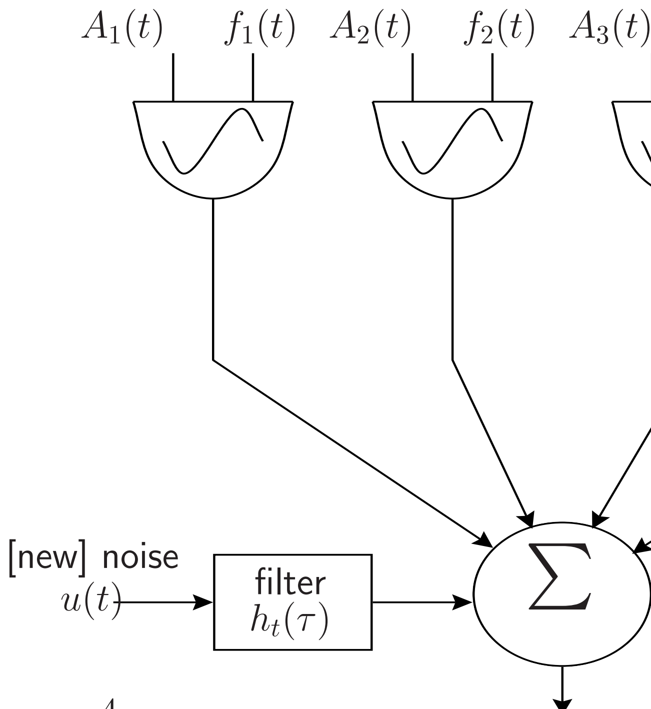


(Phase-Preserving Case)

[13] http://ccrma.stanford.edu/~scottl/thesis.html

Sines+Noise Modeling

Sines+Noise Synthesis (S+N) generalizes the sinusoidal signal models to include a *filtered noise component*:



$$y(t) = \sum_{i=1}^{4} A_i(t) \cos\left[\int_0^t \omega_i(t)dt + \phi_i(0)\right] + (h_t * u)(t)$$

where

- $u(t) =$ white noise
- $h_t(\cdot) =$ slowly changing noise filter

Sines + Noise Sound Examples

Xavier Serra 1989 thesis demos (Sines + Noise signal modeling)

- Guitar
  - Original
  - Sinusoids alone
  - Residual after sinusoids removed
  - Sines + noise model
- Piano
  - Original
  - Sinusoids alone
  - Residual after sinusoids removed
  - Sines + noise model
- Voice
  - Original
  - Sinusoids
  - Residual
  - Synthesis
  - Original, Sinusoids, Residual, Synthesis

Musical Effects with Sines+Noise Models

- Piano Effects
  - Pitch downshift one octave
  - Pitch flattened
  - Varying partial stretching
- Voice Effects
  - Frequency-scale by 0.6
  - Frequency-scale by 0.4 and stretch partials
  - Variable time-scaling, deterministic to stochastic

Cross-Synthesis with Sines+Noise Models

- Voice “modulator”
- Creaking ship’s mast “carrier”
- Voice-modulated creaking mast
- Same with modified spectral envelopes

Sines + Transients Sound Examples

In this technique, the sinusoidal sum is phase-matched at the cross-over point only (with no cross-fade).

- Marimba
  - Original
  - Sinusoidal model
  - Original attack, followed by sinusoidal model
- Piano
  - Original
  - Sinusoidal model
  - Original attack, followed by sinusoidal model

Notes

- Only one voice analyzed at a time
- Analyzed sounds were generally tonal (having a distinct pitch)
- FFT analysis resolution was fixed by window length (a few periods)
- No transient model (but sinusoids could start with correct initial phase)

