Universität Stuttgart
Fakultät Informatik

Prüfer:	Prof. Dr. Rul Gunzenhäuser
Betreuer:	Xavier Rodet
begonnen am	15. Dezember 1997
beendet am	16. Juni 1998
CR-Klassifikation:	J.5.5, I.5.4, I.2.7

Diplomarbeit Nr. 1622

Spectral Envelopes in Sound Analysis and Synthesis

Diemo Schwarz

IRCAM
Institut de la Recherche et Coordination Acoustique/Musique
1, place Igor–Stravinsky
F-75004 Paris

Institut für Informatik
Breitwiesenstraße 20-22
D-70565 Stuttgart

Abstract

In this project, *Spectral Envelopes in Sound Analysis and Synthesis*, various methods for estimation, representation, file storage, manipulation, and application of spectral envelopes to sound synthesis were evaluated, improved, and implemented. A prototyping and testing environment was developed, and a function library to handle spectral envelopes was designed and implemented.

For the **estimation** of spectral envelopes, after defining the requirements, the methods *LPC*, *cepstrum*, and *discrete cepstrum* were examined, and also improvements of the discrete cepstrum method (*regularization*, *stochastic (or probabilistic) smoothing*, *logarithmic frequency scaling*, and *adding control points*). An evaluation with a large corpus of sound data showed the feasibility of discrete cepstrum spectral envelope estimation.

After defining the requirements for the **representation** of spectral envelopes, *filter coefficients*, *spectral representation*, *break-point functions*, *splines*, *formant representation*, and *high resolution matching pursuit* were examined. A combined spectral representation with indication of the regions of formants (called *fuzzy formants*) was defined to allow for integration of spectral envelopes with precise formant descriptions. For **file storage**, new data types were defined for the SDIF *Sound Description Interchange Format* standard.

Methods for **manipulation** were examined, especially **interpolation** between spectral envelopes, and between spectral envelopes and formants, and other manipulations, based on primitive operations on spectral envelopes. For sound **synthesis**, application of spectral envelopes to *additive synthesis*, and *time-domain or frequency-domain filtering* have been examined.

For prototyping and testing of the algorithms, a spectral envelope **viewing program** was developed. Finally, the **spectral envelope library**, offering complete functionality of spectral envelope handling, was developed according to the principles of software engineering.

Contents

List of Figures

Chapter 1

Introduction

For many sound synthesis and processing applications, the spectral envelope plays a crucial role. For musical applications, the spectral envelope largely determines the timbre, or "colour" of a sound. For the voice, important characteristics such as the type of a vowel or part of the vocal quality of a voice depend on the spectral envelope. In general, the spectral envelope is a smoothed version of the frequency spectrum of a sound, and is often independent of the pitch.

1.1 Motivation

In the context of computer music, it is important to have precise control of the spectral envelope of a sound e.g. for composers to realize their ideas or for the synthesis of the singing voice to localize formants precisely and thus to obtain good voice quality.

In the context of additive analysis and synthesis, where a sound is decomposed into sinusoids, rendering each minute detail of the sound accessible to manipulation, spectral envelopes can lead a way out of the dilemma of how to control the parameters of the hundreds of sinusoids over time in a sensible way.

1.2 Outline of the Project

The objective of the project is to develop methods of spectral envelope estimation, manipulation, and application for sound synthesis, and to define a flexible and efficient representation of the data, both for internal and for permanent storage. The methods found are to be implemented in a portable function library.

The project takes place at the IRCAM (*Institut de la Recherche et Coordination Acoustique/Musique*) in Paris, France, a national institute of research in music and acoustics, musical creation and production, and education, putting its main emphasis on the use of computers in contemporary music.

1.3 Outline of this Documentation

The documentation of the spectral envelope project can be roughly divided into four parts.

The first part, fundamentals, consists of chapters 1 and 2, and covers this introduction and the prerequisites of the project. In chapter 2, the basics of digital signal processing will be briefly visited, followed by an introduction to additive sound synthesis, the concept of spectral envelopes and the source–filter model of speech production. Finally, a quick tour of the other programs and systems of IRCAM related to spectral envelopes will set the framework for their application.

In the second part, the methods used and developed in this project will be described formally. After a description of the estimation of spectral envelopes in chapter 3, the internal representation is presented in chapter 4, manipulation of spectral envelopes is covered in chapter 5, and the application of spectral envelopes to sound synthesis can be found in chapter 6.

The third part, implementation, describes the translation of the results of the previous chapters into a working software system. Chapter 7 first presents some general considerations for the implementation, such as a brief introduction to software engineering, the architectural notation used. Then, the architecture of the spectral envelope library is presented, followed by a definition of the file format used. Appendix A describes the usage and the structure of the program ViewEnv to experiment with spectral envelope algorithms.

Finally, an overview of existing and possible applications of spectral envelopes is given in chapter 8, and a summary of the project and a conclusion from a scientific and an artistic viewpoint is given in chapter 9.

1.4 Acknowledgements

I would like to thank the following people, without whom this project would have been much harder to accomplish, or not possible at all:

Xavier Rodet for accepting me into the analysis/synthesis team, for the interesting subject and countless suggestions, Prof. Dr. Rul Gunzenhäuser for agreeing to examine the work, Prof. Dr. Klaus Lagally for his unbureaucratic approach and encouraging of the endeavor, Katharina Mehner for more understanding and support than can be asked for and for her critical revision and comments of the report, Margret Schwarz and Rudolf Schwarz for laying the basis for all this to be possible, Andrew Brown for a steady dose of philosophy and his incredible accuracy pinning flawed thoughts, Marcelo Wanderley for the fruitful discussions and structured comments, Marine Campedel for the cheerful sharing of her knowledge, Geoffroy Peeters for *"un p'tit café?"* and the music, Christoph Vergez for his good humor and the practical tips, Rémi Gribonval for ideas on wavelets, Adrien Lefèvre for the nice environment, Natalie Henrich for being my first fearless user and perseverant β-tester, Venice for Stefania, and all the other people of the analysis/synthesis team for the stimulating and pleasant stay.

Chapter 2

Basic Concepts

This chapter will describe some basic concepts necessary to understand the spectral envelopes project. After a short introduction to some concepts in digital signal processing which permeate the whole project in section 2.1, the basic ideas of additive sound analysis and synthesis are explained in section 2.2, followed by a closer examination of the concept of spectral envelopes. Finally, to embed the project in the ongoing work at IRCAM, the programs used here, and the programs and systems where spectral envelope manipulation will be used will be visited briefly in section 2.5.

2.1 Digital Signal Processing

This section will give a brief introduction to, and a formal definition of, the basic concepts and methods of the theory of digital signal processing used in this project. They form the basis for the methods presented in the subsequent chapters. Necessarily, this introduction will be very brief and restricted. In particular, I will consider the theory of digital signal processing apart from the more general theory of analog signal processing, and will pass over the various problems and mathematical prerequisites one has to take care of (stability, convergence). See [RH91] for a broad introduction, [Rob98] for an online tutorial, and [OS75] for a more profound presentation.

2.1.1 Sampling

The audio data we wish to treat will generally be present in the form of electric oscillations. These can either come from a microphone recording acoustic sound waves, from a tape, or an electronic instrument or device. To convert these oscillations to a form that can be treated by a computer they have to be digitised, i.e. reduced from an analog form (time- and value-continuous) to digital (time- and value-discrete). This process is called **A/D conversion** or **sampling**. It consists of two steps:

1. First, the analog signal $s(t)$ is converted to a time-discrete form $x(n)$ by sampling its value (in the more specific sense of the word) in periodical intervals of duration t_s, the **sampling period**, (see figure 2.1). The **sampling rate** is then defined as:

   $$f_s = \frac{1}{t_s} \tag{2.1}$$

   The index n of the sampled signal $x(n)$ is related to time by $t = nt_s$ such that

   $$x(n) = s(nt_s) \tag{2.2}$$

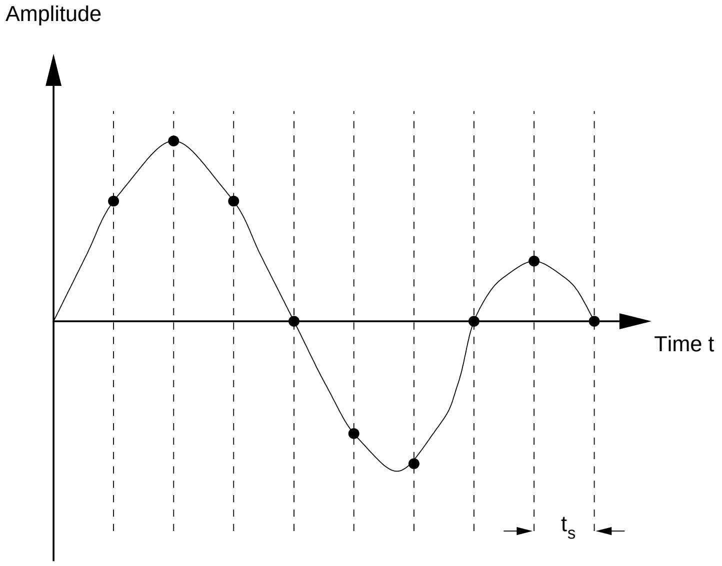

Figure 2.1: Conversion of an analog to a time-discrete signal (*sampling*)

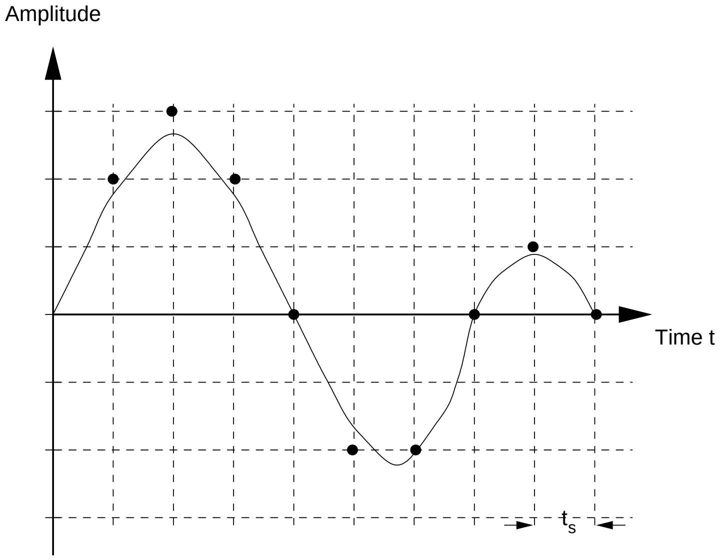

Figure 2.2: Conversion of a time-discrete to a digital signal (*quantization*)

2. Then, the value-continuous samples are rounded to yield a value-discrete binary number, commonly called a **sample**. For most commercial audio applications, a 16 bit integer format is used, for research, however, a normalized 32 bit floating point format with values ranging from -1.0 to 1.0 is preferred. This step 2 is called **quantization** (see figure 2.2).

The result of A/D conversion is shown in figure 2.3. It is a curve with discrete steps at every sample. Note, however, that for most of the theory of digital signal processing, as for the rest of this chapter, the samples are generally considered to be value-continuous.

There are various sources of errors in the process of A/D conversion, including jitter, quantization noise, and aliasing:

Jitter

Deviations of the periodicity of the sampling of the analog signal (step 1 above) are called jitter. Jitter is a technical problem that is solved with current technology.

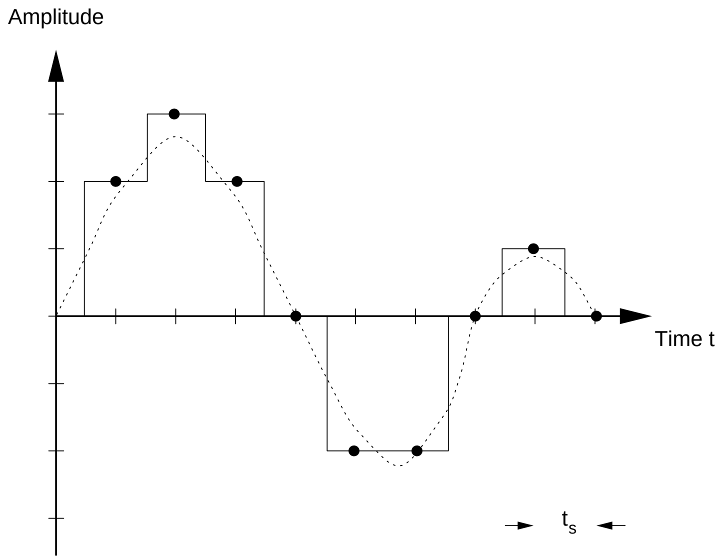

Figure 2.3: Result of A/D conversion

Quantization noise
Because of the rounding errors in the process of quantization (step 2 above), quantization noise is introduced into the sampled signal. The rounding errors are dependent of the number of quantization intervals, i.e. the number of values D one sample can express. Because a sample is stored as a binary number, D has the form 2^n, with n being the number of bits (the word length) of the binary number. As a rough estimate, the ratio between the signal and the quantization noise increases by 6 dB per bit added (see section 2.1.2 for a definition of the unit dB).

Aliasing
According to the sampling theorem of Nyquist, the sampling frequency f_s must be at least twice the highest frequency occurring in the signal.[1] If not, the parts of the signal above $f_s/2$ are introduced as low frequency aliasing noise in the sampled signal. To avoid this, an *anti-aliasing filter* is inserted before the A/D converter to suppress frequencies above half of the sampling frequency.

Converting a digital signal back to an analog signal is called **D/A conversion**. Roughly, it consists of reading the samples and generating an electric oscillation accordingly. As this will look like the step-curve in figure 2.3, it has to be smoothed by a low-pass filter (a filter that lets only the frequencies below his cutoff-frequency pass) to yield a signal suitable to being amplified and played via loudspeakers, or recorded on analog tape.

2.1.2 Power and Energy

The notion of energy and power for digital signals is only slightly related to the physical units pertaining to analog electrical signals. They are, nevertheless, related to the perceived loudness, albeit in a non-trivial way.[2]

The **power** of a signal $x(n)$ is defined by

$$p_x(n) = |x(n)|^2 \tag{2.3}$$

[1] To be precise, it is enough that the width of the frequency band the signal uses (the bandwidth) be less than the Nyquist frequency. This is exploited in digital telephony, where the transmitted signal is restricted to between c.a. 300 Hz and 4300 Hz, thus a sampling rate of 8000 Hz is sufficient.

[2] To discover the relationship between physically measurable features of sound and the human perception of sound is the aim of psychoacoustics, see [Zwi82].

Then its **energy** is

$$E_x = \sum_{n=-\infty}^{\infty} p_x(n). \tag{2.4}$$

Taken as such, this value doesn't tell us much. It is useful, however, to compare two signals. For this end, the unit **decibel** (dB) is introduced: The ratio R in dB of two signals with energies E_1 and E_2 is

$$R = 10 \cdot \log_{10} \frac{E_1}{E_2} \tag{2.5}$$

Often, however, an implicit reference energy of $E_2 = 1$ is used. A ratio of 10 dB corresponds roughly to doubling the loudness. The unit decibel is well suited to practical use, since the dynamic range of the human ear is quite large, still, the values in decibel stay in a manageable range. For example, if the threshold of hearing is defined as a reference point of 0 dB, the dynamic range reaches up to 120 dB, the pain threshold. This corresponds to doubling the loudness 12 times.

2.1.3 Fourier Transform

After A/D conversion, the signal $x(n)$ is represented in the time domain as a sequence of numbers. According to Fourier's ingenious idea, any function, even nonperiodic ones, can be expressed as a sum of (periodic) sinusoids (which can not be further decomposed, i.e. they are "pure"[3]). In the complex domain, a sinusoid can be expressed by

$$e^{j\omega n} = \cos \omega n + j \sin \omega n \tag{2.6}$$

with the **angular frequency** ω given by

$$\omega = \frac{2\pi}{N} k \tag{2.7}$$

for some discrete frequency k and N frequency bands, or, related to frequencies in Hertz (Hz), as

$$\omega = \frac{2\pi}{f_s} f \tag{2.8}$$

So, to reveal the frequency structure of x, it can be converted to a frequency domain representation X by the **discrete Fourier transform** (DFT):

$$X(k) = \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}nk} x(n) \tag{2.9}$$

X is called the **frequency spectrum** of the digital signal x of length N. To go the other way, we use the **inverse discrete Fourier transform**:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} e^{-j\frac{2\pi}{N}nk} X(k) \tag{2.10}$$

The result of the Fourier transform in (2.9) is N complex numbers $X(k)$ which define the **magnitude spectrum** $M(k)$ and **phase spectrum** $\phi(k)$ for a discrete frequency k:

$$M(k) = |X(k)| \tag{2.11}$$

$$\phi(k) = \angle X(k) = \arctan \left| \frac{\mathbf{re}\ X(k)}{\mathbf{im}\ X(k)} \right| \tag{2.12}$$

[3] In mathematical terms, this "purity" means the sinusoids of different frequencies form an orthogonal basis of a vector space, the Fourier space. The time-domain is in fact another vector space, with the dirac functions $\delta(n - n_0)$ for $n_0 \in \mathbf{Z}$ and the Fourier transform is simply a change of basis

The magnitude spectrum gives the intensity of a sinusoid at frequency k, while the phase spectrum gives its shift in time. Often, the phase spectrum is ignored (it is not important for speech recognition), and the term Fourier spectrum or spectrum is used for the magnitude spectrum.

There exists a generalization of the discrete Fourier transform[4], called the **Z–transform**, which often simplifies the representations in the frequency domain. The Z–transform $X(z)$ of a discrete sequence $x(n)$, where z is a complex variable, is defined as:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n} \tag{2.13}$$

If z is substituted by $e^{j\omega k}$, the relationship to the Fourier transform is obvious. See [OS75] for a rigorous presentation of the Z–transform, or [Mel97] for an on-line tutorial.

The computational complexity of the DFT, when computed directly, is $O(N^2)$. By taking advantage of the periodicity of the analysing sinusoid $e^{-j\frac{2\pi n}{N}k}$ and applying the **divide and conquer** principle of splitting the problem into successively smaller subproblems, a variety of more efficient algorithms of complexity $O(N \log N)$ have been developed which came to be known collectively as the **fast Fourier transform** (FFT). They generally require that N be a power of two. As the discrete Fourier transform and its inverse differ only in the sign of the exponent and a scaling factor, the same principles lead to the **inverse fast Fourier transform** (IFFT).

2.1.4 Convolution, Filtering and Linear Systems

The fundamental operation of digital signal processing is the **convolution** of two signals $x(n)$ and $h(n)$ to yield $y(n)$, defined by

$$y(n) = x(n) * h(n) := \sum_{k=-\infty}^{\infty} x(k) h(n-k) \tag{2.14}$$

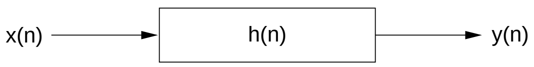

Figure 2.4: Convolution of a discrete signal $x(n)$ with a signal $h(n)$.

This is what is behind the notion of **filtering**: The output signal y is x filtered by h. It is usually expressed graphically as shown in figure 2.4. This operation is commutative and associative:

$$x(n) * h(n) = h(n) * x(n) \tag{2.15}$$
$$h_1(n) * (h_2(n) * x(n)) = h_2(n) * (h_1(n) * x(n)) = (h_1(n) * h_2(n)) * x(n) \tag{2.16}$$

Systems which perform linear operations such as convolution, addition, and multiplication by a constant are part of the class of **linear time-invariant systems** (LTI systems).[5] They are notated as in figure 2.5.

Figure 2.5: A linear time-invariant system performing operation T.

The properties of **linearity** and **time-invariance** are defined as:

$$T[ax_1(n) + bx_2(n)] = aT[x_1(n)] + bT[x_2(n)] \tag{2.17}$$
$$y(n) = T[x(n)] \iff y(n-k) = T[x(n-k)] \quad \text{for all } k \in \mathbf{Z} \tag{2.18}$$

[4] Just as, for the continuous case, the Laplace transform is the generalization of the continuous Fourier transform.

[5] Some scolars prefer the name **linear shift-invariant systems** for discrete signals, since there is no inherent notion of time in a sequence of numbers.

As a consequence, an LTI system is completely defined by its **impulse response** $h(n)$, which is the output of the system to a single **unit impulse** $\delta(n)$ with

$$\delta(n) = \begin{cases} 1 & \text{for } n = 0 \\ 0 & \text{for } n \neq 0 \end{cases} \tag{2.19}$$

An important property is that a convolution of two signals in the time-domain can be expressed by a multiplication of their spectra in the frequency-domain. If X, Y and H are the Z–transforms of x, y and h, then

$$y(n) = x(n) * h(n) \iff Y(z) = X(z) \cdot H(z) \tag{2.20}$$

This is expressed in figure 2.6. The Fourier transform $H(z)$ of the impulse response $h(n)$ of a filter is called the **transfer function** of the filter.

The above provides the link between the intuitive notion of filtering as a selective attenuation or amplification of certain frequencies in a signal to the mathematical operations of convolution and Fourier transformation.

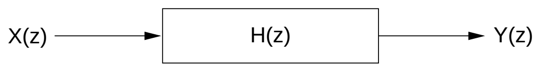

Figure 2.6: Filtering in the frequency-domain.

2.1.5 Windowing

So far, we have considered the signal as a whole, even if it was of infinite length. For practical applications, and in order not to lose all time information in the transition to frequency domain, small sections of the signal of about 10–40 ms are treated one at a time. These small sections are called **windows** or **frames**. To avoid introducing artefacts into the frequency-domain, caused by discontinuities at the borders of the window, the signal is first multiplied by a **window function** $w(n)$, which provides for a smooth fade-in and fade-out of the window.

$$x_w(n) = x(n)w(n) \tag{2.21}$$

The windowed signal $x_w(n)$ is then converted by the Fourier transformation. The windows are usually placed so that they overlap (see figure 2.7).

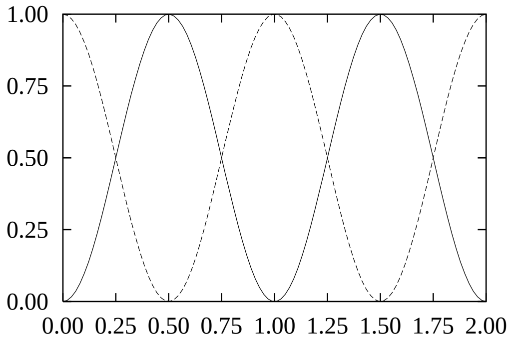

Figure 2.7: Overlapping Hamming windows

In the following, the formulas and graphs of some of the most widely used window functions are shown, together with the magnitude spectra of a signal consisting of two sinusoids at different frequencies and amplitudes, multiplied by the corresponding window function. These spectra show the distortion or smear that is introduced by the window function. Ideally, there would be only two sharp peaks.

Rectangle (figure 2.8)

$$w(n) = \begin{cases} 1 & \text{for} \quad 0 \leq n < N \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

Bartlett or triangular window (figure 2.9)

$$w(n) = \begin{cases} n\frac{2}{N-1} & \text{for} \quad 0 \leq n \leq \frac{N-1}{2} \\ 2 - n\frac{2}{N-1} & \text{for} \quad \frac{N-1}{2} < n < N \end{cases} \tag{2.23}$$

Hamming (figure 2.10)

$$w(n) = 0.54 - 0.46 \cos \frac{2\pi n}{N-1} \tag{2.24}$$

Hanning (figure 2.11)

$$w(n) = 0.5 - 0.5 \cos \frac{2\pi n}{N-1} \tag{2.25}$$

Blackman (figure 2.12)

$$w(n) = 0.42 - 0.5 \cos \frac{2\pi n}{N-1} + 0.08 \cos \frac{4\pi n}{N-1} \tag{2.26}$$

Gauss (figure 2.13)

$$w(n) = e^{\frac{\left(n - \frac{N-1}{2}\right)^2}{-2\sigma^2}} \qquad \text{with} \quad \sigma = \frac{14.41N}{100} \tag{2.27}$$

2.2 Additive Analysis and Synthesis of Sound

On closer scrutiny of the Fourier spectrum of the sound of a musical instument or the voice, a specific structure can be observed: Besides a peak at the **fundamental frequency** f_0 (the frequency heard as the **pitch** of the sound), there are peaks at 2 times, 3 times, 4 times, and so on, of the fundamental frequency, as can be seen in figure 2.14.

These peaks are in fact the frequency-domain representation of sinusoids at integer multiples of the fundamental frequency, called **harmonics**, or **harmonic partials**. Taking advantage of this, it is possible to represent a sound by a list of the amplitudes and phases of the harmonic partials and the residual noise. The latter is the part of the spectrum which can not be expressed by sinusoids at integer multiples of the fundamental frequency. The residual noise is usually very low, as in figure 2.14, but it can also be much louder and contribute essentially to the characteristics of a timbre, e.g. in the breath noise in the attack of a wind instrument, as in figure 2.15, or the thump of the hammer of a piano. In general, the sharp transients in the attack phase of an instrumental sound are best modeled by noise.

Not all sounds, however, can be expressed by harmonic partials. Take for example the spectrum of the sound of a bell in figure 2.16. The partials are spaced at fractional multiples of the fundamental frequency. Nevertheless, these sounds can be represented by the sinusoidal partials and noise model, when the frequency of each partial is recorded, along with its amplitude and phase.

Most of the classical instruments and the voice have a purely harmonic spectrum. Some sounds, like the bell or metal plates have inharmonic spectra, which are mostly discribed as “metallic”. Percussion sounds, finally, have only a feeble partial structure, and consist mainly of noise.

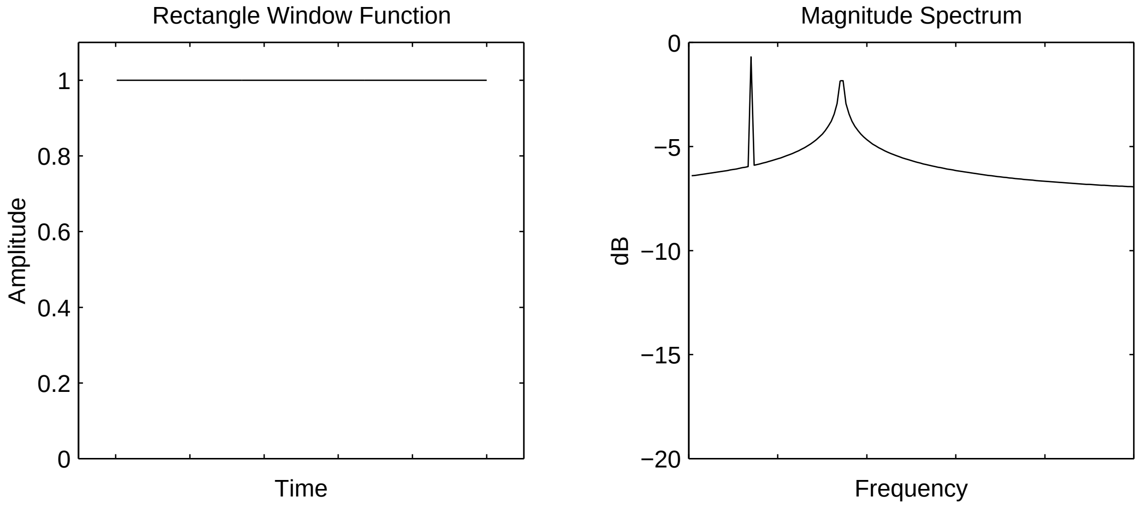

Figure 2.8: The Rectangle window function and its effect

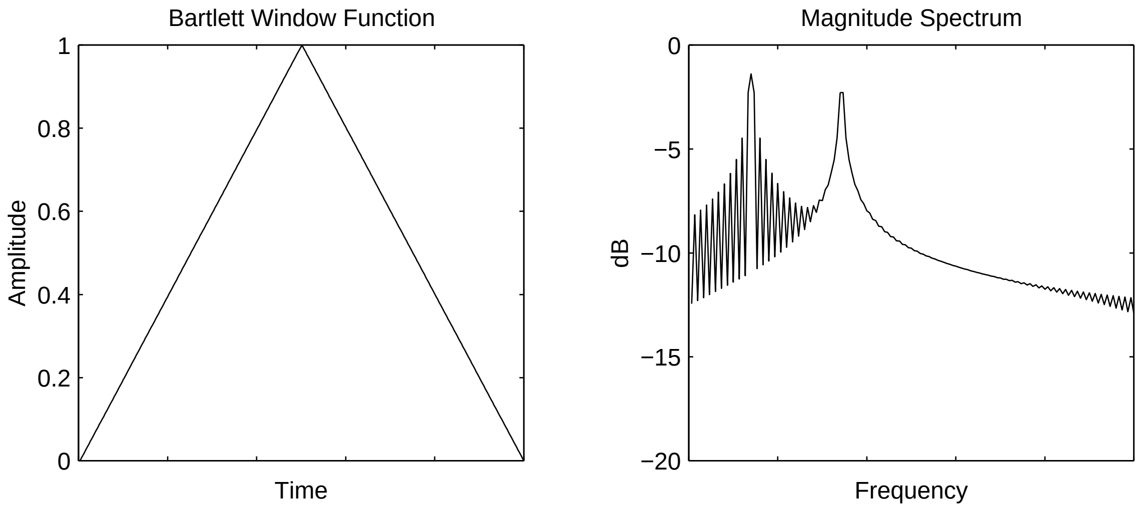

Figure 2.9: The Bartlett window function and its effect

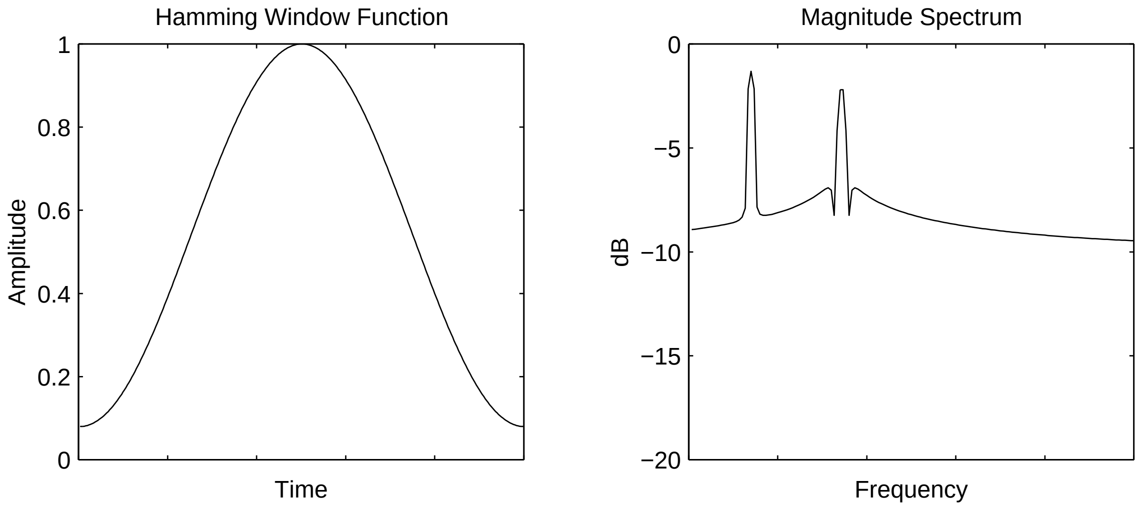

Figure 2.10: The Hamming window function and its effect

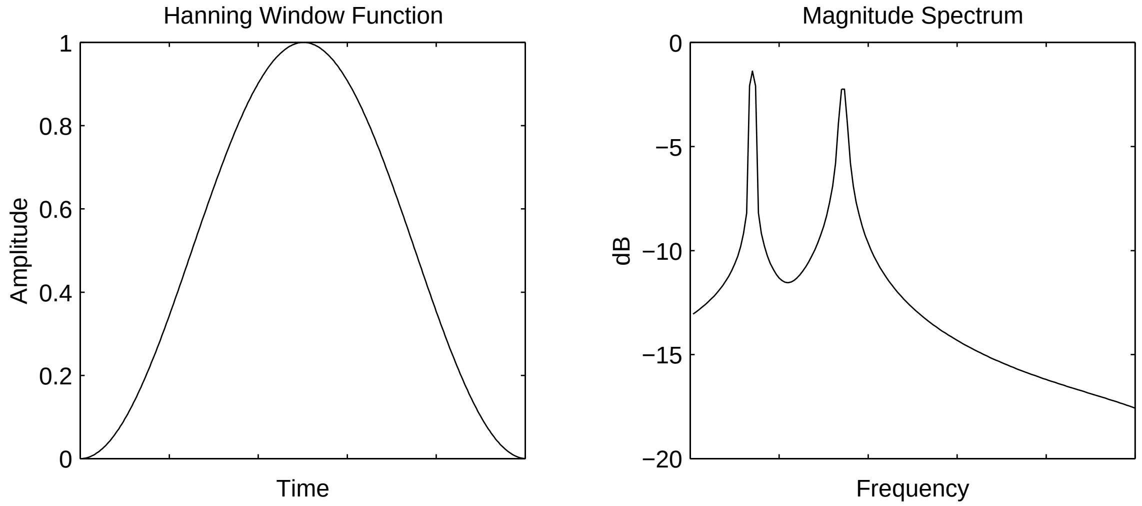

Figure 2.11: The Hanning window function and its effect

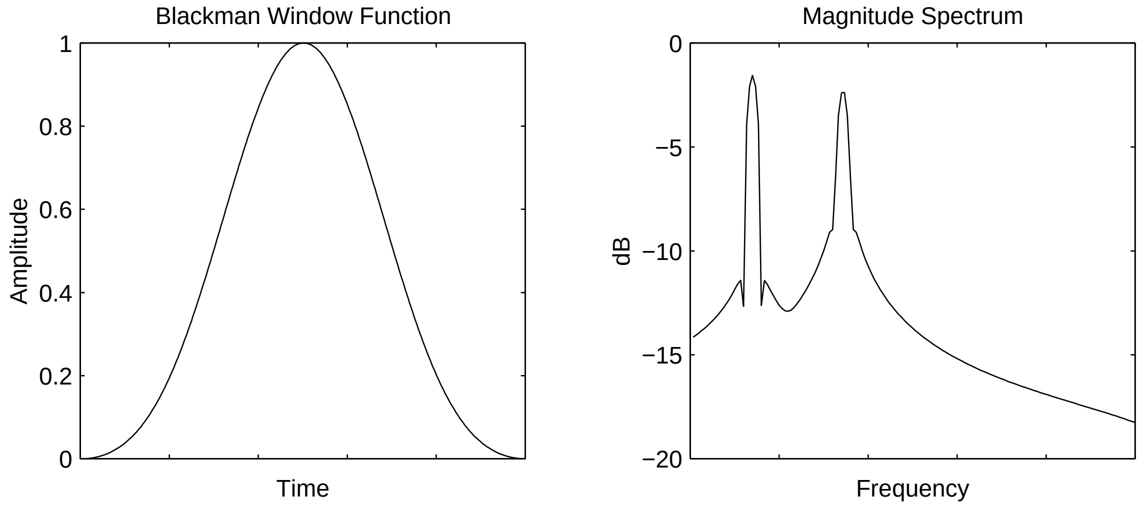

Figure 2.12: The Blackman window function and its effect

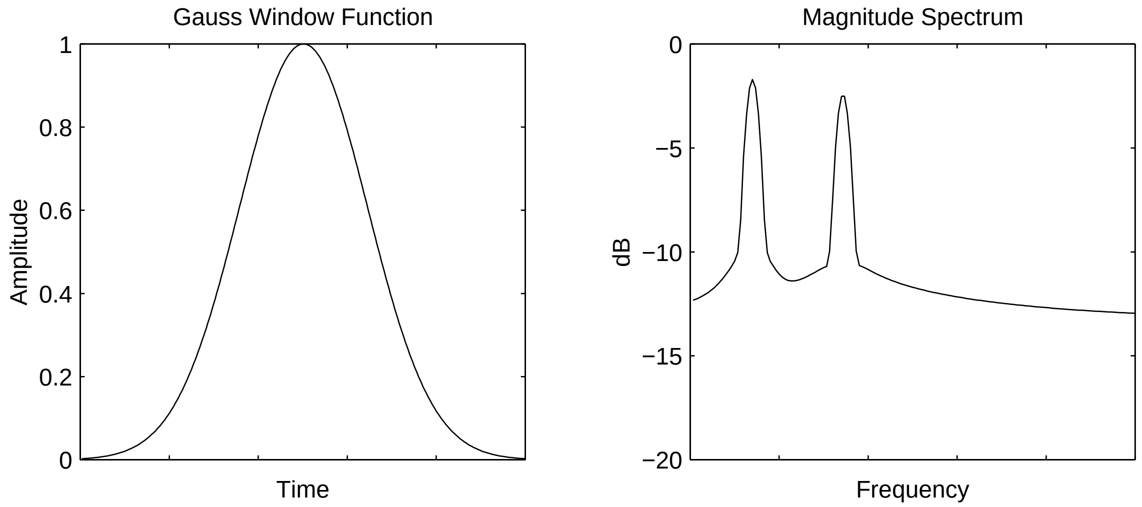

Figure 2.13: The Gauss window function and its effect

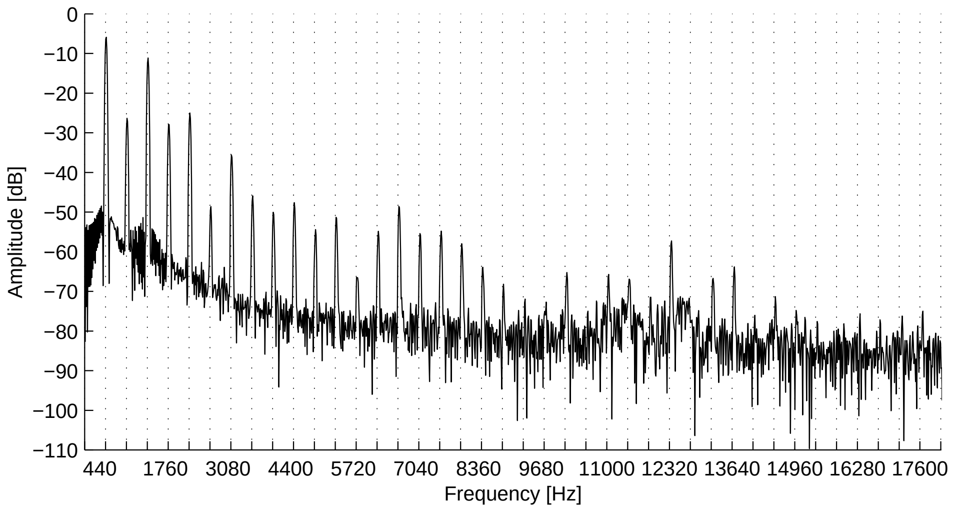

Figure 2.14: Spectrum of a clarinet played at 440 Hz. The grid lines spaced at 440 Hz intervals show that the sound is made up only of harmonic partials.

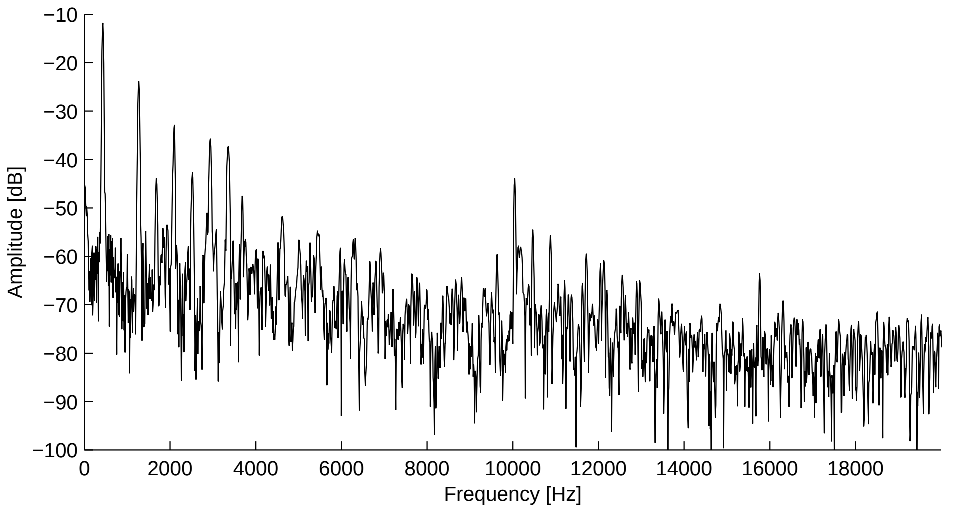

Figure 2.15: Magnitude spectrum of a shakuhachi flute. This spectrum clearly reveals a harmonic partial structure, but the residual noise is much louder than in figure 2.14, in relation to the partials.