Sines+Noise+Transients

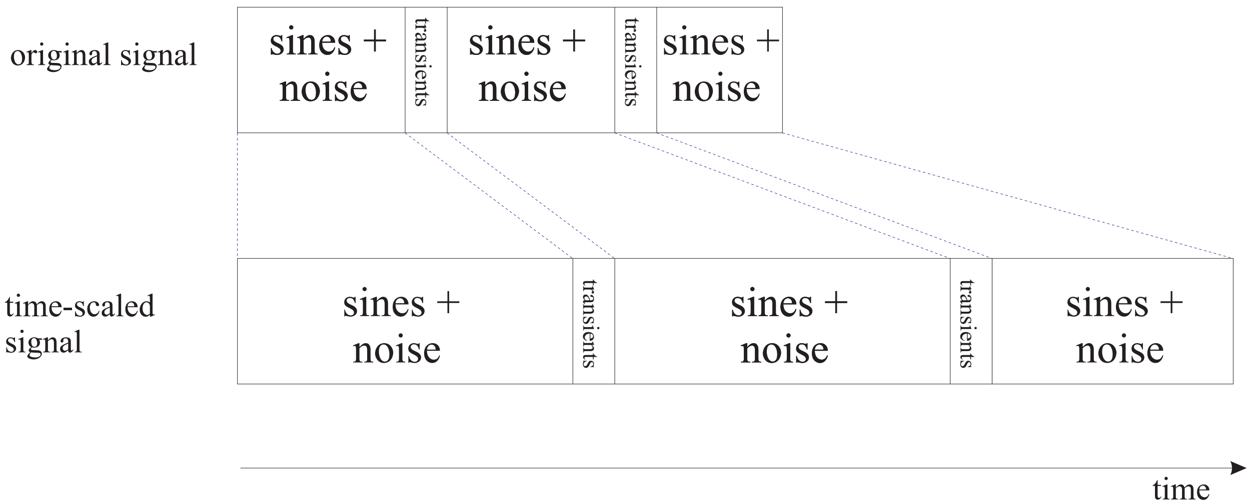
Why Model Transients Separately?

- Sinusoids efficiently model spectral *peaks* over time
- Filtered noise efficiently models spectral *residual* vs. t
- Neither is good for *abrupt transients* in waveform
- Need to switch to a *transient model* during transients
- Need sinusoidal *phase matching* at the switching time

Transient models

- Original waveform frame
- Wavelet expansion
- MPEG-2 AAC (with short window)
- Frequency-domain LPC
  (time-domain amplitude envelope)

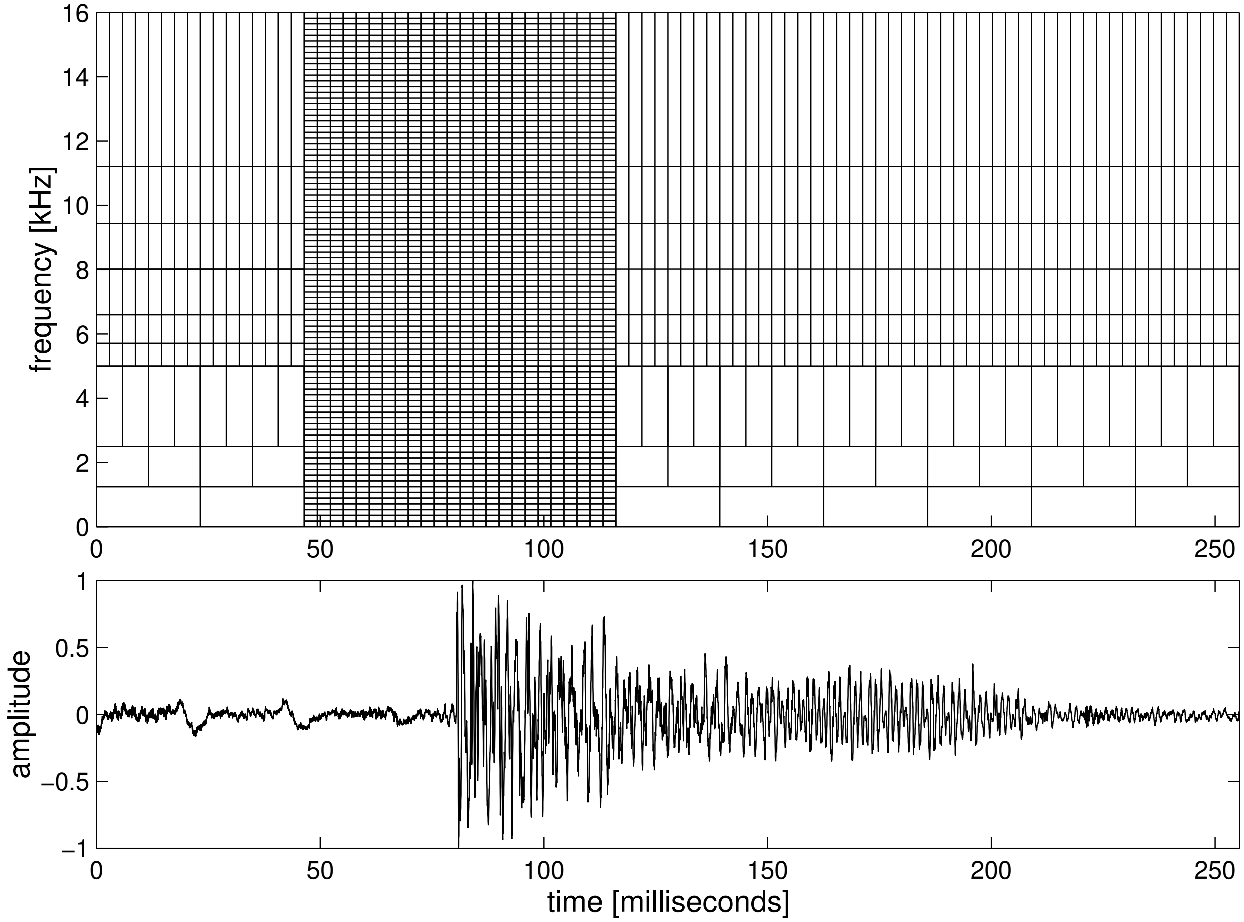
Time Scaling for Sines+Noise+Transients Models



(From Scott Levine's Thesis)