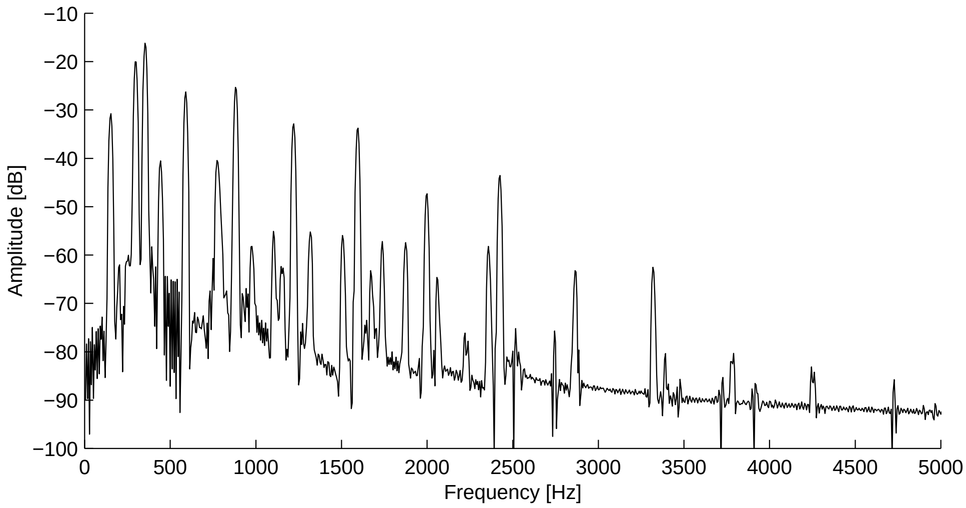

Figure 2.16: Spectrum of a bell sound (an inharmonic spectrum)

Of course, the character of the sound of an instrument is not determined by one spectrum only, but by its evolution in time. Especially, the changes of the proportions of the amplitudes of the partials, and little fluctuations of the partials over time allow us to recognise an instrument. Moreover, especially the attack phase (the first few milliseconds) of a sound contains essential information for the recognition of the instrument. For more about the acoustic correlates of the character of musical instruments see [vH54].

If we now turn to **additive synthesis**, we can easily generalize over harmonic or inharmonic sounds. All sounds are represented by a succession of time-frames consisting of a sinusoidal component with partials at frequencies f_i with amplitudes a_i and phases ϕ_i, $i = 1..n$, and a residual noise component $r(n)$ as a time-domain discrete signal. Synthesis for a frame at time t is done by evaluating

$$s(k) = r(k) + \sum_{i=1}^{n} a_i \cos(2\pi f_i t + \phi_i) \tag{2.28}$$

for all samples $s(k)$ of this frame.

For example, the additive analysis of the sound of the japanese shakuhachi flute of figure 2.15 yields the harmonic partial peaks shown as crosses in figure 2.17. The additive re-synthesis according to equation (2.28) has the spectrum shown in figure 2.18. No non-sinusoidal components are present in this spectrum. The irregular bottom line is due to the unavoidable spectral smear of the FFT-window for displaying the spectrum, as demonstrated in section 2.1.5. Now, if we subtract the resynthesized signal from the original signal in time-domain(!), all that is left is the residual part $r(n)$, consisting of non-sinusoidal noise, the spectrum of which is shown in figure 2.19. This signal contains the typical breath noise of shakuhachi playing. With other instruments, the residual signal reveals to us the clicking of the keys of a clarinet, the scratching of the bow of violin, the sound of the hammer of a piano hitting the string, without any trace of the vibration that it triggered.

It is now easy to manipulate the harmonic part, given that every single component of the sound is accessible. The two simplest but most effective manipulations being transposing the sound and changing its duration, both independent from each other.[6] Various other manipulations are possible, such as selectively detuning the harmonics or a morphing of the partial structure with that of another sound.

Manipulating the residual part is not that obvious, but can be reasonably accomplished by considering it to be a random noise signal and controlling its spectral envelope (by filtering) over time.

For an in-depth discussion of additive analysis and synthesis, the idea of which was first published in [RM69], see [Rod97a].

2.3 Spectral Envelopes

In this section, I will give a more detailed description of what spectral envelopes are. A **spectral envelope** is a curve in the frequency–amplitude plane, derived from a Fourier magnitude spectrum. It describes one point in time (one window, to be precise). The following properties are desirable for spectral envelopes:

Envelope fit

The curve describes an envelope of the spectrum, i.e. it wraps tightly around the magnitude spectrum, linking the peaks.

[6] When the sound is represented as a sequence of samples in time-domain, changing the duration will alter the pitch and vice versa, e.g. a transposition of an octave (twice the original frequency) will speed the sound up by a factor of two. With nowadays fast DSP chips, this can be compensated for, but the sound quality suffers proportionally to the amount of transposition or tempo change.

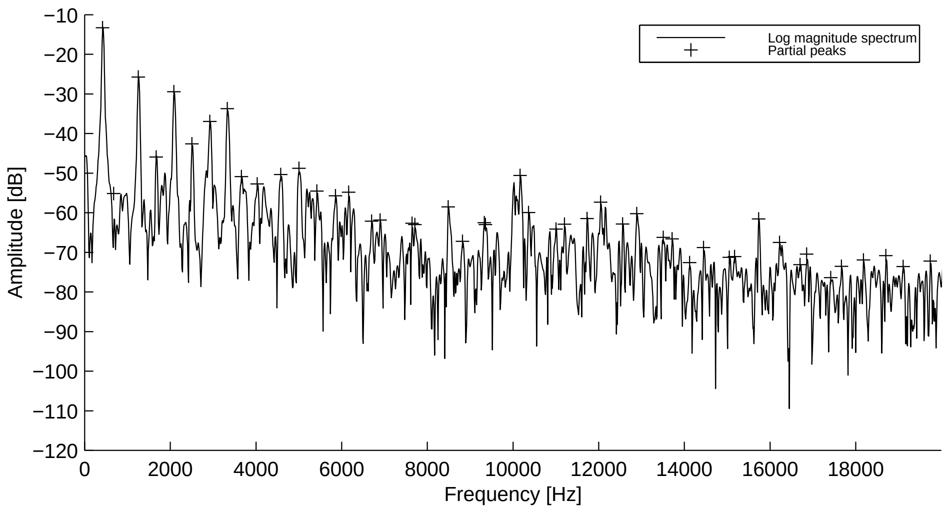

Figure 2.17: Spectrum of a shakuhachi flute and partials found by additive analysis

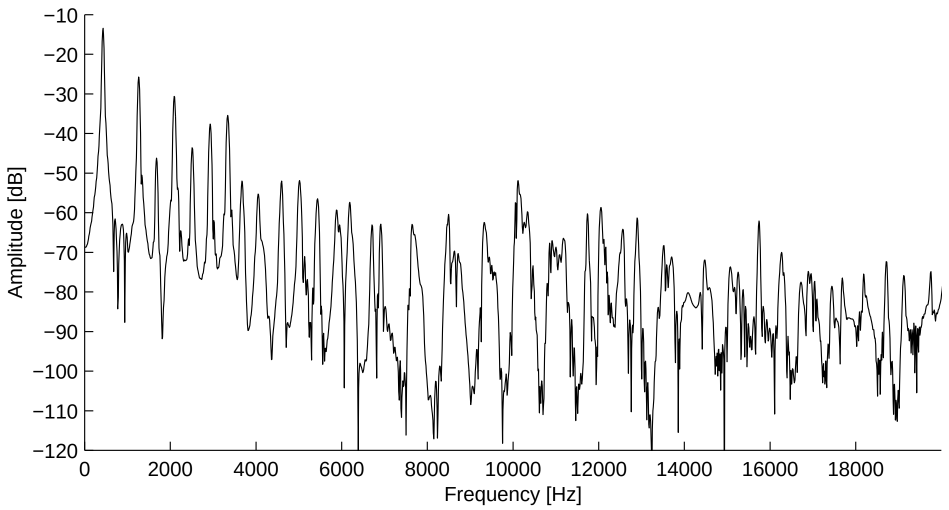

Figure 2.18: Spectrum of the resynthesized sinusoidal part of a shakuhachi flute

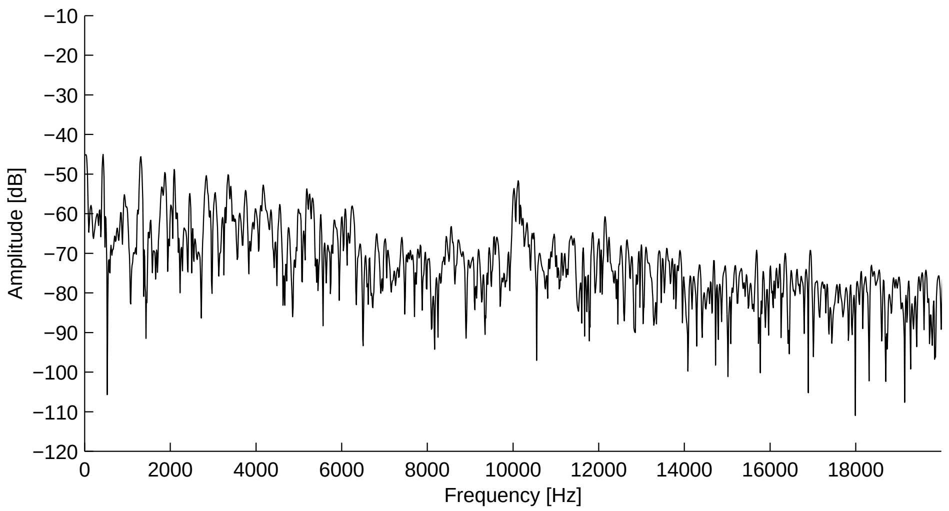

Figure 2.19: Spectrum of the non-sinusoidal residual noise of a shakuhachi flute

Regularity

A certain smoothness or regularity of the curve is required. This means, the spectral envelope must not oscillate too much, but it should give a general idea of the distribution of the signal's energy over frequency.[7]

Steadyness

We want the curve to be steady (in the mathematical sense of a steady function), i.e. it has no corners (where the first derivative jumps).[7]

From the examples of spectral envelopes in figures 2.20 to 2.22 we see that the characteristics of musical instruments lead to distinctive spectral envelopes. The spectral envelope also reflects the class of a vowel (the **phoneme**) in speech, as can be seen in figures 2.23 to 2.25.

2.3.1 Spectral Envelope Correction for Transposition

In speech or in the singing voice, the spectral envelope is quite independent of the pitch (see section 2.4 for why this is so). However, if we transpose the vowel in figure 2.23 up by one octave by multiplying the frequencies of all partials by 2 and performing an additive resynthesis, the spectral envelope will necessarily be transposed also. Figure 2.26 shows this effect which sounds quite unnatural (it is sometimes termed the *mickey mouse effect*). The unnaturalness comes from the fact that the formants are shifted up one octave, which corresponds to shrinking the vocal tract to half of its length. Obviously, this is not the natural behaviour of the vocal tract.

To avoid this, the spectral envelope has to be kept constant, while the partials "slide" along it to their new values. This means that the amplitude of a transposed partial is no longer determined by the amplitude of the original partial, but by the value of the spectral envelope at the frequency of the transposed partial, as in figure 2.27. This way, only the partials are shifted, but the spectral envelope and thus the formant locations stay the same, making the vowel sound natural.

For an easier comparison, figure 2.28 shows the spectral envelopes of the transposed sound with and without spectral envelope correction, on a frequency grid spaced at 366 Hz intervals, the fundamental frequency of the transposed sound. It can be clearly seen that the partials of both are at the same frequencies, but at different amplitudes, and that one spectral envelope is the stretched version of the other (although the compressed spectral envelope lacks some of the details of the stretched one).

2.3.2 Vibrato Tracing of Spectral Envelopes

Xavier Rodet remarked that an interesting way to observe the true spectral envelope of the singing voice is to exploit its independence of pitch, making use of the small but fast variation of pitch while singing with a vibrato. In one test recording, during one period of vibrato of c.a. 200 ms, the fundamental frequency f_0 oscillates around 149 Hz by a maximum deviation in frequency of $b_0 = 3.4$Hz. The harmonic partials at k times f_0 frequency follow, and oscillate by $k \cdot b_0$. Because the spectral envelope stays fixed, they sweep underneath its curve, tracing small portions of its contour, but nevertheless giving its exact slope.

Figure 2.29 shows the partials of all the 20 time-frames during one period of a vibrato, collapsed together into one frame. (The data was generated by Nathalie Henrich [Hen98].) In figure 2.30, a close-up of figure 2.29, the traces left by each partial while they follow the oscillation of the fundamental frequency show up as distinct groups of crosses. All in all, at higher frequencies, the amount of irritating factors augments, and more noise is apparent.

[7] I have deliberately avoided the term **smoothness** here, since it could be understood both as regularity and steadyness. However, in the later chapters, the term smoothness will be used meaning regularity, since steadyness is guaranteed with the methods chosen.

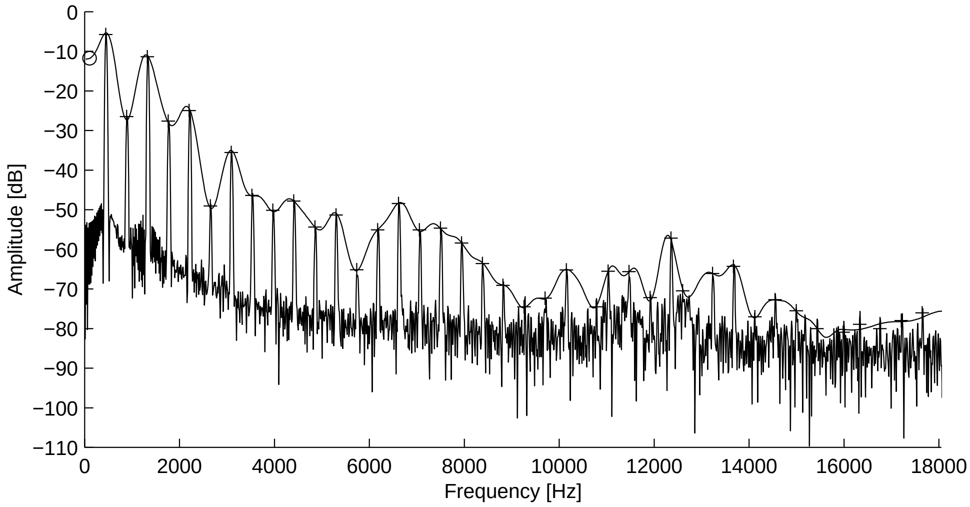

Figure 2.20: Spectrum and spectral envelope of the clarinet sound of figure 2.14

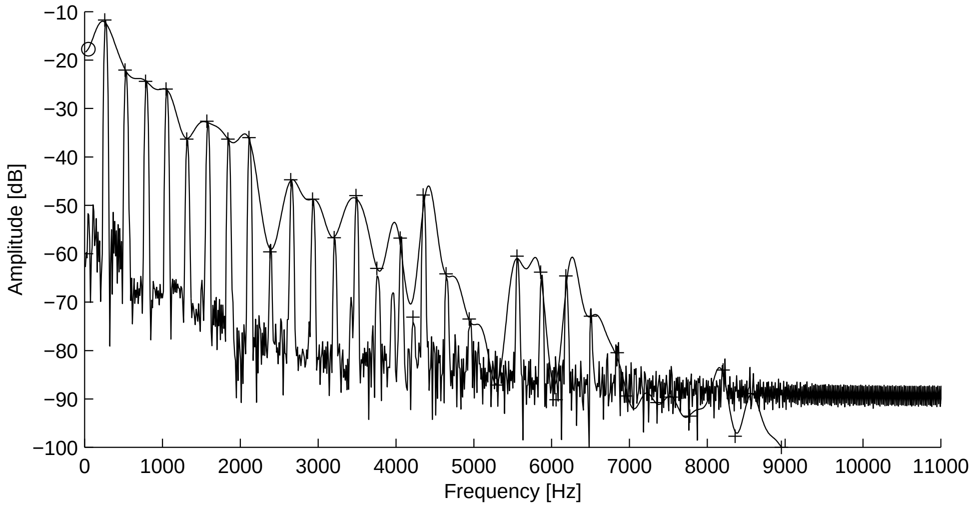

Figure 2.21: Spectrum and spectral envelope of a piano

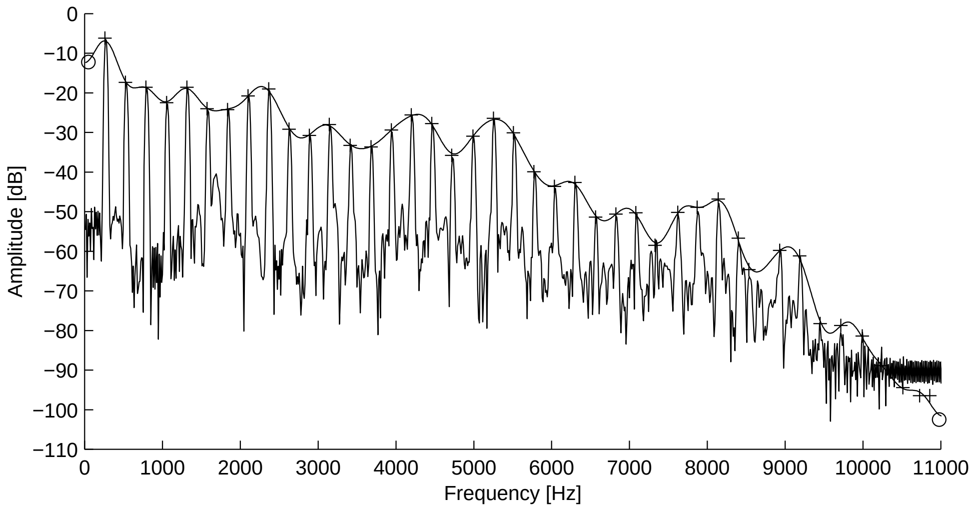

Figure 2.22: Spectrum and spectral envelope of a violin

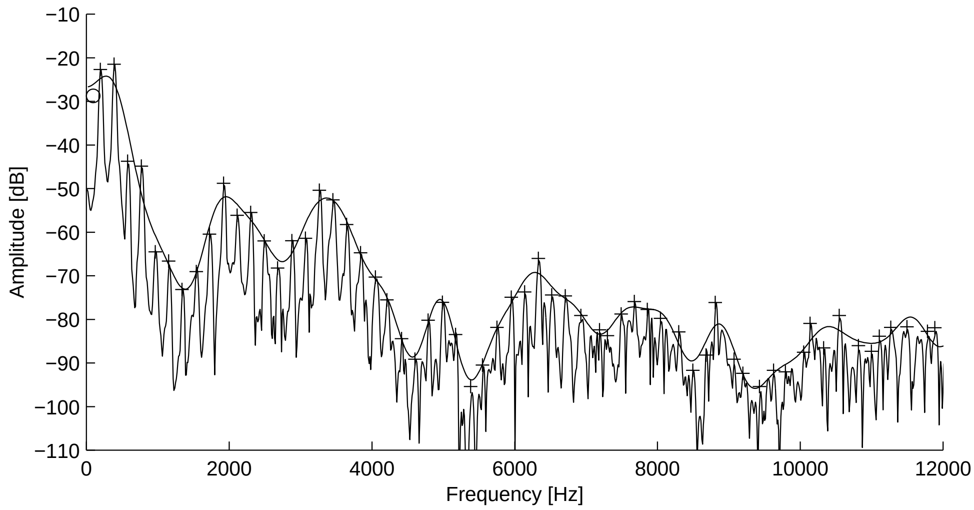

Figure 2.23: Spectrum and spectral envelope of a the vowel /e/

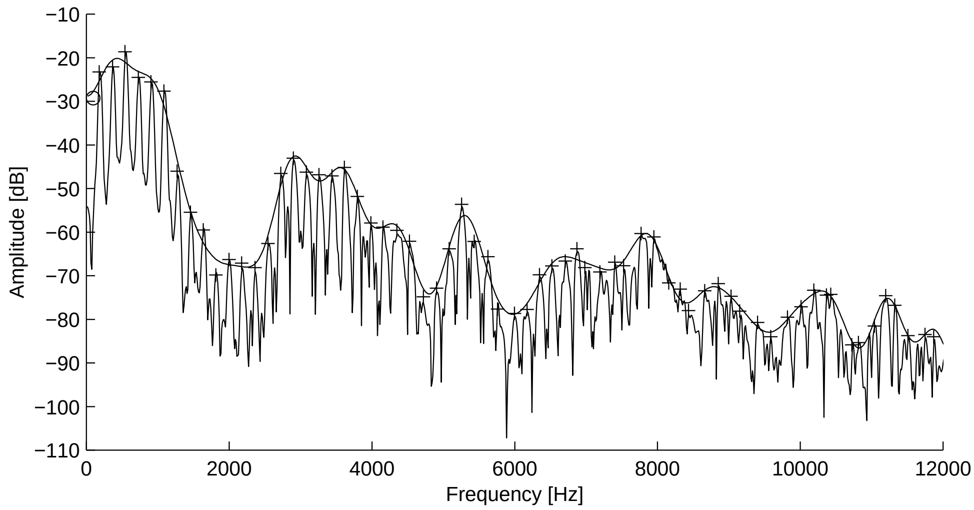

Figure 2.24: Spectrum and spectral envelope of a the vowel /a/

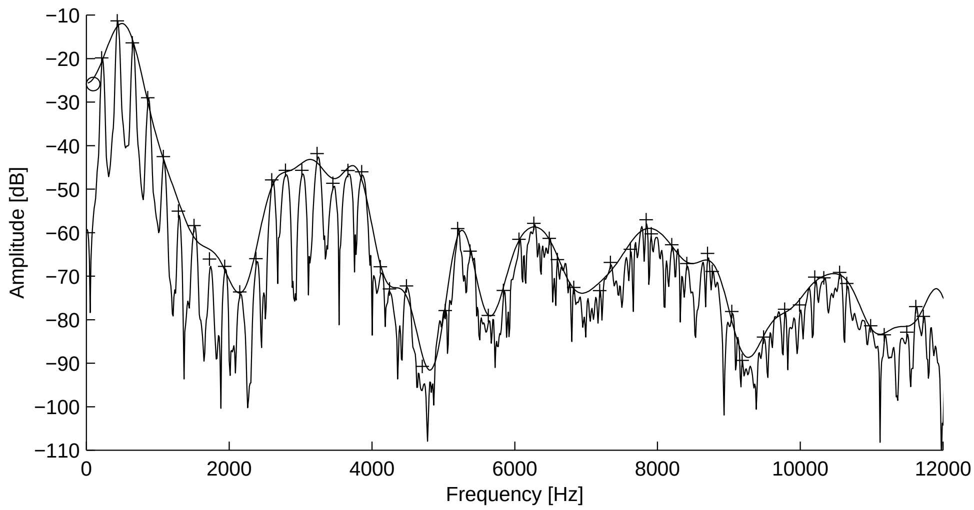

Figure 2.25: Spectrum and spectral envelope of a the vowel /o/

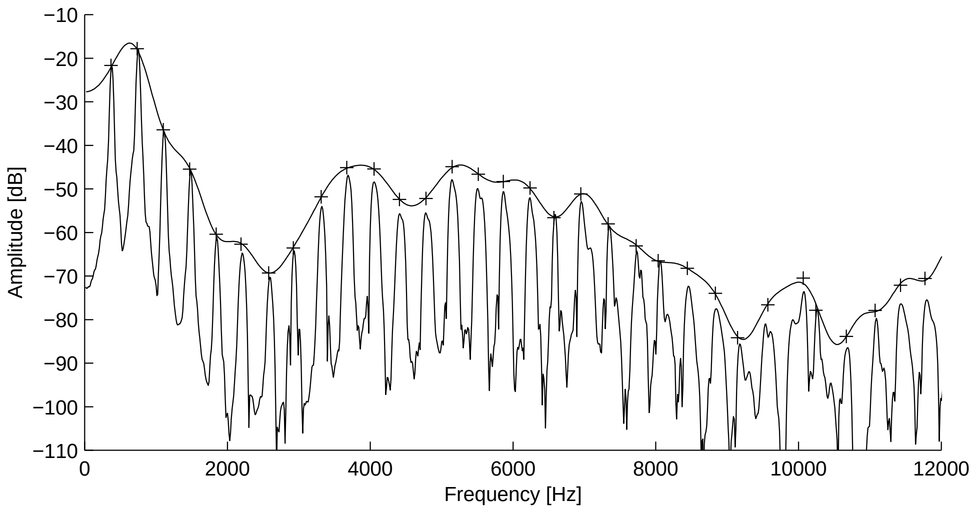

Figure 2.26: Transposition of voice without spectral envelope correction

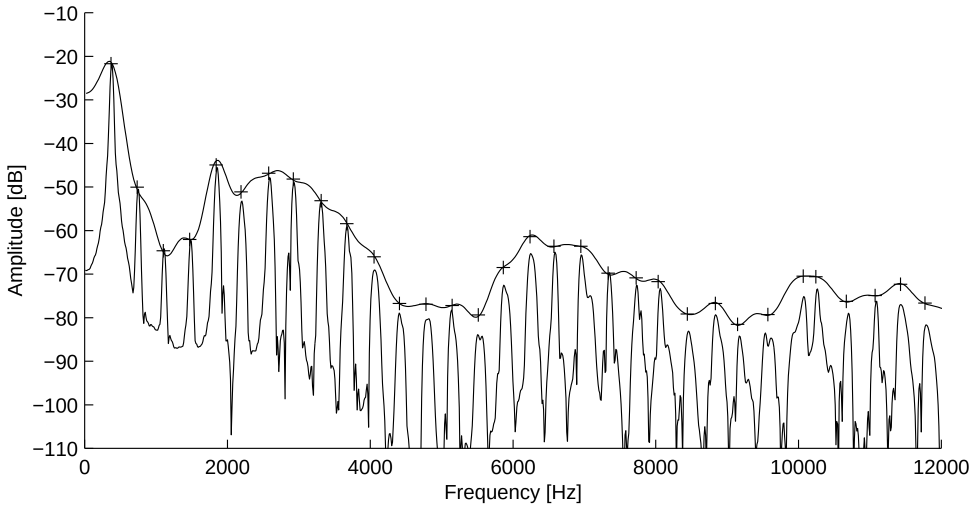

Figure 2.27: Transposition of voice with spectral envelope correction

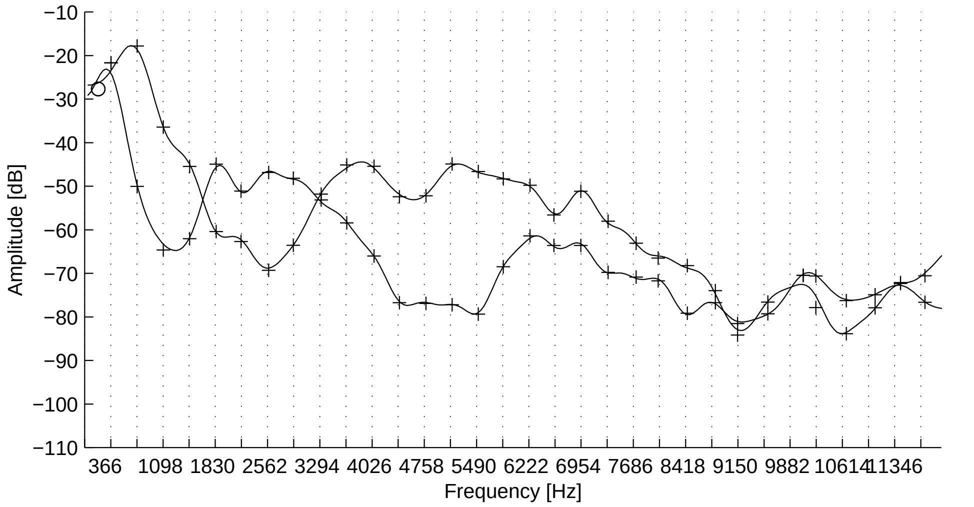

Figure 2.28: Transposition of voice: The spectral envelopes of figures 2.26 and 2.27 are layered to show the effect of transposition with and without spectral envelope correction

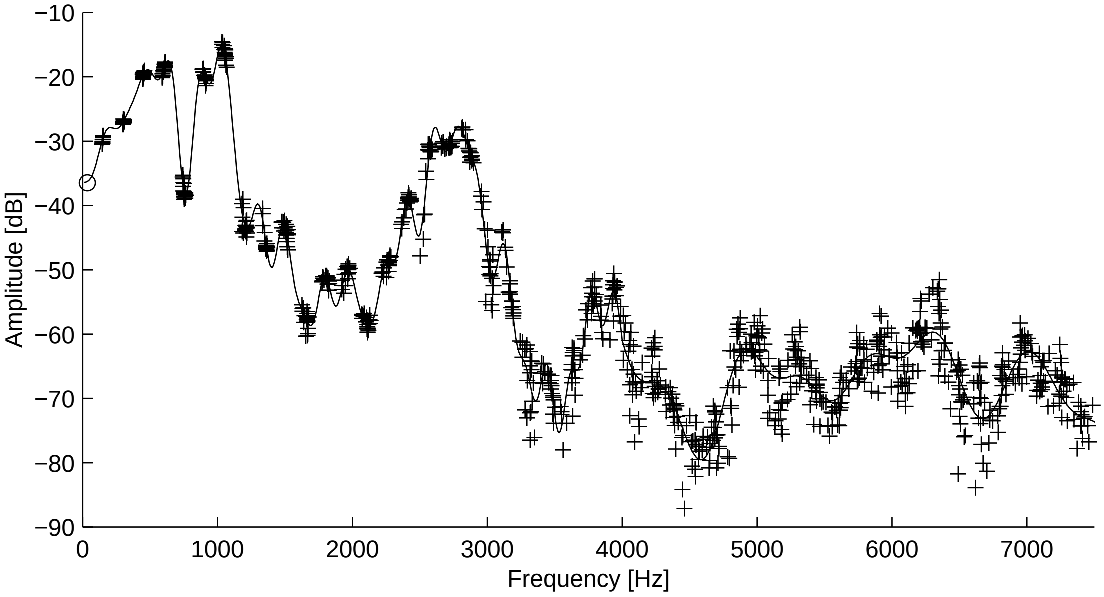

Figure 2.29: Spectral envelope of the singing voice traced by a vibrato. The crosses mark the partials in the frequency–amplitude plane of all time-frames from one period of the vibrato. The spectral envelope follows the slope exactly.

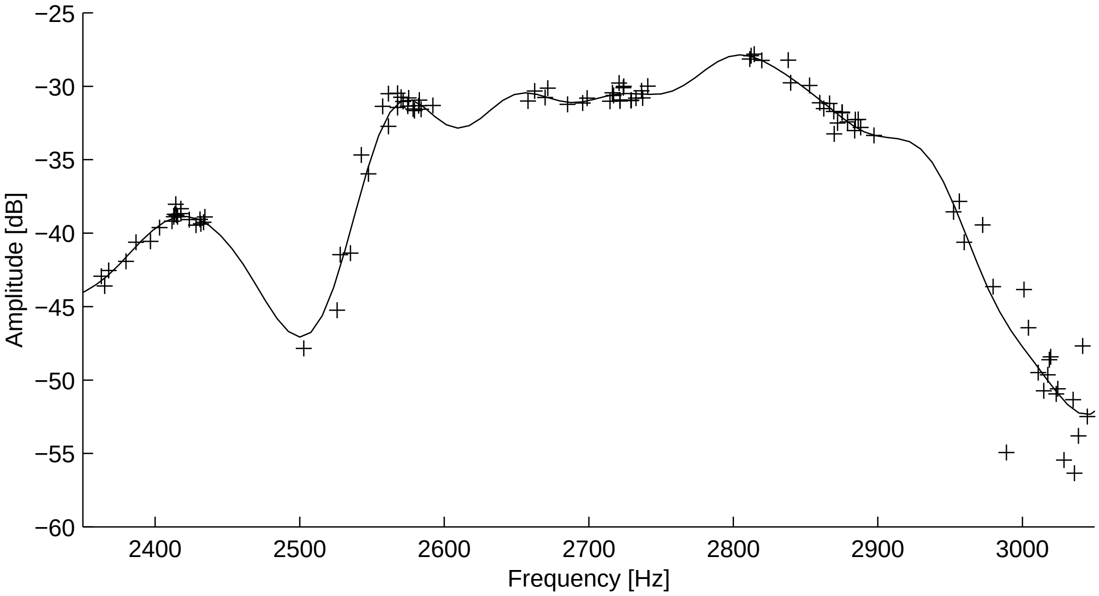

Figure 2.30: Enlargement of figure 2.29

2.4 The Source–Filter Model of Speech Production

Because one of the main applications of spectral envelopes will be the improvement of the analysis and synthesis of the singing voice, we will take a look at a somewhat simplifying, but practical model of speech production. It is well grounded in phonetic and phonological research and applies to the singing voice also. For more details see [CY96].

We distinguish two types of voicing:

Voiced speech is generated by the modulation of the airstream of the lungs by periodic opening and closing of the vocal folds in the glottis or larynx. This is used e.g. for vowels and nasal

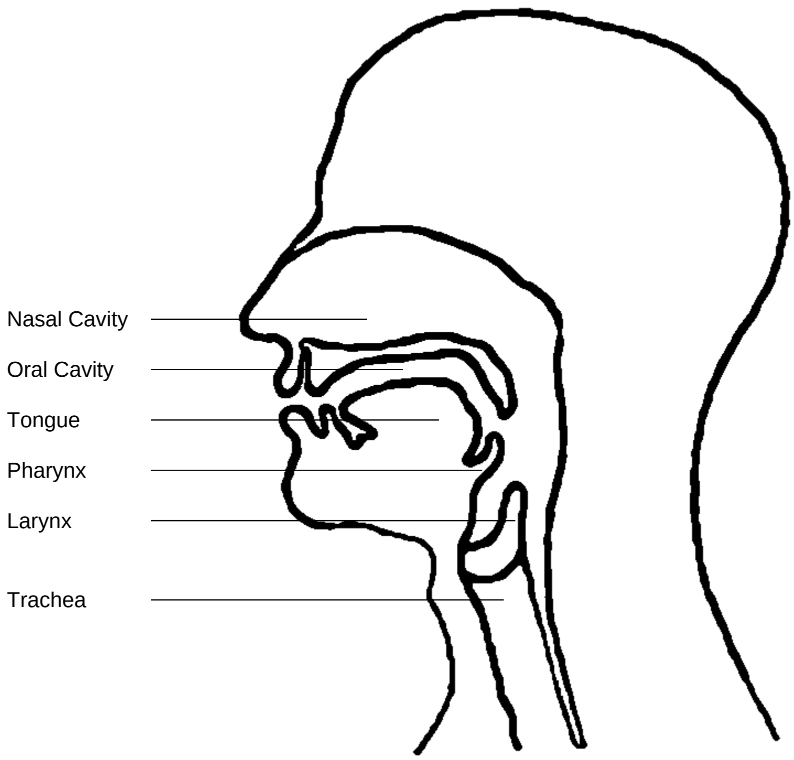

Figure 2.31: Saggital cross section of the vocal tract

consonants like /m/, /n/.

Unvoiced speech is generated by a constriction of the vocal tract (see figure 2.31) narrow enough to cause turbulent airflow, which results in noise, e.g. in fricatives like /f/, /s/, or breathy voice (where the constriction is in the glottis). Unvoiced plosives like /p/, /t/, /k/ fall into this category, too.

Of course, these two types can be applied simultaneously, e.g. in voiced fricatives like /z/ or /v/, or in the voiced plosives /b/, /d/, /g/.

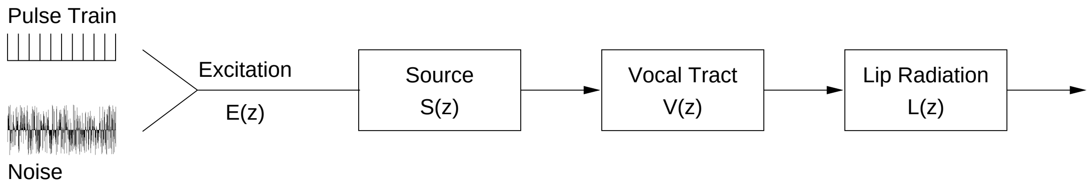

Figure 2.32: Source–filter model of speech production

From a signal processing point of view, this behaviour can be modeled by a linear system (see section 2.1.4) as shown in figure 2.32. The source is modeled by either an impulse train for the voiced speech component, or a random signal for the unvoiced component, yielding an excitation signal $e(n)$ with Fourier transform $E(z)$. The actual source is the waveform of the vibration of the vocal folds or the turbulence caused by a constriction. Its spectrum is obtained by filtering with the source filter $S(z)$. Then, the effect of the shape of the vocal tract is modeled by $V(z)$. Finally, the radiation characteristics of the lips are taken into account by $L(z)$.

By the associativity of linear systems (equation (2.16)), these three filters can be combined to one single filter $H(z)$ by multiplication of their respective transfer functions:

$$H(z) = S(z)V(z)L(z) \tag{2.29}$$

While the source spectrum $S(z)$ and lip radiation $L(z)$ are mostly constant and well known *a priori*, the vocal tract transfer function $V(z)$ is *the* characteristic part to determine **articulation** and thus the content of the speech being uttered. It deserves therefore our special attention and a closer look how it can be modeled adequately.

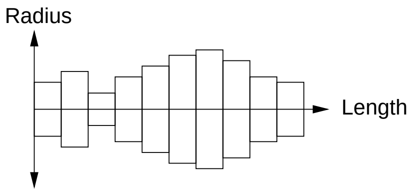

Figure 2.33: Acoustic tube model of the vocal tract

The Acoustic Tube Model

The vocal tract can be viewed as an **acoustic tube** of varying diameter. We can abstract from its curvature and divide it into cylindrical sections of equal width as shown in figure 2.33 (to be rotated around the horizontal axis). Depending on the shape of the acoustic tube (mainly influenced by tongue position), a sound wave travelling through it will be reflected in a certain way so that interferences will generate **resonances** at certain frequencies. These resonances are called **formants**. Their location largely determines the speech sound that is heard.

Like in every model, some details of the complexity of the vocal tract have been omitted. First of all, the nasal tract is completely ignored. This second cavity is shaped very irregularly and introduces additional resonances and anti-resonances (nasal zeros), because of the effect of coupling. Fortunately, the zeros are not vital for the recognition of the speech sound.[8] Next, certain speech sounds like laterals (e.g. /l/) have a tongue configuration which is not at all well described by a simple acoustic tube. Also (non-linear) coupling effects between the vocal tract and the glottis are not taken into account. Finally, the model neither respects the viscuosity of the walls of the vocal tract, nor damping that occurs.

Some of these drawbacks apply also to the source–filter model, but taken as a model of the speech sound, rather than the speech production, it serves remarkably well. Especially, it provides us with a definition of a spectral envelope for the voice: The spectral envelope is the transfer function of the filter part $H(z)$ of the source–filter model, as in equation (2.29). Thus, to estimate the spectral envelope, we need to separate $H(z)$ from $E(z)$, the Fourier transform of the excitation signal. Because the final speech signal is a convolution of the source signal with the impulse response of the filter $H(z)$, this is called **deconvolution**. Chapter 3 will present some methods to accomplish this.

The source–filter model applies to a large class of instruments also, especially those which use a resonating body to amplify the oscillations of a source. For example in string instruments like the guitar or violin, the corpus will have resonant frequencies (and anti-resonances), such that it acts as a filter (albeit constant over time, contrary to the voice). Also in wind instruments like the trumpet, the spectral envelope will be highly independent of the pitch, but varying with playing style (dynamics, lip pressure and stiffness), and it will determine the timbre to a large degree.

2.5 The Software-Environment at IRCAM

This section will give a brief overview of the software systems for sound analysis, synthesis, and processing developed at IRCAM which are related to spectral envelopes. First, the systems which will use spectral envelopes will be shortly described. Then, the systems the spectral envelope library is based upon will be presented. Later, chapter 8 will explain how each system could benefit from spectral envelope handling. In fact, most of the programs will make use of the spectral envelope library developed in this project.

[8]They can, however, lead to problems in formant detection, when they are close to the center frequency of a (broad) formant, they seem to split it into two formants.

ADDITIVE
The ADDITIVE program [Rod97b] performs the additive analysis and resynthesis described in section 2.2. It analyses a sound file according to the sum of harmonic sinusoids (harmonic partials) model whose frequencies are integer multiples of the fundamental frequency f_0. Note that it is crucial to know the (time-varying) fundamental frequency as exactly as possible to be able to recognise the harmonic partials. Therefore, a first analysis step consists of pitch estimation, after which the parameters of the partials (*number, frequency, amplitude* and *phase*) are estimated and written to a parameter file called **format file**. The *number* parameter groups the resulting partials into **tracks** or **partial trajectories**. The synthesis stage takes a partial parameter file as input and computes a synthetic signal which is close to the original signal. In fact, substracting this resynthesised sinusoidal signal from the original leaves the residual part of the signal, i.e. everything which can't be represented by harmonic sinusoids. This proves the tremendous accuracy of the additive analysis method used.

HMM
The HMM program [DGR93] uses a more generalized approach to additive analysis. The underlying model is no longer restricted to harmonic sinusoids, but incorporates inharmonic sinusoids (at fractional multiples of the fundamental frequency) as well. Partial tracking is done by a purely combinatorial **Hidden Markov Model**, using the **Viterbi algorithm**. A partial trajectory is considered as a sequence of peaks in time which satisfies continuity constraints on the slopes of the parameters. The method even allows the frequency lines of partial trajectories to cross.

XTRAJ
To display partial parameter files generated by ADDITIVE or HMM graphically, the program XTRAJ, a descendent of XGRAPH, plots the partial trajectories in the time–frequency plane, while the amplitude of the partials is coded by colour (see figure 2.34).

CHANT
The CHANT project [RPB84, RPB85] was originally intended for the analysis and synthesis of the singing voice, but was quickly expanded to cover general sound synthesis by rule. It is based on the FOF model of synthesis (see section 4.5), a flexible and fast time-domain additive synthesis method. Today, CHANT is implemented in the CHANT-library [Vir97], which is controlled by DIPHONE (see below).

DIPHONE
DIPHONE [RL97] is a graphical sound composition environment which controls additive synthesis and CHANT. It runs on Apple Macintosh and is expandable by plugins (each synthesis method is in fact a plugin). The central concept of the program is that of concatenating diphones: A **diphone** is a segment of a parametric description of sound. When diphones are combined to sequences, the overlapping parts between them will be interpolated, allowing e.g. for astounding morphing between completely different sounds.