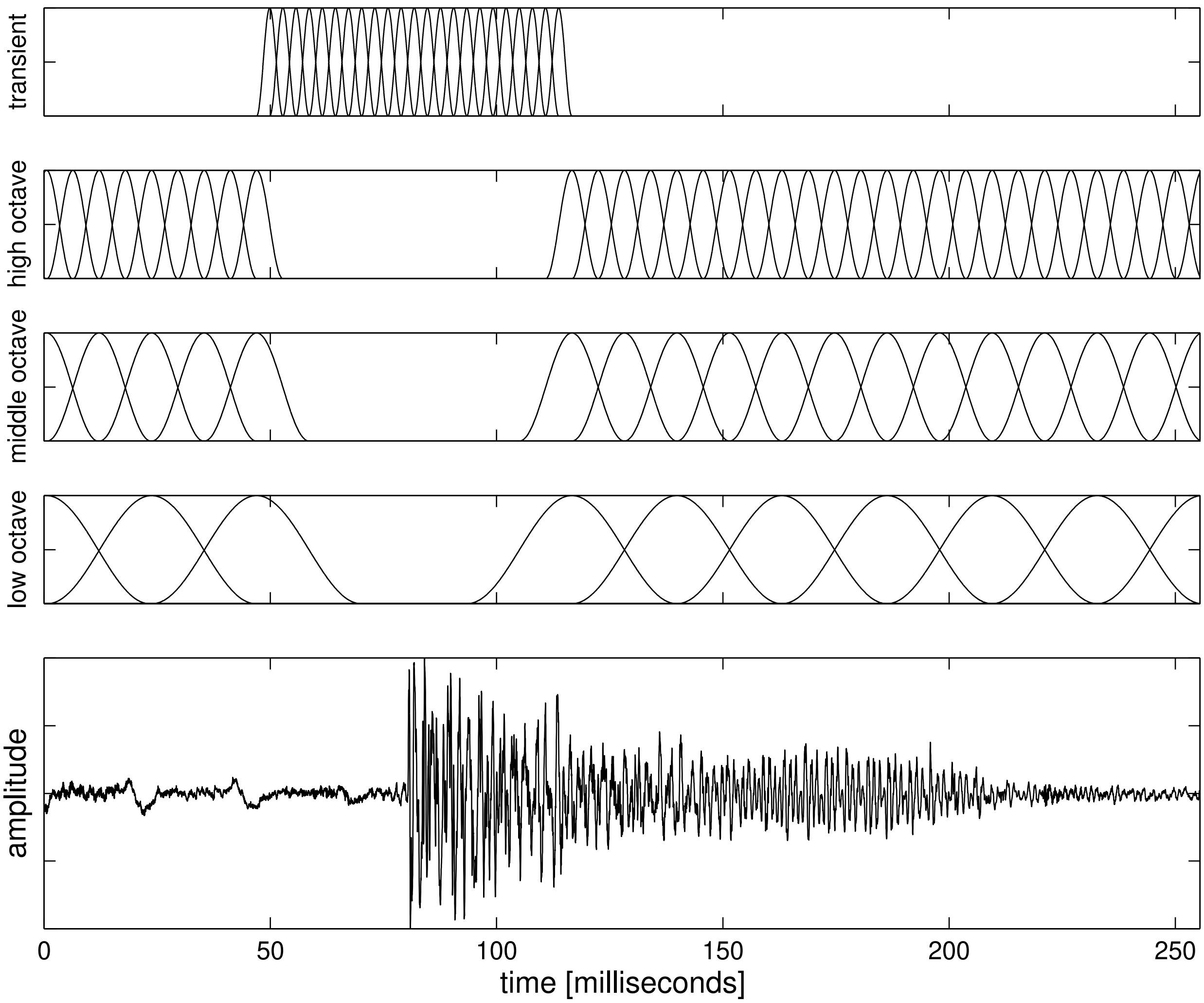
- In sines+noise models, transients are "smeared" over time
- In sines+noise+transients models, they are only time-shifted
- Missed transients can cause artifacts in S+N+T models
- Need to consider carefully what should be defined as a transient
- Hybrid schemes possible (transients stretch some)