FTS / MAX / *jMax*
FTS (*Faster Than Sound*) [Puc91b] is IRCAM's real-time signal processing system, controlled by the graphical programming environment MAX [Puc91a]. It was first developed to run on the IRCAM Signal Processing Workstation (ISPW), a custom-built DSP-card to plug into NeXT-workstations. It has been ported to run natively (i.e. without special signal processing hardware) on SGI workstations and on the Linux operating system [DDPZ94], the new improved user interface *jMax* is based on Java. It is a modular, extensible system, which allows for the setup of any sound synthesis and signal processing algorithm.

The following software systems are used by the spectral envelope library:

UDI
The **Universal DSP Interface** [WRD92] is a portable library of digital signal processing routines. It provides the commonly needed vector and signal processing operations, which will run on a general purpose computer, as well as on fast specialized DSP hardware, if present.

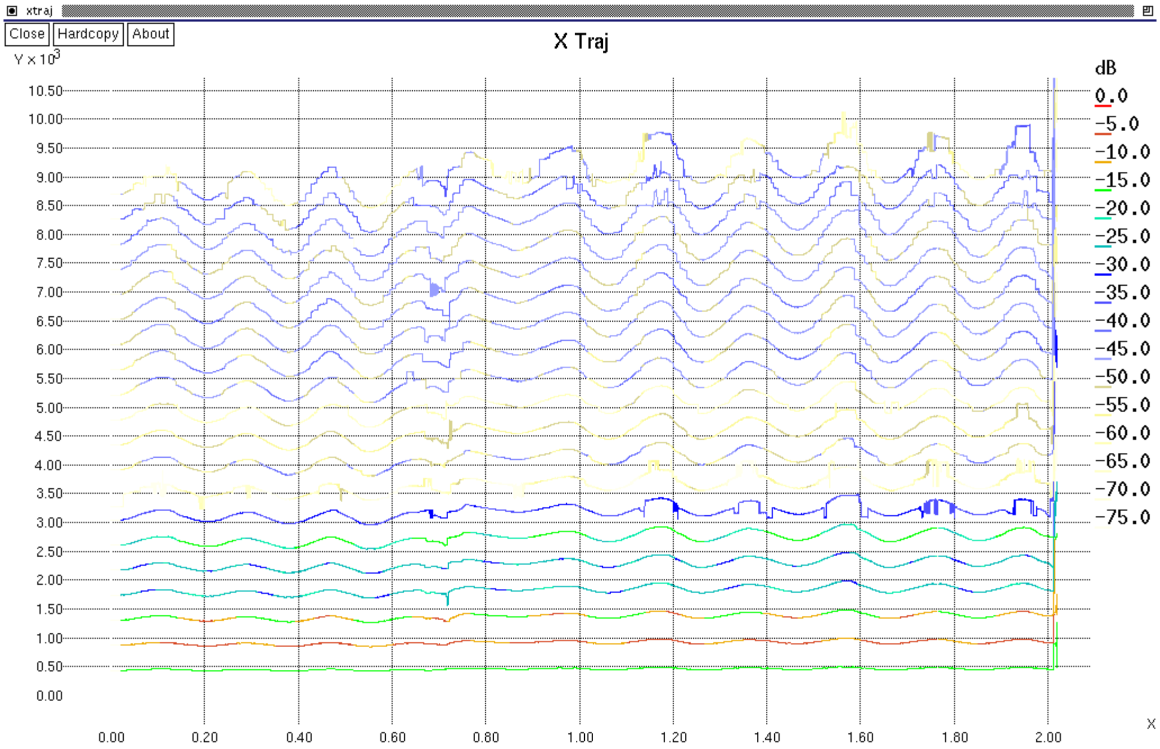

Figure 2.34: Display of partial trajectories of a tenor's vibrato with XTRAJ

PM

PM [Gar94] is a library for additive analysis, transformation, and synthesis. It is the basis for the ADDITIVE program. To the spectral envelope library, it provides functions and data abstractions to handle and manipulate sets of additive partial parameters, time-frames of sets of partials, and break-point functions.

STTOOLS

The STTOOLS library handles the input, output, and conversion of sound files. Both standard file formats like AIFF (Audio Interchange File Format) and the IRCAM's `sf` sound file format can be used. This library is also the basis for the command-line tools to convert, query, and play sound files.

SDIF

The SDIF (*Sound Data Interchange Format*) [Vir98] is a file format developed by IRCAM and CNMAT (Center for New Music and Audio Technologies, Berkeley) which standardizes and unifies the various representations for sound data which surpass a simple time-domain representation as a sampled signal. SDIF is an open, extensible, frame based format. It can combine multiple time-tagged frames of data of different types, and is optimized both for archiving and for streaming. At IRCAM, it is already used for CHANT and additive synthesis. An SDIF-library exists which offers functions to read and write data, and define new data types. This project added the definition of new data types for spectral envelopes. Which are described in detail in section 7.5.

Chapter 3

Estimation of Spectral Envelopes

Estimation is the task of computing the spectral envelope of a signal. First, the general requirements for this step will be introduced in section 3.1, then the estimation methods LPC, cepstrum, and discrete cepstrum will be described in the following sections 3.2–3.5. These descriptions will proceed from a non-formal introduction (*what the algorithm does*) to a detailed formal development (*how it is done*), in some cases followed by a definition of the class of the algorithm in the taxonomy of signal processing systems. For the discrete cepstrum method, an evaluation will be presented in section 3.6.

As the methods for spectral envelope estimation described in this chapter all have their strong and weak points, depending on the signal and the needs of the user, all methods have been implemented in the spectral envelope library, to keep it as flexible as possible.

3.1 Requirements

The requirements for the estimation are basically the fulfillment of the properties of spectral envelopes as described in section 2.3, with some additions and precisations.

Exactness

A precise hit on the point in the frequency–amplitude plane defined by a sinusoidal partial is desirable. In section 2.3 this was called the **envelope fit** property, in that the spectral envelope is an envelope of the spectrum, i.e. it wraps tightly around the magnitude spectrum, linking the peaks.

The degree of exactness is determined by the perceptual abilities of the human ear. In the lower frequency range, it can distinguish differences in amplitude as small as 1 dB. For higher frequencies, the sensitivity is a little lower.

Sometimes it is not possible to link every peak, e.g. when the additive analysis finds a group of peaks close to each other in the upper frequency range. Then, the spectral envelope should find a resonable intermediate path, e.g. through the center of gravity of each frequency slice of the cloud.

Robustness

The estimation method has to be applicable to a wide range of signals with very different characteristics, from high pitched harmonic sounds with their wide spaced partials to noisy sounds or mixtures of harmonic and noisy sounds. Very often, the problems lie in additive analysis in that very low amplitude peaks are identified as sinusoidal partials although they pertain to the residual noise or even to the noise floor of the recording. This is also a question of choosing the right parameters for additive analysis, e.g. the threshold for partial amplitudes.

Regularity
A certain smoothness or regularity is required. This means, the spectral envelope must not oscillate too much, but it should still give a general idea of the distribution of the signal's energy over frequency. This translates to a restriction on the slope of the envelope (given by its first derivative), which may be dependent on context.

Steadyness
We want the curve to be steady (in the mathematical sense of a steady function), i.e. it has no corners (where the first derivative jumps).

3.2 LPC Spectral Envelope

LPC (linear predictive coding, see [MG80, Opp78, Rob98]) is an early method of digital signal processing, developed originally for speech transmission and compression. By the special properties of the method, it can also be used for spectral envelope estimation.

The idea behind LPC analysis is to represent each sample of a signal $s(n)$ in the time-domain by a linear combination of the p preceding values $s(n-p-1)$ through $s(n-1)$. p is called the **order** of the LPC. The approximated value $\hat{s}(n)$ is computed from the preceding values and p **predictor-coefficients** (also called **LPC-coefficients**) a_i as follows:

$$\hat{s}(n) = \sum_{i=1}^{p} a_i \, s(n-i) \tag{3.1}$$

Now, for each time-frame, the coefficients a_i will be computed such that the prediction error $e(n) = \hat{s}(n) - s(n)$ for this window is minimal. For transmission, it is sufficent to send the p coefficients and the residual signal $e(n)$, which uses a smaller range of values and can thus be coded with fewer bits. The receiver can easily recover the original signal from $e(n)$ and the a_i.

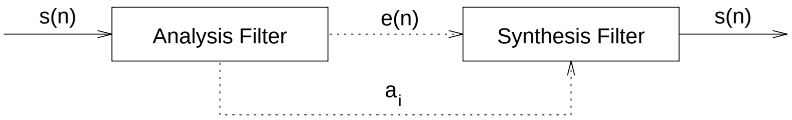

Figure 3.1: LPC-analysis and synthesis for transmission

Transmitter and receiver can also be regarded as a linear system with an adaptive filter, as shown in figure 3.1. What happens when the residual signal $e(n)$ is minimized, is that the **analysis filter** with a transfer function given by

$$A(z) = 1 - \sum_{i=1}^{p} a_i z^{-1} \tag{3.2}$$

tries to suppress the frequencies in the input signal $s(n)$ that have a high magnitude, in order to achieve a maximally flat spectrum (this is sometimes call **whitening** of a spectrum). The **synthesis filter** on the receiving side is the inverse of the analysis filter: It amplifies the frequencies that have been attenuated by the transfer function of the analysis filter

$$\frac{1}{A(z)} = \frac{1}{1 - \sum_{i=1}^{p} a_i z^{-1}} \tag{3.3}$$

As can be seen, the synthesis filter $1/A(z)$ is an **all-pole filter**, since its transfer function is defined by a rational function with no zero points in the numerator, but with p zero points in the denominator $A(z)$. Because these zero points come in compex-conjugate pairs, the absolute value (the magnitude) of the transfer function of the resulting filter shows $p/2$ **poles**, or peaks.

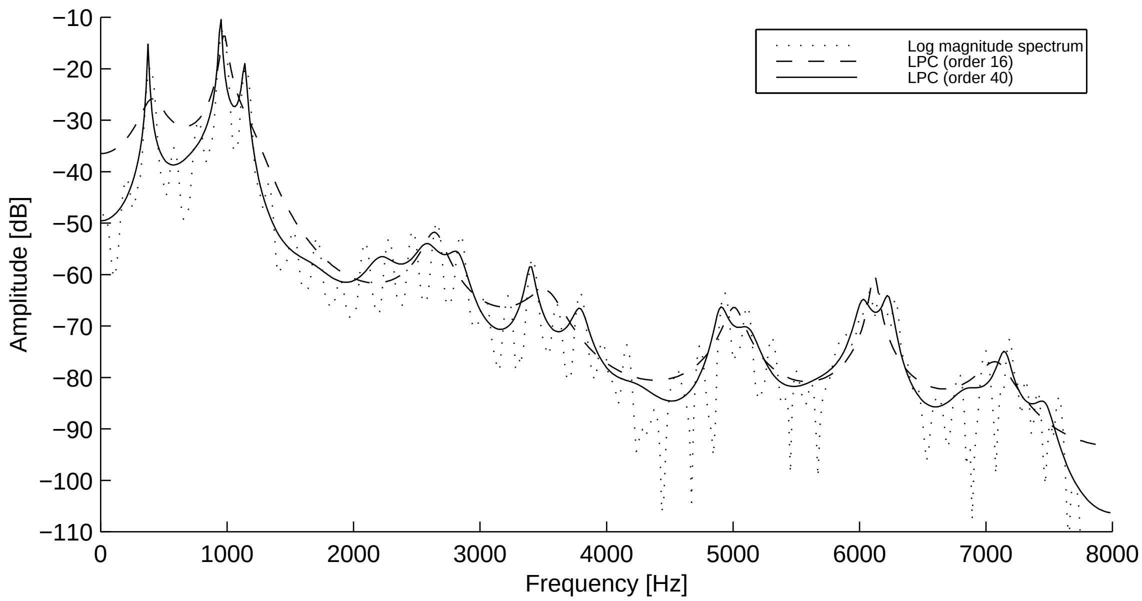

Figure 3.2: The LPC spectral envelope of a mongolian chant. As the order increases, more poles are available to the model, and the envelope follows the spectrum more accurately.

As the analysis filter tries to flatten the spectrum, it will adapt to it in a way that its inverse filter will describe the spectral envelope of the signal. As the order decreases (i.e. fewer poles are available), the approximation of the spectral envelope will become coarser, but the envelope will nevertheless reflect the rough distribution of energy in the spectrum. This can be seen in figure 3.2.

For the actual evaluation of the predictor-coefficients to minimize the prediction error, two classes of methods exist: the **autocovariance method** and the **autocorrelation method**. Both have their advantages and disadvantages [Opp78], however, the autocorrelation method is more widely used, since it can be efficiently implemented using the **Durbin–Levinson recursion**. I won't elaborate on the methods here, since they are amply described in the literature.

In the course of evaluation of the predictor-coefficients, an intermediate set of parameters, the **reflection coefficients** k_i are obtained, which, in fact, correspond to the reflection of acoustic waves at the boundaries between successive sections of an acoustic tube, as presented in section 2.4. These coefficients have advantages for synthesis, and can be interpolated without problems for the validity (stability) of the resulting synthesis filter.

Various other parameter sets exist [MG80, Rob98]: the roots of the analysis filter $A(z)$, log area ratios (LAR), the logarithm of the ratios of the areas of the sections of the acoustic tube model given by $\frac{A_{i+1}}{A_i} = \frac{1-k_i}{1+k_i}$, the line spectral pairs, and others. Since it is possible to convert between them, they don't need to be considered separately for representation.

Disadvantages of the LPC method

A disadvantage of the LPC spectral envelope in analysing harmonic sounds (sounds with a prevalent partial structure) is that it will tend to envelope the spectrum as tightly as possible, and will under certain conditions descend down to the level of residual noise in the gap between two harmonic partials. This will happen whenever the space between partials is large, as in high pitched sounds, and when the order is high enough, i.e. there are enough poles to come to lay on every partial peak. See figure 3.3 for an example of this effect.

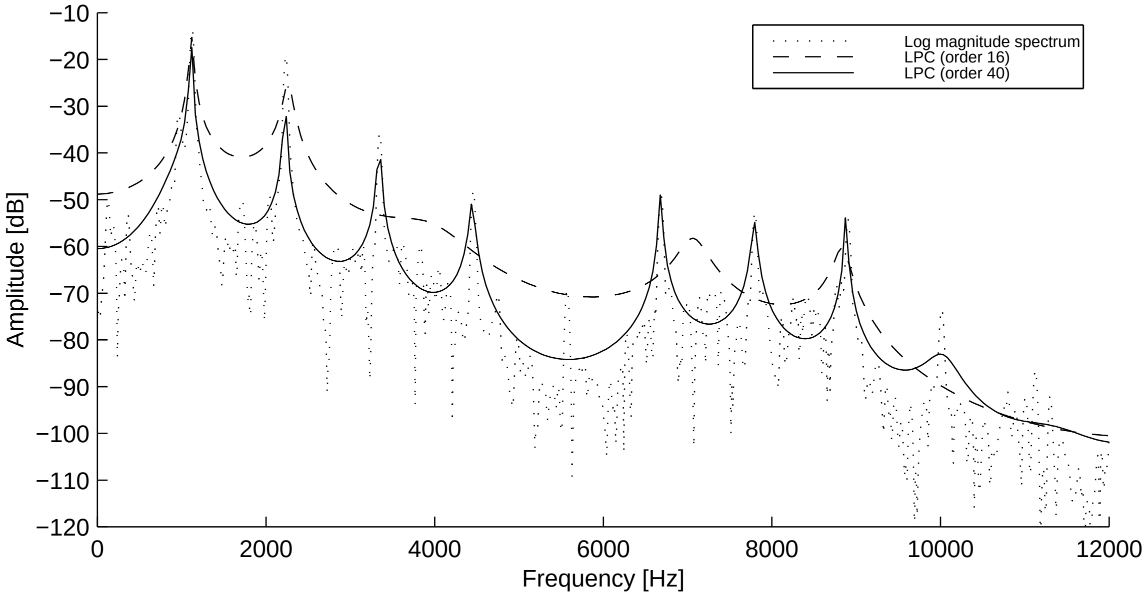

Figure 3.3: Problematic behaviour of the LPC spectral envelope estimation when the partials are spaced far apart. The LPC of order 40 reaches most of the peaks, but "hangs down" in between, while the LPC of order 16 reaches only two peak exactly and describes the average between peaks and residual noise for the rest.

3.3 Cepstrum Spectral Envelope

The cepstrum is a method of speech analysis based on a spectral representation of the signal. To explain the general idea of the cepstrum method used for spectral envelope estimation, two approaches are possible. First, one can simply think of obtaining the spectral envelope from a Fourier magnitude spectrum by successively smoothing its curve to get rid of the rapid fluctuations. This boils down to applying a low pass filter to the spectrum, interpreted as a signal, which lets only the slow fluctuations (low frequency oscillations of the curve) pass, hence the smoothing.

Second, remembering that a signal can be viewed as a convolution of a source signal with a filter, we have to somehow separate the source spectrum from the filter transfer function, which is a very good estimation of the spectral envelope.

According to the source–filter model of speech production introduced in section 2.4, a speech signal $x(n)$ can be expressed as a convolution between a source or excitation signal $e(n)$, produced by the glottis, and the impulse response of the vocal tract filter $h(n)$:

$$x(n) = e(n) * h(n) \tag{3.4}$$

In frequency-domain, this convolution becomes the multiplication of the respective Fourier transforms:

$$X(\omega) = E(\omega) \cdot H(\omega) \tag{3.5}$$

Taking the logarithm of the absolute value of the Fourier transforms (the magnitude spectra, see equation (2.11)), the multiplication of equation (3.5) is converted to an addition:

$$\log |X(\omega)| = \log |E(\omega)| + \log |H(\omega)| \tag{3.6}$$

If we now apply a Fourier transform[1] to the logarithm of the magnitude spectrum, we get the frequency distribution of the fluctuations in the curve of the spectrum c, which is called the **cepstrum**:

$$c = F^{-1}(\log |X(\omega)|) = F^{-1}(\log |S(\omega)|) + F^{-1}(\log |H(\omega)|) \tag{3.7}$$

Under the reasonable assumption that the source spectrum has only rapid fluctuations (the excitation signal e is a stable, regular oscillation of around 100 Hz), its contribution to c will be concentrated in its higher regions, while the contribution of H will be the slow fluctuations in the spectrum of X, and will therefore be concentrated only in the lower part of c, as can be seen in figure 3.4. Thus, the separation of the two components becomes trivial: Only the first p $c_1 \dots c_p$ of the cepstral coefficients c_i are kept, where p is called the **order** of the cepstrum. These represent the low frequency components, i.e. the slowly changing fluctuations, whence the smoothing of the spectrum X to become a spectral envelope. This smoothing effect can be seen in figure 3.5.

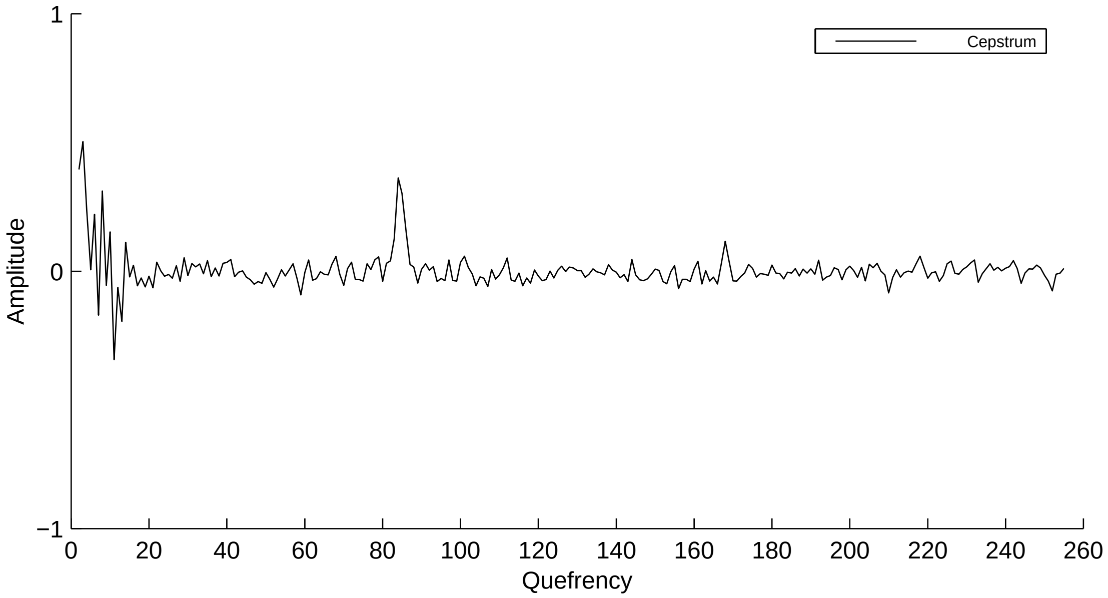

Figure 3.4: The cepstrum c itself. The quefrency is given as the index i of the cepstral coefficients c_i (normally, it is measured in seconds). It can be seen that most information is concentrated in the left part, up to order 20. The sharp peak at about 84 corresponds to the distance of the regularly spaced lobes in the spectrum in figure 3.5, which are the harmonic partials at integer multiples of the fundamental frequency. This is the contribution of the source spectrum.

The unit of the cepstrum was baptised **quefrency**, by virtue of inversing the syllables of *frequency*, analog to *cepstrum* stemming from an inversion of *spectrum*, to reflect the properties of the method. Various other nomenclatura have been invented adhering to the same style, but only these two new words have caught on.

In the taxonomy of signal processing methods, the cepstrum belongs to the class of **homomorphic deconvolution** methods.

To finally obtain the spectral envelope from the cepstral coefficients, one defines the frequencies f_i at which the value of the envelope is to be obtained (the *bins* of the envelope). Usually, one wants n equidistant frequencies up to the Nyquist frequency $f_s/2$:

$$f_i = i\frac{f_s/2}{n}, \qquad i = 1..n \tag{3.8}$$

[1] In fact, we apply an inverse Fourier transform because then we get back the right units and quantities of time and amplitude. Comparing equations (2.9) and (2.10), it can be seen that the Fourier transform differs from the inverse Fourier transform only in a scaling factor and the sign of the exponent.

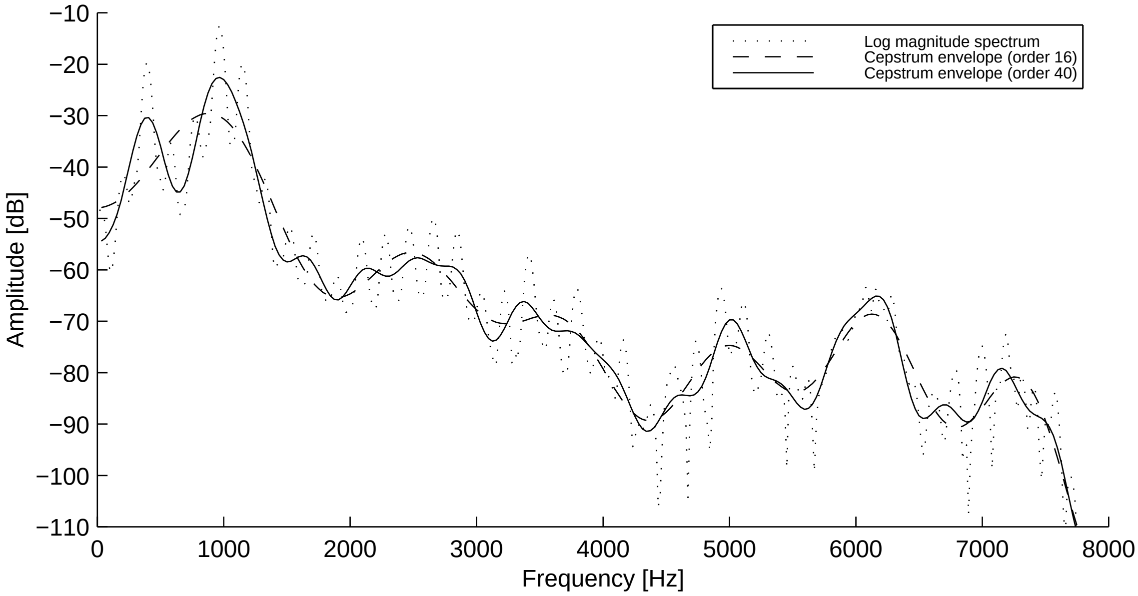

Figure 3.5: The cepstrum spectral envelope of a mongolian chant

Then, after passing to angular frequencies

$$\omega_i = f_i \frac{2\pi}{f_s} \tag{3.9}$$

the envelope value v_i for frequency f_i is

$$v_i = \exp\left(\sum_{j=1}^{p} c_j \cos j\omega_i\right) \tag{3.10}$$

Note that most of this expression is independent of the c_i, especially the expensive cosine evaluation, and can therefore be precomputed as an (n, p) matrix Φ with

$$\Phi_{ij} = \cos j\omega_i \tag{3.11}$$

so that equation (3.10) becomes

$$v = \exp\left(\Phi \cdot c\right) \tag{3.12}$$

Disadvantages of the Cepstrum Method

There are two disadvantages of the cepstrum method of spectral envelope estimation. First, as the cepstrum is essentially a low pass filtering of the curve of the spectrum interpreted as a signal, it will actually average-out the fluctuations of the curve of the spectrum. The effect can be seen in figure 3.5. This is not what we want, because then the resulting curve has no longer the enveloping property to link the peaks of the curve (cf. section 2.3).

Second, similar to LPC, in analysing harmonic sounds (with a conspicuous partial structure) they will follow the curve of the spectrum down to the residual noise level in the gap between two partials, especially when the partials are spaced far apart as for high pitched sounds. See figure 3.6 for an example of this behaviour.

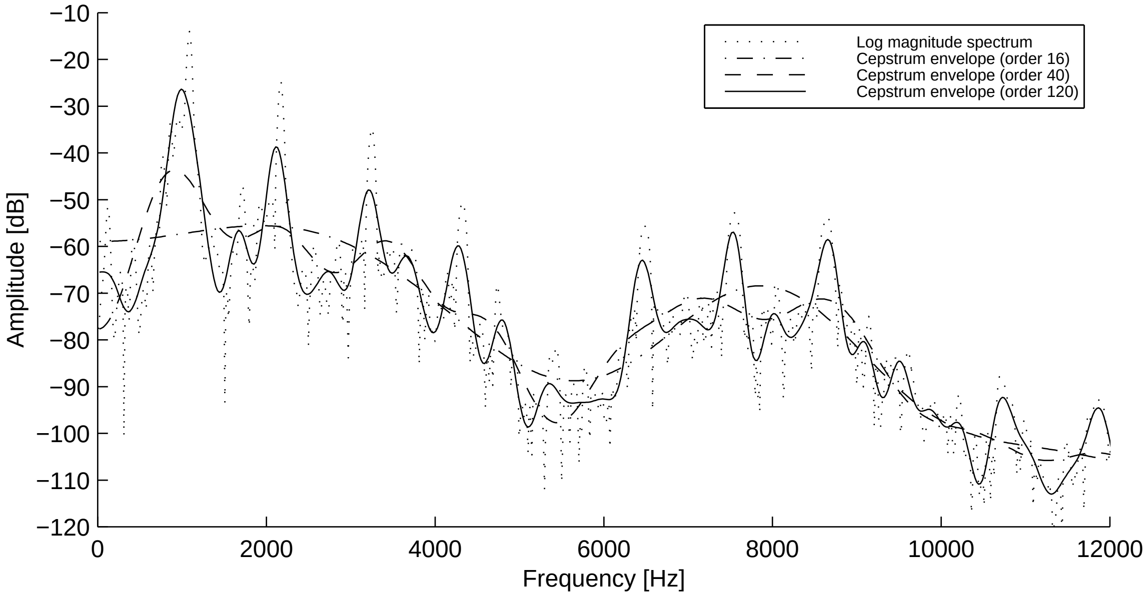

Figure 3.6: Problematic behaviour of the cepstrum spectral envelope estimation when the partials are spaced far apart.

3.4 Discrete Cepstrum Spectral Envelope

Contrary to the previous two methods, LPC, which is computed from the signal, and cepstrum, which is computed from a spectral representation of the signal with points spaced regularly on the frequency axis, the discrete cepstrum spectral envelope is computed from discrete points in the frequency/amplitude plane. These points, which do not have to be regularly spaced in frequency, are the spectral peaks of a sound, which will most often be the sinusoidal partials found by additive analysis (see section 2.2).

As described at the end of sections 3.2 and 3.3, the LPC or cepstrum spectral envelopes will both exhibit the problem to descend down to the level of residual noise between partials which are spaced too far apart, as can be seen in figures 3.3 and 3.6.

The discrete cepstrum, to the contrary, will not care for anything going on in the signal except the partials. It will generate a smoothly interpolated curve which tries to link the partial peaks, as in figure 3.7.

The following method to estimate the discrete cepstrum was developed by Thierry Galas and Xavier Rodet in [GR90, GR91a, GR91b]. A given set of spectral peaks (partials) with amplitudes x_i at frequencies $\omega_i, i = 1..n$, defines a magnitude spectrum X as

$$X = \sum_{i=1}^{n} x_i \delta(\omega_i) \tag{3.13}$$

We consider X to be the result of a convolution

$$X = S \cdot P \tag{3.14}$$

where S is the source spectrum with amplitudes s_i at the same frequencies ω_i as for X

$$S = \sum_{i=1}^{n} s_i \delta(\omega_i) \tag{3.15}$$

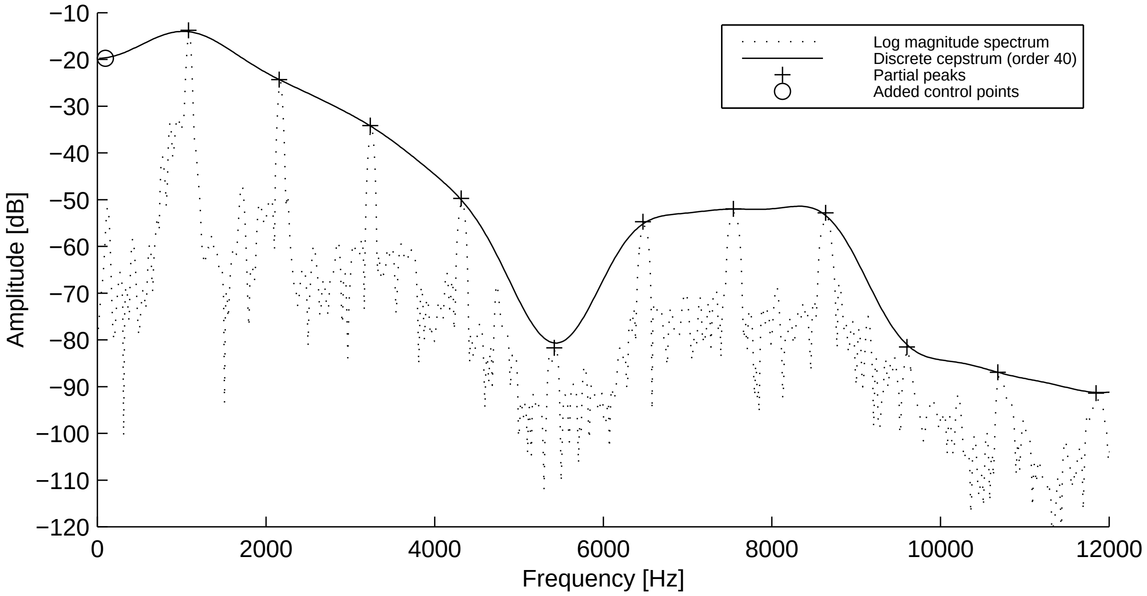

Figure 3.7: Example of a discrete cepstrum spectral envelope

and P is a filter transfer function of a filter modeled by

$$P(\omega) = \prod_{i=0}^{p} e^{c_i \cos \omega i} \tag{3.16}$$

Assuming a flat source spectrum $S(\omega) = 1$ for all ω, all we need to do is find the filter parameters c_i which minimize the quadratic error E between the log spectra. This error criterion is developed from the idea of a spectral distance.

$$E = \sum_{i=1}^{n} \left(\log s_i P(\omega_i) - \log x_i \right)^2 \tag{3.17}$$

To achieve this, one has to simply solve the matrix equation

$$Ac = b \tag{3.18}$$

where A is a matrix of order $p + 1$, where p is called the **order** of the discrete cepstrum, given by

$$a_{ij} = \sum_{k=1}^{n} \cos \omega_k i \ \cos \omega_k j \tag{3.19}$$

c is the vector of filter parameters we're looking for

$$c = \begin{pmatrix} c_1 \\ \vdots \\ c_p \end{pmatrix} \tag{3.20}$$

and b ist the column vector given by

$$b_i = \sum_{k=1}^{n} \log \frac{x_k}{s_k} \cos \omega_k i \tag{3.21}$$

The matrix A can be computed very efficiently by using an intermediate vector R given by

$$r_i = \frac{1}{2} \sum_{k=1}^{n} \cos \omega_k i \tag{3.22}$$

so that $a_{ij} = r_{i+j} - r_{i-j}$.

The matrix equation (3.18) can be efficiently solved applying the Cholesky algorithm, which factorises A such that

$$A = U D U' \tag{3.23}$$

where U is an inferior triangular matrix whose diagonal elements are 1, U' is the transposed matrix (i.e. it is a superior triangular matrix, and D is a diagonal matrix. Now the matrix equation can be solved by simple substitution and division.

The asymptotic complexity of the discrete cepstrum method described above is $O(np + p^3)$, which means that the number of partials n is not of a big concern, since the order is linear in n, but that the order p has to be kept as small as possible, because of its cubic influence.

3.5 Improvements of the Discrete Cepstrum Method

3.5.1 Regularization

The technique of regularization, developed in [GR90, COM97] improves the smoothness of the spectral envelope. Its idea is to penalize too steep a slope of the spectral envelope by adding a regularization term λB to the matrix A, defined in equation (3.19), where B is a quadratic matrix of size $p + 1$, the diagonal of which is defined by:

$$b_{ii} = 8\pi^2 (i - 1)^2. \tag{3.24}$$

Then the discrete cepstrum algorithm proceeds as in section 3.4.

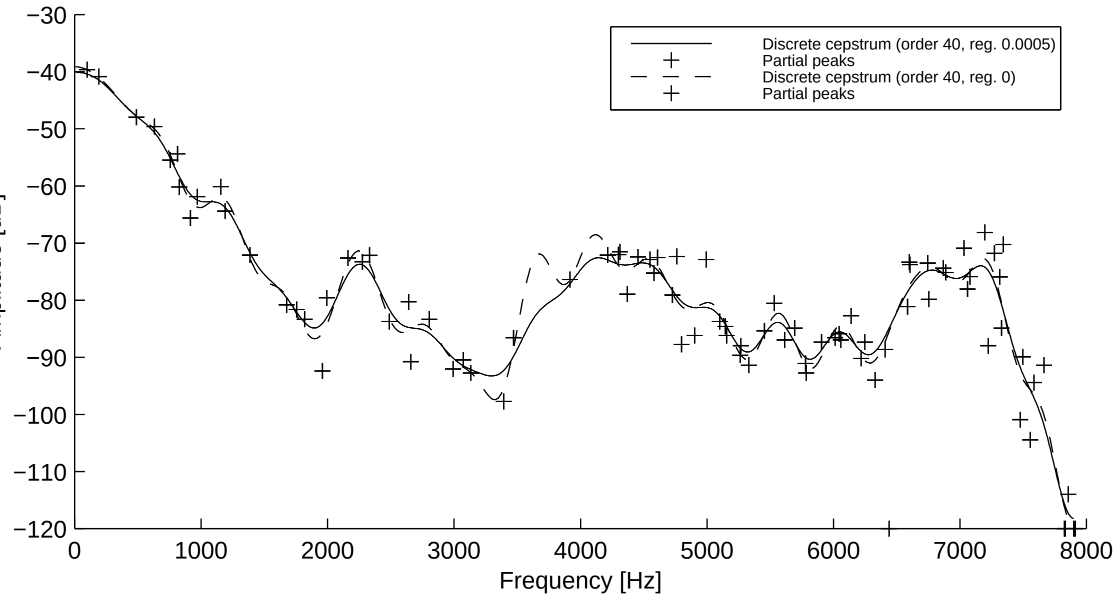

Figure 3.8: The effect of regularization: the unregularized discrete cepstrum spectral envelope ($\lambda = 0$) shows a large hump between 3500 and 4000 Hz, whereas the curve regularized by a factor of $\lambda = 0.0005$ behaves nicely.

The effect of regularization can be seen in figure 3.8. The disadvantage or regularization is that sometimes a steep slope is necessary to reach a single extremely situated peak, as with the low peak at about 3400 Hz in the figure. With regularization, the curve falls short of reaching it.

3.5.2 Stochastic Smoothing (The Cloud Method)

The cloud method developed by Thierry Galas and Xavier Rodet in [GR90] is a way to get a smoother spectral envelope with the discrete cepstrum algorithm. The method generates a cloud of points around each partial on the frequency–amplitude plane to give the discrete cepstrum algorithm more freedom trying to fit a curve that links all the partials.

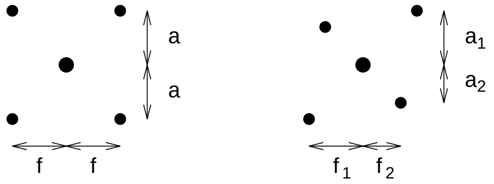

Figure 3.9: The cloud of points around the original partial generated by stochastic smoothing with indifferent slope (left), and with a hint for a rising slope (right)

The added points $x_{1..4}$ are displaced from the original point x_0 at frequency f_0 and amplitude a_0 by a frequency shift f and an amplitude factor a as shown in figure 3.9 left:

$$x_1 = (f_0 - f,\ a_0 {\cdot} a) \tag{3.25}$$
$$x_2 = (f_0 - f,\ a_0/a) \tag{3.26}$$
$$x_3 = (f_0 + f,\ a_0 {\cdot} a) \tag{3.27}$$
$$x_4 = (f_0 + f,\ a_0/a) \tag{3.28}$$

Furthermore, the shape of cloud can be used to influence the behaviour of the spectral envelope, if additional information is known, as shown in figure 3.9 right. For example, with a configuration as in the figure, if it was known that a point is situated in the rising slope of a formant, the spectral envelope could be influenced to also prefer a rising slope. The displacement of the added points is given by

$$x_1 = (f_0 - f_2,\ a_0 {\cdot} a_2) \tag{3.29}$$
$$x_2 = (f_0 - f_1,\ a_0/a_1) \tag{3.30}$$
$$x_3 = (f_0 + f_1,\ a_0 {\cdot} a_1) \tag{3.31}$$
$$x_4 = (f_0 + f_2,\ a_0/a_2) \tag{3.32}$$

However, to avoid too strong a deviation of the spectral envelope from the original point, weighting is introduced in the discrete cepstrum algorithm to attenuate the influence of the added points with respect to the original point. The original point is weighted with a factor of 5, whereas the added points are weighted with a factor of 1, as expressed by the thickness of the points in figure 3.9. The weighting h_i is introduced in the calculation of the error criterion in equation (3.17):

$$E = \sum_{i=1}^{n} h_i \left(\log s_i P(\omega_i) - \log x_i\right)^2 \tag{3.33}$$

Thus, equation (3.19) will become

$$a_{ij} = \sum_{k=1}^{n} h_i \ \cos \omega_k i \ \cos \omega_k j \tag{3.34}$$

and equation (3.21) is changed to

$$b_i = \sum_{k=1}^{n} h_i \log \frac{x_k}{s_k} \cos \omega_k i \tag{3.35}$$

From a more formal point of view, the cloud method is in fact a replacement of each original partial (spectral peak) (ω_i, x_i) by a probability distribution $\pi_i(\omega, x)$. This is due to the impossibility of knowing the precise position of the spectral peaks, which is reflected by the probability distribution, while before a perfect knowledge of the spectral peaks was assumed.

The new error criterion, assuming $s_i = h_i = 1$, is:

$$E = \sum_{i=1}^{n} \iint \pi_i(\omega, x) \left(\log s_i P(\omega_i) - \log x_i\right)^2 \mathrm{d}\omega \mathrm{d}x \tag{3.36}$$

The distribution π_i can be sampled, i.e. each spectral peak (ω_i, x_i) is replaced by a set of peaks (ω_k, x_k), to yield the cloud of points described at the beginning of this section. Formally, for a gaussian distribution

$$\pi_i(\omega, x) = e^{-\alpha^2 (x - x_i)^2} e^{-\beta^2 (\omega - \omega_i)^2} \tag{3.37}$$

the weights would then be $h_k = \pi(\omega_k, x_k)$.