Sines + Noise + Transients
Time-Frequency Map



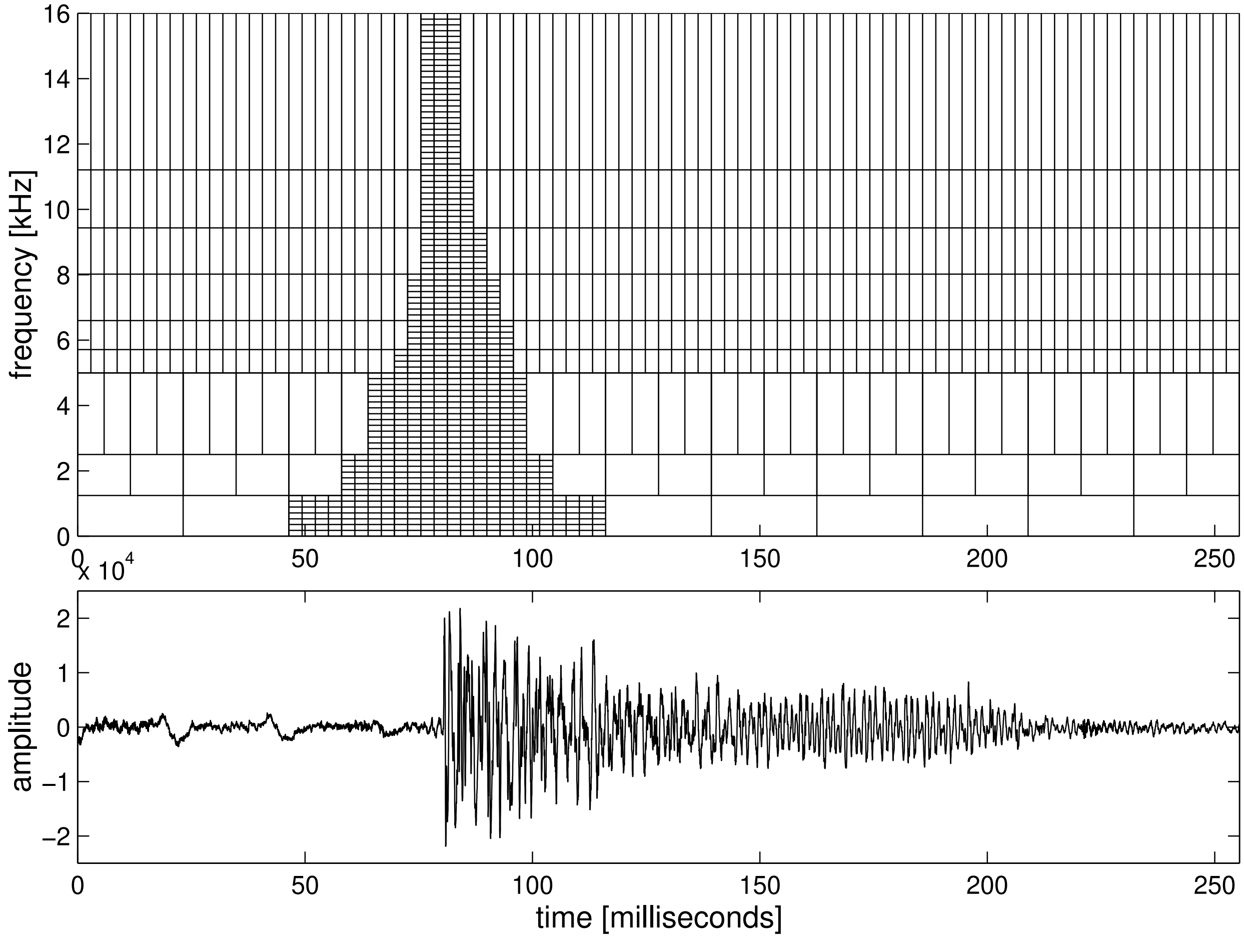
(From Scott Levine's Thesis)

Corresponding Analysis Windows



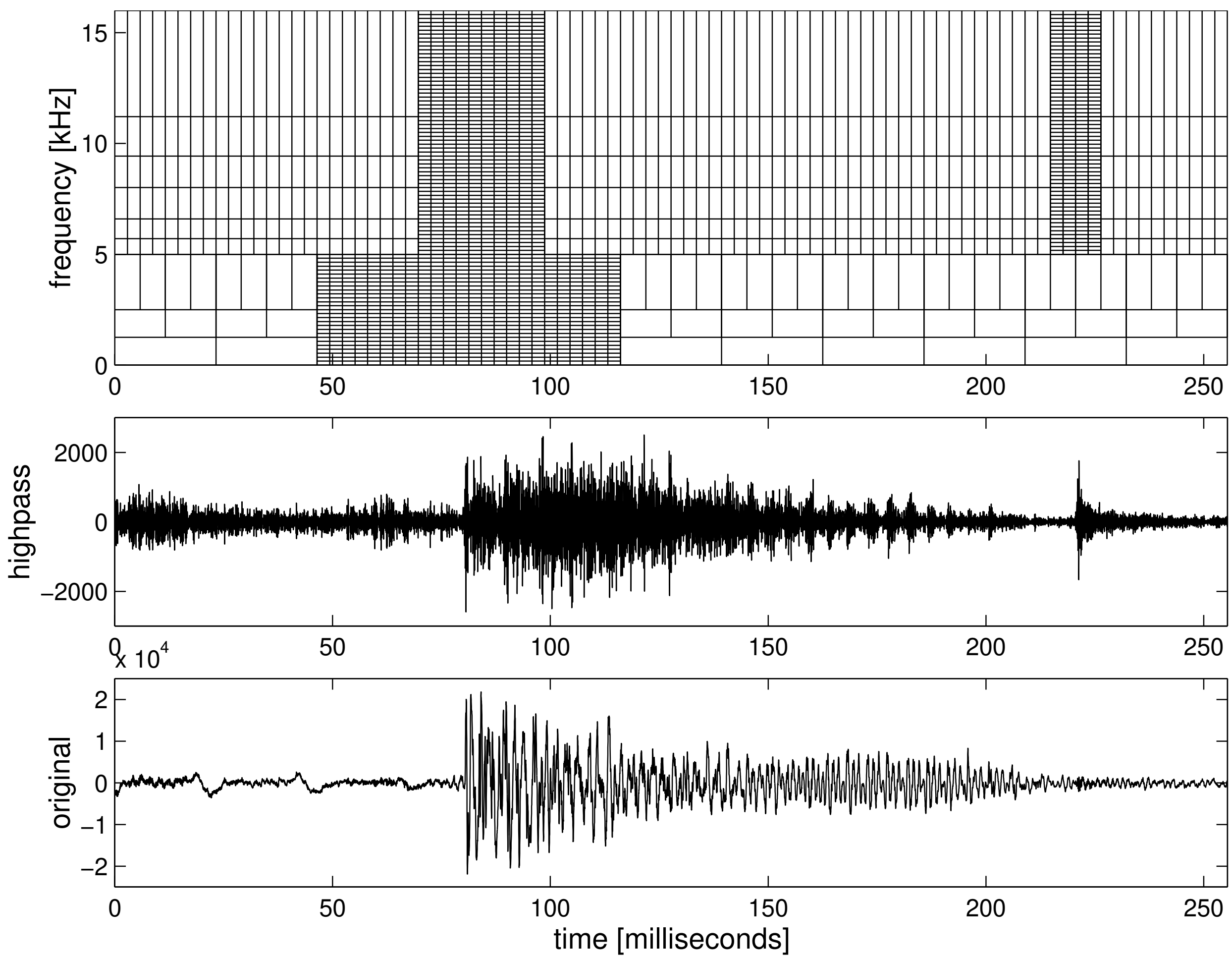
(From Scott Levine's Thesis)