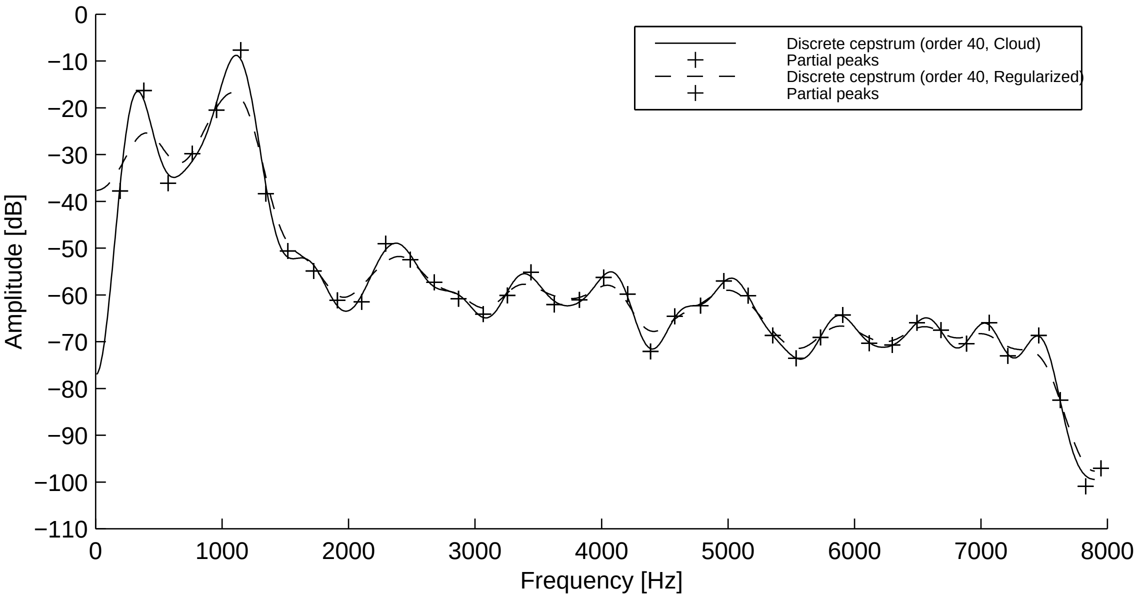

Figure 3.10: Improvement of spectral envelope estimation with stochastic smoothing. The envelope smoothed by the cloud algorithm reaches the two low frequency peaks, while the regularized envelope is too restrained.

Figure 3.10 shows the improvement of discrete cepstrum spectral envelope estimation with stochastic smoothing. The cloud method can also be combined with regularization, described in section 3.5.1 to further improve results.

3.5.3 Logarithmic Frequency Scaling

As we have seen in section 3.4, the discrete cepstrum algorithm is of cubic complexity in p, the order of the discrete cepstrum. This means that we must try to reduce the order necessary for a good estimation of the spectral envelope, to keep computation times short. One way to achieve this is to judiciously spend the preciseness or resolution where it is most needed, and reduce it where it is not so important. We can exploit the properties of the human auditory system, for that matter.

Due to the logarithmic frequency resolution of the human hearing, which also led to the mel frequency scale, we don't need to be very exact with the spectral envelope in higher frequency ranges. It suffices to represent the rough location of energy, whereas in the low frequencies, very slight deviations in frequency and amplitude are perceptible. Therefore we can introduce a logarithmic frequency scaling similar to the mel scale, as suggested in [GR91b], which is linear below a given **break frequency**, and logarithmic above. The mel scale is defined by

$$\mathrm{mel}(f) = \begin{cases} f \cdot \frac{f_m}{f_b} & \text{if} \quad f \le f_b \\ f_m \cdot (1 + \log_{10} \frac{f}{f_b}) & \text{if} \quad f > f_b \end{cases} \tag{3.38}$$

where f_b is the break frequency and f_m is the mel frequency at the break point.

Taking this formula directly poses problems, because frequencies can surpass the Nyquist frequency $f_s/2$, which is to be avoided, because it disturbs the validity of the subsequent calculations. Normalizing the range of the mel function to within the Nyquist frequency yields:

$$\mathrm{melnorm}(f) = \begin{cases} \frac{f}{f_b} \cdot f_n & \text{if} \quad f \le f_b \\ (1 + \log_{10} \frac{f}{f_b}) \cdot f_n & \text{if} \quad f > f_b \end{cases} \tag{3.39}$$

where f_n is the normalization factor given by

$$f_n = \frac{\frac{1}{2} f_s}{1 + \log_{10} \frac{\frac{1}{2} f_s}{f_b}} \tag{3.40}$$

The new logarithmic scaling function *melnorm* no longer depends on f_m.

The effect of logarithmic frequency scaling can be seen in figure 3.11.

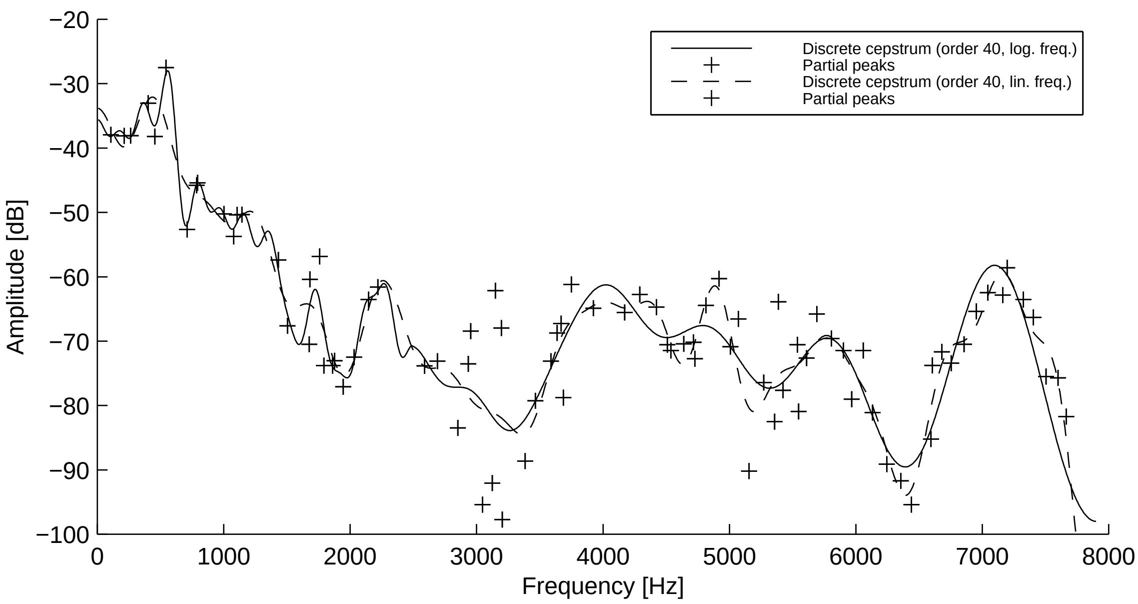

Figure 3.11: Effect of logarithmic frequency scaling. The higher part of the spectral envelope is quite inexact, but the accuracy in the low frequency range (below the break frequency of 2500 Hz) is much better.

As an additional advantage, spectral envelopes stored with a logarithmic frequency support take less space. Also the complexity of synthesis can be reduced when there are fewer points needed to represent a spectral envelope.

To retrieve the spectral envelope from cepstral coefficients obtained with logarithmic frequency scaling, the frequencies f_i of the bins of the envelope (see end of section 3.3) have to be converted to logarithmic scale:

$$f_i' = \mathrm{melnorm}(f_i) \tag{3.41}$$

Then proceed with equations (3.9) and (3.10).

There is one pitfall in the application of logarithmic frequency scaling: Performing the linear-to-logarithmic transformation before applying the cloud deteriorates the results slightly. To see why this is so, remember that the cloud algorithm (section 3.5.2) adds points with a constant linear shift around each peak frequency, which will subsequently be stretched for the linear part or unsymmetrically converted to the logarithmic scale for the rest.

3.5.4 Adding Points to Control the Envelope

There are two situations where the behaviour of the discrete cepstrum spectral envelope has to be controlled by adding artificial points to the partials for discrete cepstrum estimation: at the borders and between the highest partial and the upper border. If the cloud method of stochastic smoothing (section 3.5.2) is selected, these points would be added before the cloud is applied (i.e. each added point will have a cloud of points around it).

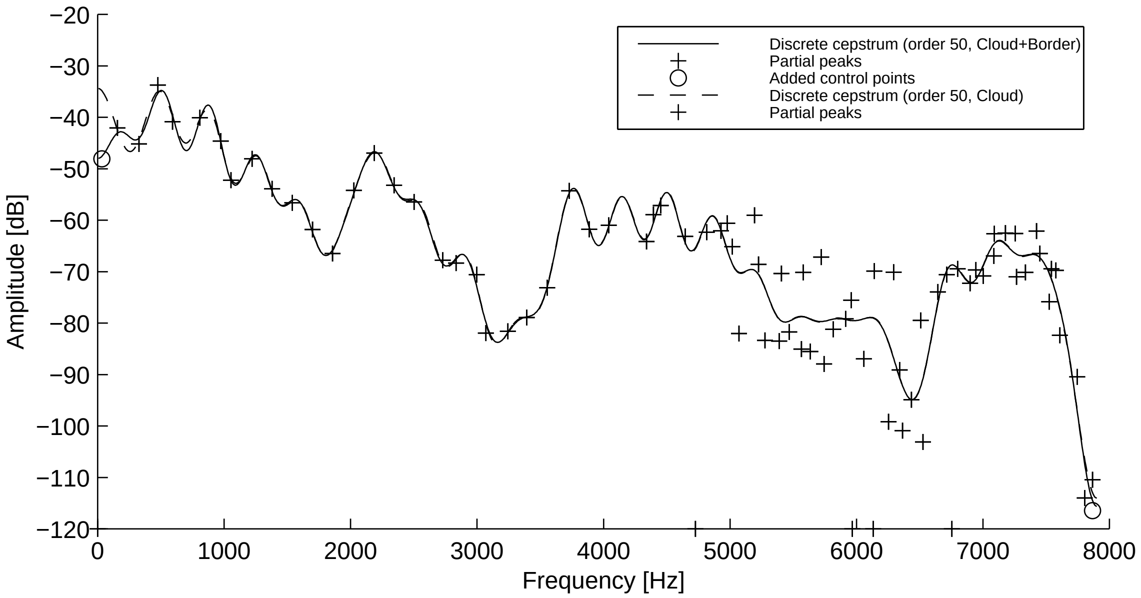

Figure 3.12: Effect of adding border points to control the spectral envelope

The **border points** are added at the frequencies m and at $(f_s/2 - m)$ with half the amplitude of the lowest/highest partial, respectively. This will force the spectral envelope to have a downward slope at the borders. Thus, if—by downward transposition of the partials while keeping the spectral envelope—there is a partial which is moved to lower frequencies, there is no risk that it will suddenly rise in amplitude, but it will be faded out smoothly. Figure 3.12 shows the effect of adding border points.

However, if the frequency of the lowest partial is less than m, the low border point is not added, since this would obviously cause an unjustified dip in the spectral envelope, which would disturb the smoothness. We don't have to worry about the border condition then anyway, since then the lowest partial would be very close to the 0 Hz border, constraining the spectral envelope enough to prevent it from rising. The same holds analogously for the high border point.

The **filling points** fill up a possible gap between the frequency of the highest partial and $f_s/2$ with very low amplitude peaks, spaced at $f_s/2n$ (n being the number of points of the envelope requested). This is to avoid too much freedom for the spectral envelope. If the spectral envelope was unconstrained in a large frequency range, it would oscillate wildly.

3.6 Evaluation of Discrete Cepstrum Estimation

The evaluation of discrete cepstrum spectral envelope estimation described in section 3.4 serves to make sure that the developed algorithm performs well. Especially the property that the input points don't have to be regularly spaced can lead to large gaps and thus too much freedom for the envelope, such that there is a risk of bogus humps where the envelope rises several dB over the desired level. If such a hump occurs in one time frame only, it can be audible as an artefact (an impulse) in resynthesis.

To check that such humps don't occur with the developed algorithm, a large corpus of recordings of the singing voice has been evaluated. It comprises 42 minutes of recordings of a female coloratura soprano and a male counter tenor made for the film *Farinelli* [DGR94]. The corpus covers the upper frequency range, where the risk for humps is greatest.

The difficulty in evaluation is that the test if the estimated spectral envelope is well-behaved has to be automated to be feasible, but how can this property of being well-behaved be described to the automaton? If we knew a formal description of a well-behaved spectral envelope, we would immediately implement it as an estimation algorithm. We can, however, compare the spectral envelope to something that is not eligible to be a spectral envelope, but is close, and is guaranteed to produce no humps. This something is the curve obtained by **linear interpolation** of the points in the frequency–amplitude plane. It is not a good spectral envelope because it has corners.

The simplest approach is to take the vertical distance between the linear interpolation and the estimated spectral envelope, as shown in figure 3.13. The histogram of absolute differences, which shows how their number is distributed, and the point of maximum difference are recorded and can be examined later, to see how the deviation can be avoided.

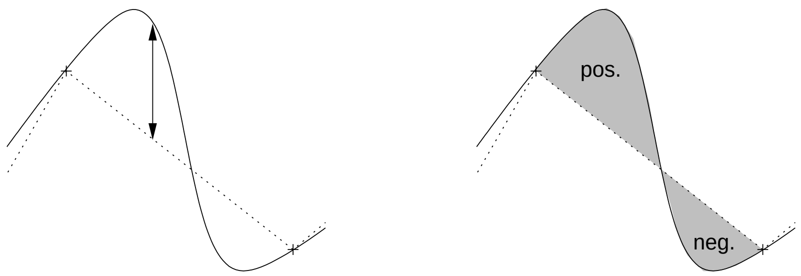

Figure 3.13: Principle of evaluation of the discrete cepstrum method by vertical deviation (left), and difference in area (right)

A more sophisticated approach takes the area between the linear interpolation and estimated spectral envelope, as demonstrated in figure 3.13 (right). Because, for the human hearing, upward deviations are very annoying, while downward deviations are hardly noticed at all, it would suffice to consider positive areas.

The evaluation showed some deviations first, which could be reduced by choosing a regularization factor of $\lambda = 0.0005$, after this, no humps in sensitive frequency ranges and at high amplitudes could be observed. Figure 3.14 shows such a case, where a deviation of over 16 dB occurs close to 8000 Hz, but it comes from a sensible interpolation of the algorithm between extremely placed partials at low amplitudes, and can thus be ignored.

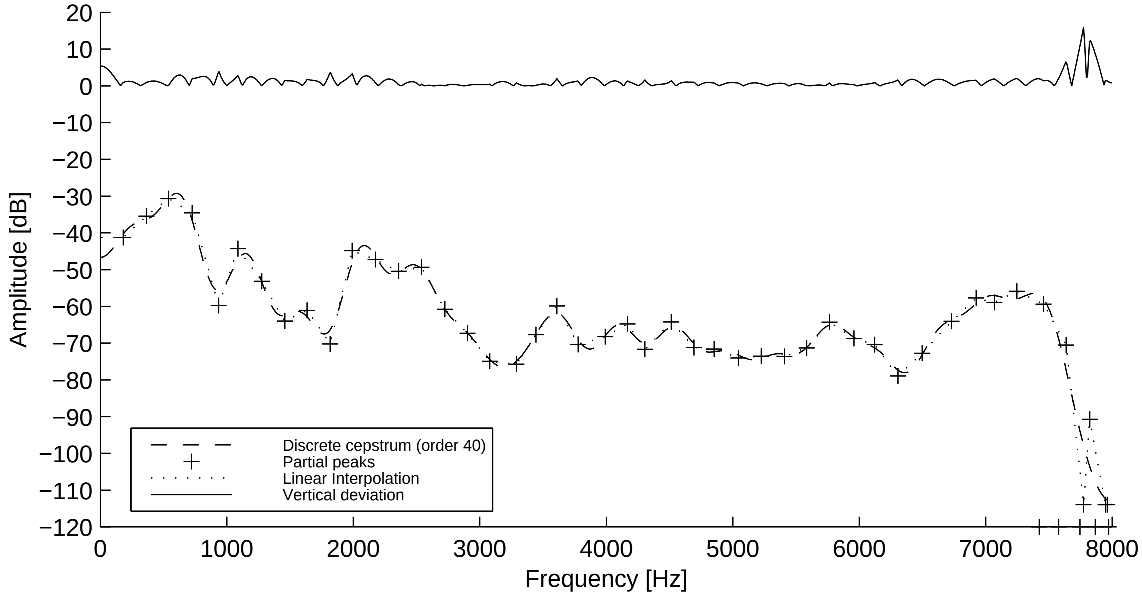

Figure 3.14: Example of evaluation of the discrete cepstrum method

Chapter 4

Representation of Spectral Envelopes

The representation of spectral envelopes is the pivoting point of the project. As we have seen in the previous chapter, the various estimation methods result in very different parameterizations of spectral envelopes. However, the choice of one canonical representation is essential for the flexibility of further processing.

Also, the choice of a good canonical representation of spectral envelopes is crucial for their applicability to a specific task. The ability to manipulate them in a useful and easy way, and the speed of synthesis depend heavily on the representation. Specifically, the requirements of locality, flexibility, speed of synthesis, and space will be laid out in section 4.1. Theu will then be tested with the different possible representations in the subsequent sections 4.2–4.6. Finally, section 4.7 will give a summary of the qualities of the examined methods, and present the representation which was chosen.

4.1 Requirements

Preciseness

Naturally, the representation has to describe the spectral envelope obtained from estimation as precisely as possible. Methods which don't fulfill this basic requirement have not been considered here.

Stability

The requirement of stability mandates that the representation be resilient to small changes in the data to be represented. Small changes, e.g. in the presence of noise, must not lead to big changes in the representation, but must result in equally small changes.

The requirement of stability is of great importance if we consider that the data to be represented can result from various different estimation methods, like cepstral or LPC analysis, or even from manual input, and that some noise is always present.

Locality

The locality requirement states that it be possible to achieve a local change of the spectral envelope, i.e. without affecting the intensity of frequencies further away from the point of manipulation.

Ideally, the representation would fulfill the requirement of **orthogonality**, where one component of the spectral envelope can be changed without affecting the others at all.

Flexibility and ease of manipulation
The representation must be chosen such that manipulations with an exactly defined desired outcome can be easily specified, e.g. a certain formant location that has to be reached in voice synthesis. For the manipulative abilities to be really useful for musical applications, the relationship between the parameters of the manipulation and the effect on the spectrum has to be easily understandable.

Speed of synthesis
The representation should be usable for sound synthesis as directly as possible, without first having to be converted to a different form at high computational costs. This requirement is heavily dependent on the type of synthesis, e.g. additive synthesis or filtering. This means that no ideal solution can be presented, but a compromise which is not the fastest choice for each synthesis type, but which doesn't penalize too much, even in the worst case.

Space
It is required that the representation not take up too much space, especially with the file representation in mind.

Manual input
Finally, the representation should be easy to specify manually, e.g. by drawing a curve or placing primitive shapes, or by textual input of parameters.

4.2 Filter Coefficients

The most straightforward representation of spectral envelopes are the filter coefficients, which are the output of the estimation, be it the cepstral coefficients c_i (section 3.3 ff.), or one of the LPC coefficients a_i, k_i (section 3.2). They are the most precise representation possible.

They are stable, but not local. The non-locality is due to the fact that they essentially represent a spectral envelope in time-domain (by one of different possible filter models), so changing one coefficient will change the envelope's value at multiple frequencies.

Because the changes one wishes to effect will be specified in the frequency-domain, they are also not easy to manipulate.

They do certainly fulfill the space requirement (only order p values), but are costly in evaluation for additive synthesis, since p cosines have to be evaluated for each frequency at which we desire to know the amplitude of the envelope. For subtractive synthesis, however, they are efficient, since they can directly be used for fast time-domain filtering.

They can not at all be specified manually.

4.3 Spectral Representation

The spectral representation is the natural way a spectral envelope was presented in the previous chapters: as an amplitude curve in the frequency-domain, based on an equidistant or logarithmically spaced frequency grid. Each of the n grid point is also called a frequency bin. This representation is also called sampled spectral envelope representation, because we take a sample of the value of the (theoretically) continuous curve at each point of the frequency grid.

This representation is as stable as the filter coefficients, because it is derived directly from them. It is local, and when the frequency scale is linear, it is orthogonal, because the amplitude at each frequency can be changed independently from the others.

It is the most flexible representation (due to the high locality), but not that easy to manipulate, because the locality demands that all the values at all the frequencies be given. Especially when we think of an application for the singing voice, the preferred manipulations are changes of the position and bandwidth of formants, which means that new amplitude values would have to be specified for the whole frequency range of the spectral envelope the formant occupies.

They are fastest for additive synthesis and reasonably fast for filtering. They are reasonably compact, since the space required can be as low as 100 points, even less when a logarithmic frequency scale is used.[1] Manual input is easy.

4.4 Geometric Representation

Starting from a spectral representation, a geometrical representation can be derived, which tries to describe the amplitude curve of the spectral envelope in the frequency-domain with fewer points not spaced at equidistant frequencies. The geometrical representation can be of the form of a break-point function or splines, as described in the following.

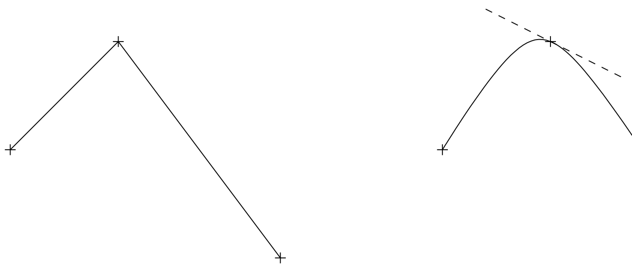

Figure 4.1: Geometric representations of spectral envelopes: as a break point function (left); as splines with dashed continuous slope (right)

Break point functions

A break point function (BPF) is a general method of representing a function, be it in time- or frequency-domain. It consists of n **break points** P_i at (x_i, y_i). In our case, x_i is the frequency and y_i the amplitude. The $n-1$ segments between the break points are interpolated linearly, as in figure 4.1, left.