Quasi-Constant-Q (Wavelet) Time-Frequency Map



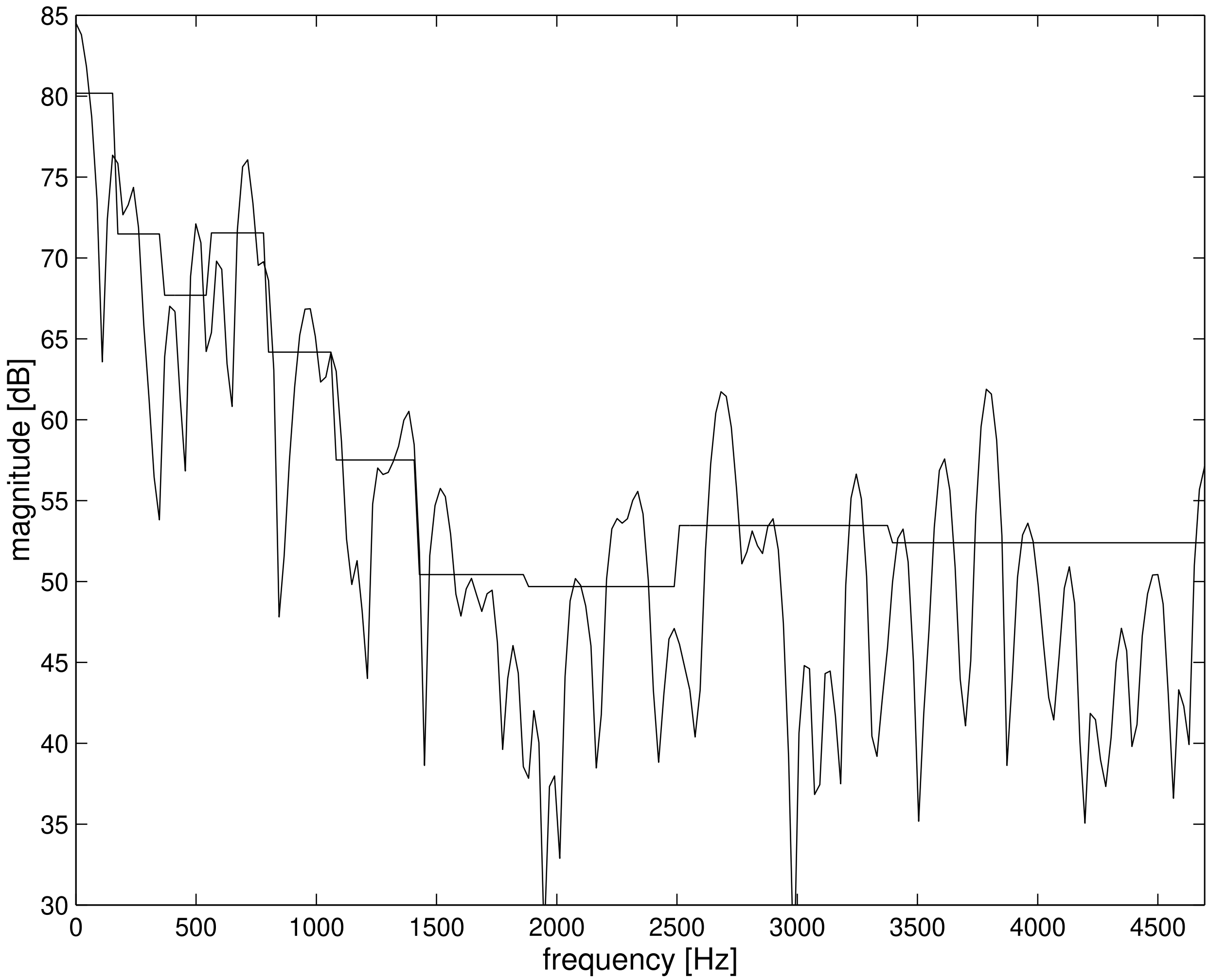
(From Scott Levine's Thesis)

Micro-Transients



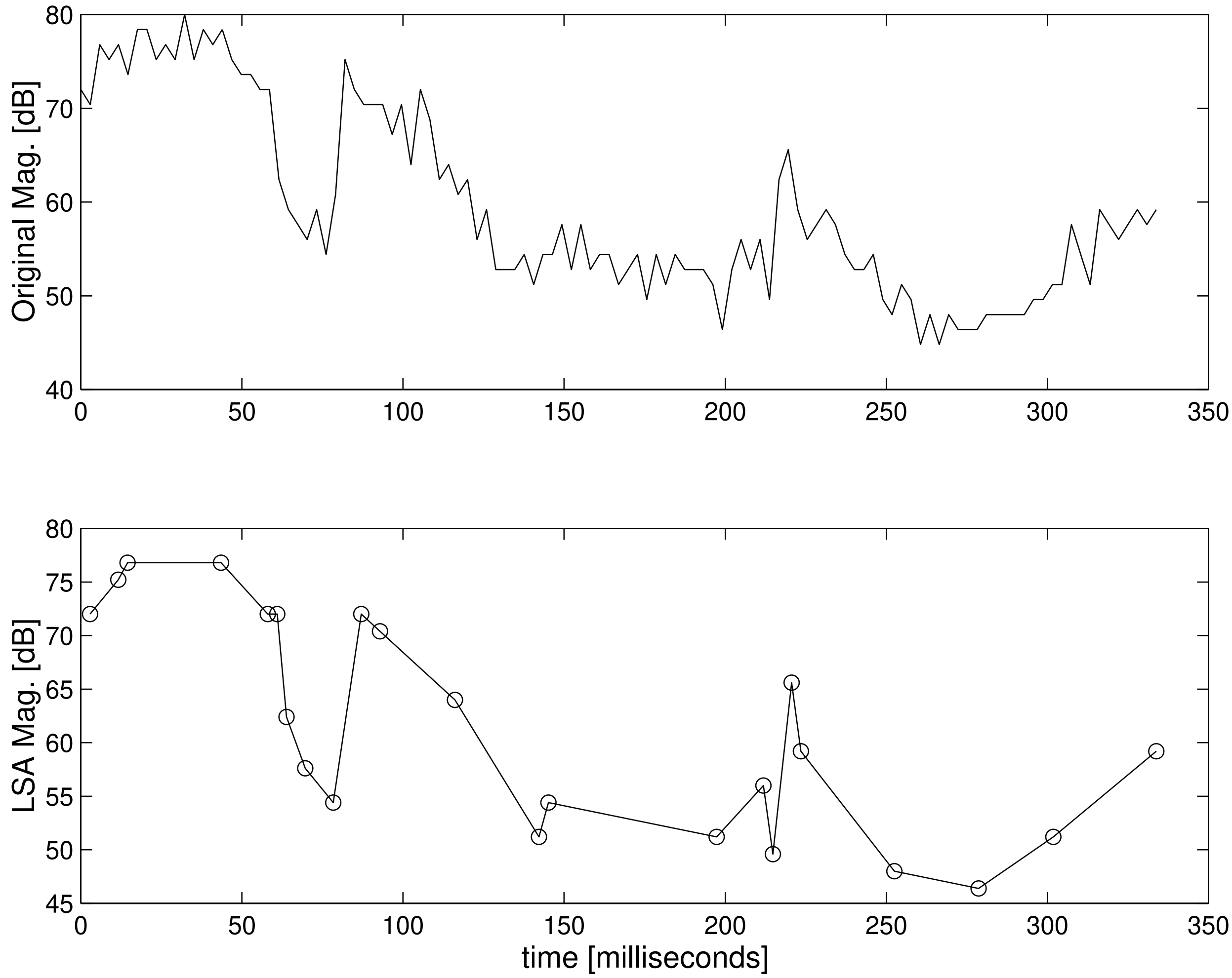
(From Scott Levine's Thesis)

Bark-Band Noise Modeling at High Frequencies



(From Scott Levine's Thesis)

Amplitude Envelope for One Noise Band



(From Scott Levine's Thesis)

For more information, see Scott Levine's thesis.[14]

[14] http://ccrma.stanford.edu/~scottl/thesis.html

Sines + Noise + Transients Sound Examples

Scott Levine Thesis Demos
(`http://ccrma.stanford.edu/~scottl/thesis.html`)

Sines + Noise + Transients at 32 kbps

Mozart's Le Nozze di Figaro

- Original
- Compressed using MPEG-AAC at 32 kbps
- Compressed using sines+transients+noise at 32 kbps

- Multiresolution sinusoids alone
- Residual Bark-band noise
- Transform-coded transients (AAC)
- Bark-band noise above 5 kHz

"It Takes Two" by Rob Base & DJ E-Z Rock

- Original
- MPEG-AAC at 32 kbps
- Sines+transients+noise at 32 kbps

- Multiresolution sinusoids
- Residual Bark-band noise
- Transform-coded transients (AAC)
- Bark-band noise above 5 kHz

Time-Scale Modification

(pitch unchanged)

- Time-scale factors [2.0, 1.6, 1.2, 1.0, 0.8, 0.6, 0.5]

Pitch Scaling (timing unchanged)

- Pitch-scale factors [0.89, 0.94, 1.00, 1.06, 1.12]