Splines

Splines [UAE93] are similar to break point functions, but they provide for quadratic or cubic interpolation of each section between the points P_i given, using a polynomial of degree 2 or 3. Additionally, continuity constraints at the given points stipulate that the slope be identical. This slope (the dashed line in figure 4.1, right) can also be accessible as a parameter for manipulation.

Splines would be applied to spectral envelope representation in a way that the points P_i would be placed on the maxima and minima of the spectral representation, where the slope is zero, and on the turning points, where the curvature changes direction.

A weak general point of geometric representations is that they don't model the spectral envelope in a way relevant to its properties as a signal, but simply as a curve in euclidian space. Especially interdependencies between the given points, that arise from the signal character of the spectral envelope are not taken into account automatically.[2]

The stability of geometric representations is seriously disturbed by the fact that small changes can cause a sudden change of the maxima found. They are quite local, and can be made more so by adding points manually. They are flexible and easy to manipulate and will always give nice smooth curves. However, there is a tradeoff between ease of manipulation and preciseness: If there is a point

[1] A word of warning, though: When choosing too small a number n of points for the envelope in relation to the highest frequency of half the sampling rate, severe aliasing of the spectral envelope can result. This is the same effect as sampling a signal with a sampling rate lower than twice the highest frequency in the signal. Here, the signal is the spectral envelope to be represented, while the number of points corresponds to the sampling frequency (their distance is the sampling period). Of course, this problem is greatly alleviated using a logarithmic frequency scale also for representation.

[2] Nevertheless, splines can be of some use for spectral envelope *estimation* when they are used to pre-smooth the spectrum before an estimation method is applied to find the parameters of a filter model. In [TAW97], the authors present how this can be applied to low-order LPC-analysis.

that governs a large area that can thus be manipulated easily, the preciseness can suffer because a large stretch of the curve will be interpolated.

Regarding speed, the geometric representations are slightly more costly in synthesis than spectral representation. For splines, the evaluation of the interpolating polynomials have to be taken into account. The space needed is less than for spectral representation, even more so if pruning of redundant points can be applied (again at the cost of preciseness).

For specifying spectral envelopes manually by drawing, geometric representations are very well suited.

4.5 Formants

In a formant representation, a spectral envelope is composed of a parametric description of formants (the resonances of the vocal tract or of other acoustic resonator—see section 2.4) and a residual envelope. Three ways to represent formants will be presented: FOFs, standard formants, and fuzzy formants (cf. figure 4.2).

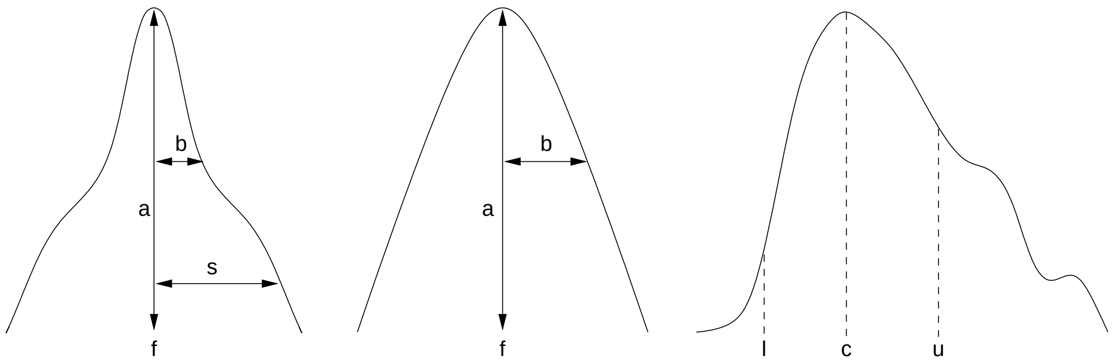

Figure 4.2: A FOF (left), a precise formant (middle), and a fuzzy formant, which is simply a frequency region in a spectral envelope (right), with their frequency-domain parameters

Formant-wave functions

A **FOF**, from the french *Forme d'onde formantique* [Rod84], is originally a method of high quality voice synthesis and sound synthesis in general. It forms the basic synthesis model of the CHANT system (section 2.5). A FOF is a time-domain representation of a single formant as a basic waveform, several of which are added to build up the desired spectrum (typically 5–7). A FOF is parameterized both in terms of the frequency-domain and of the time-domain. The parameters, in fact, specify the spectral envelope of one formant.

The frequency-domain parameters of a FOF are center frequency f, amplitude a, bandwidth b, and skirt width s, which can be controlled independently from the bandwidth; the time-domain parameters are phase ϕ and excitation and attenuation times. It can be seen, that this is much more information than is needed for a description of a spectral envelope.

Precise formants

A more economical way to describe formants is to use the standard parameters of a resonance: center frequency f, amplitude a, and bandwidth b. The bandwidth specifies half of the width of the formant at 3 dB down from the peak. Then the spectral envelope $v(\omega)$ of one formant is of the form (with an appropriate scaling of the parameters):

$$v(\omega) = e^{-\left(\frac{\omega - c}{b}\right)^2} \tag{4.1}$$

Fuzzy formants

As an augmentation of the spectral representation (section 4.3), I define fuzzy formants as a formant region within a spectral envelope where it is believed or known that a formant lies.

With labeled source material (a recording of the voice with annotations what speech sounds (phonemes) are spoken or sung), the positions of the formants in vowels are fairly well known.

A fuzzy formant is specified by three frequency parameters, the lower bound l, the upper bound u, and the center c, if known. Additionally, a bookkeeping parameter gives an identification to each formant, such that they can be associated into **formant tracks**.

With a formant representation, the general problems of finding and identifying formants exist. For unlabeled data, the identification which hump in the spectral envelope is really a formant, and if it's the first, second, etc, is far from being trivial.

The formant representation is not stable, since a slight ditch in the spectral envelope could suddenly create a new formant. They are local, however, and flexibly and very easily manipulable. Synthesis is reasonably fast, both for the frequency-domain and for the time-domain. They are very compact in storage, if a pure formantic representation is sufficient (or the loss in preciseness is bearable), but for most cases they would need a residual spectral representation to be stored along with them.

For specifying spectral envelopes manually, especially for the precise synthesis of the voice, formant representations are best suited.

4.6 High Resolution Matching Pursuit

High Resolution Matching Pursuit (HRMP) [GBM+96, GDR+96] is a method to decompose a sound into a time–frequency representation that keeps both the exact location in time of the fast transients (the attacks), and a good frequency resolution in the sustained parts. This matches the perceptive capabilities of the ear. Figure 4.3 shows an example.

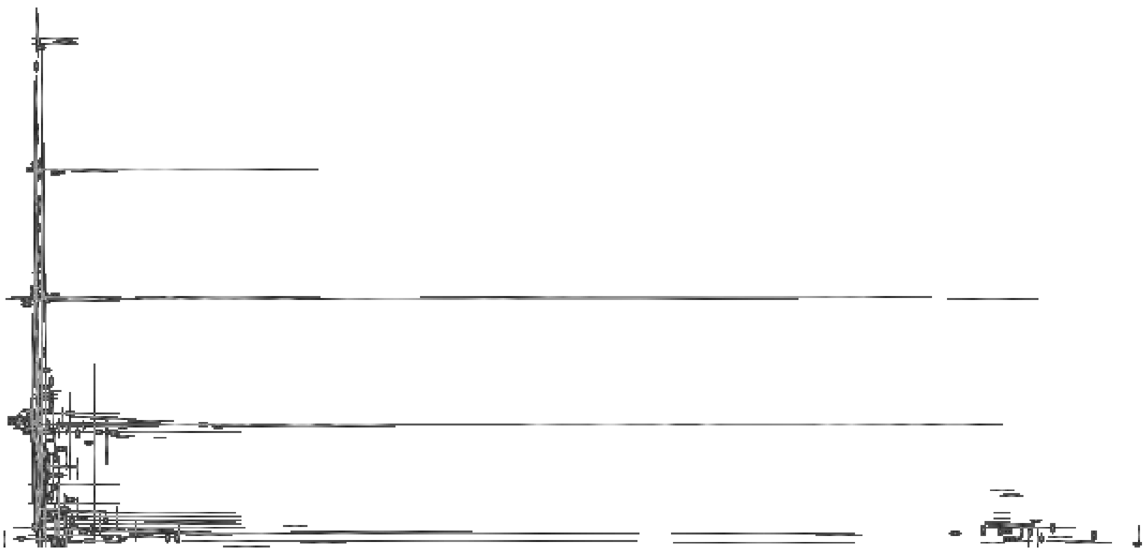

Figure 4.3: Time–frequency decomposition of a piano note with HRMP from [GBM+96]. It can be seen that the attack as well as the sustained harmonics are precisely localized. The structure at the end is the release of the key, causing the damper to touch the piano cord.

Since it is a time–frequency analysis, HRMP is not appropriate as such for spectral envelope representation, which pertains to one instance in time only, but it is an interesting method worth a look. The idea is to use a dictionary of primitive time–frequency components, called atoms, and to select the ones that together yield the closest match with the signal.

To apply HRMP to spectral envelope representation, it would be restricted to the frequency domain. The dictionary would contain curve pieces, especially formant shapes. Unfortunately this method would not be stable, since very different atoms could be selected for small changes. It would be local, because the atoms would have a small frequency support, and very flexible to manipulate.

The easyness would depend on the dictionary used, as would the speed of synthesis and the space. (If the structures get too complex, significant redundancy could occur.)

HRMP structures would be very complicated to specify manually.

4.7 Summary and Chosen Methods

Table 4.1 shows a comparison of the representations introduced in the preceding sections with a score (++, + , o , − , −−) indicating the degree to which they fulfill the requirements for representations from section 4.1.

The methods chosen to be implemented are filter coefficients, the spectral representation, precise and fuzzy formants. The filter coefficients are kept because they provide the most precise and a very compact representation, they are closest to the source and can be easily converted to the canonical representation. The spectral representation is the most general, and will be the canonical representation. The precise formants are necessary for high quality singing voice synthesis. The fuzzy formants, along with the spectral representation, will make it possible to interpolate by formant shift between spectral envelope in spectral representation and precise formant representation, as will be explained in section 5.1.2.

Representation	Stability	Locality	Flexibility/Ease of Manipulation	Speed of Synthesis time-/freq-domain	Space	Manual Input
Filter Coefficients	++	−	−−/ −	++/ −	+	−−
Spectral	++	++	++/ o	+ / ++	o	+
Geometrical	−	+	+ / ++	o / −	+	++
Formants	−	+	++/ ++	+ / +	++	++
HRMP	−−	+	++/ o	o / o	o	−−

Table 4.1: Comparison and score of representation methods in regard of requirements

Chapter 5

Manipulation of Spectral Envelopes

This chapter describes the different types of manipulation that are possible with spectral envelopes. If not said otherwise, it is assumed that the spectral envelopes are in the form of the canonical spectral representation described in section 4.3.

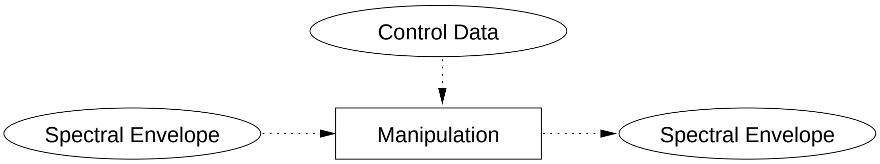

Figure 5.1: General manipulation of spectral envelopes

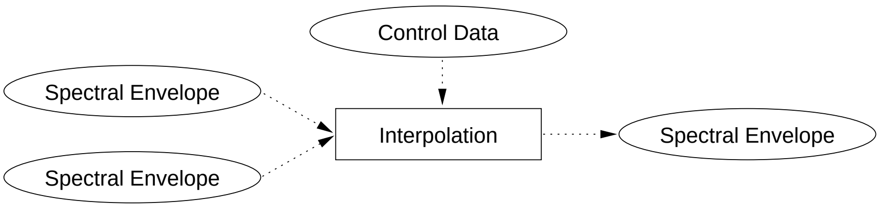

Figure 5.2: Interpolation, the most important type of manipulation of spectral envelopes. The resulting spectral envelope will be a weighted average of the two input spectral envelopes (termed the original spectral envelope and the target spectral envelope).

Manipulation is at the heart of the creative process. It allows composers and musicians to surpass what is given in recorded sounds, either to create sounds which are far from the original, or to subtly modify given sounds, or to mix characteristics of different sounds. By manipulation, a spectral envelope is changed according to some parameters, as depicted in figure 5.1. The most important type of manipulation, interpolation between spectral envelopes, is described in section 5.1, manipulations changing the amplitude of a spectral envelope are covered in section 5.2, other types of manipulations in section 5.3.

5.1 Interpolation

First of all it must be noted that there are two different meanings of the word interpolation. One meaning refers to finding a value of a function that is given only at discrete points, when the value is inbetween two of the given points. With spectral envelopes, we use interpolation in this sense when we want to know the value of the envelope $v(f)$ at an arbitrary frequency f, which is not one of the given points of the envelope.[1] If f_l and f_r are the two points closest to f, then the **linear interpolation** is:

$$v(f) = \frac{v(f_r) - v(f_l)}{f_r - f_l}(f - f_l) + v(f_l) \tag{5.1}$$

which is the value of $v(f)$ if the given points of the envelope are considered to be connected by line segments (see figure 5.3).

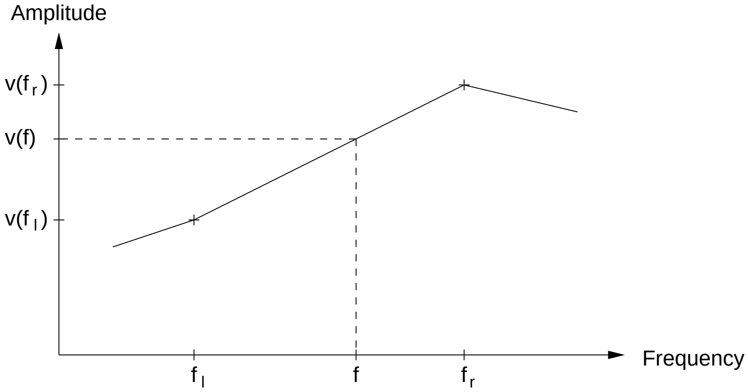

Figure 5.3: Linear interpolation within a spectral envelope

5.1.1 Interpolation between Spectral Envelopes

The second meaning of interpolation is finding an intermediate state in the gradual transition from one parameter set to another, in our case going from one spectral envelope to another. This **interpolation between envelopes** is in fact a **weighted sum** of the spectral envelopes. It can always be reduced to the first sense of interpolation, in that we take the linearly interpolated values $v_1(f)$ and $v_2(f)$ for each frequency f of two spectral envelopes and interpolate between them by an **interpolation factor** m. If $m = 0$ we will keep the original spectral envelope v_1, if $m = 1$ we will receive the target spectral envelope v_2:

$$v_m(f) = (1 - m)v_1(f) + mv_2(f) \tag{5.2}$$

If we consider that the envelopes can be sampled each at a different set of discrete frequency points f_1 and f_2, to actually compute the interpolated spectral envelope we have to build the union of the frequency points $f_u = f_1 \cup f_2$ and compute the interpolated value at each point of f_u.

5.1.2 Shifting Formants

When dealing with the spectral envelope of speech or the singing voice, we want to respect the formant structure of the envelope. This means that if we want to interpolate between two spectral envelopes, we don't want the amplitudes at each frequency interpolated as in equation (5.2), but shift the formants from their place in the original spectral envelope to that in the target spectral envelope. In fact, we want to simulate the effect of interpolating the articulatory parameters of the vocal tract. Figure 5.4 explains the different approaches.

The prerequisites for shifting formants in this way are of course that we know where the formants are located, and which formant in the original spectral envelope is associated with which formant in the target spectral envelope. The former is not at all obvious and is a question of formant detection. The latter is even more difficult for an automated method without providing manual input. It is a question of labeling the formants of successive time frames to generate formant tracks.

[1] Of course, a spectral envelope is only defined between 0 Hz and half the sampling frequency.

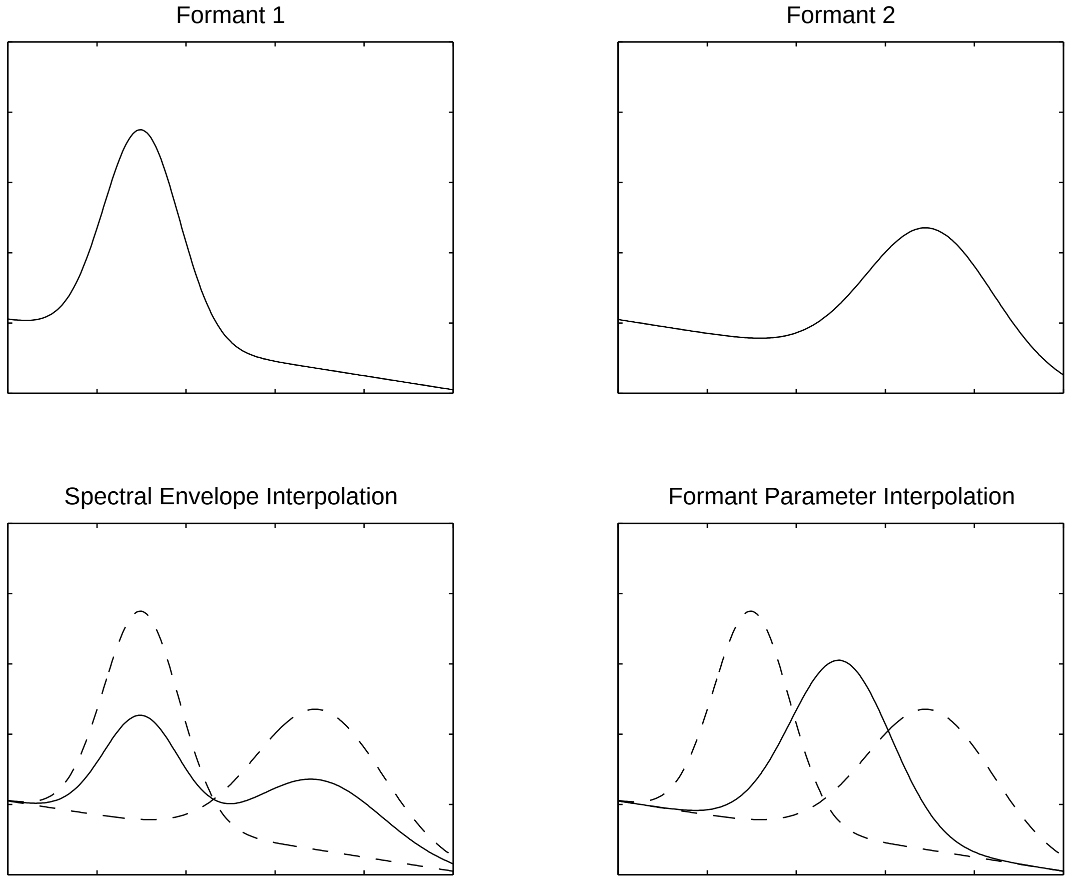

Figure 5.4: Formant interpolation versus formant shift: The upper row shows two spectral envelopes whose single formants are to be interpolated. In the lower left figure we see what we get from the direct interpolation of spectral envelopes—which is a weighted sum of the two curves, here with an interpolation factor of 0.5. The original formants are shown as dashed curves. The lower right figure shows what we really want: interpolating the parameters of the formants, resulting in a frequency shift.

Fortunately, for some applications, we know *a priori* where the formants *should be*. For example, when treating the voice in a piece where the lyrics are known, like an opera. Then it is known which vowels are sung, and thus we can look up the formant positions in the formant tables from phonetics literature. In this case, the spectral envelope representation would be augmented by fuzzy formants, or a spectral envelope representation using exact formants will be provided, as described in section 4.5.

5.1.3 Shifting Fuzzy Formants

The fuzzy formant representation of spectral envelope consists of an envelope in spectral representation plus several formant regions with an index for identification. Given two spectral envelopes with two fuzzy formants with the same index, it is still not clear how the intermediate spectral envelopes, with the formant on its way from its position in the original envelope to that in the target envelope, are to be generated. Several questions arise: How to fill the hole the formant leaves when it starts to move away? What to do with the envelope in the places the formant moves over? How should the shape of the formant change between the original and the target shapes?

After an idea by Miller Puckette, it is possible to interpolate an envelope in spectral representation in a way that formants are shifted exactly as we want. The idea is to first integrate over the envelopes (in the discrete case, this amounts to building the cumulative sum of the spectral envelope), and then to horizontally interpolate between the integrals. We retrieve the interpolated formant by subsequent differentiation of the result.

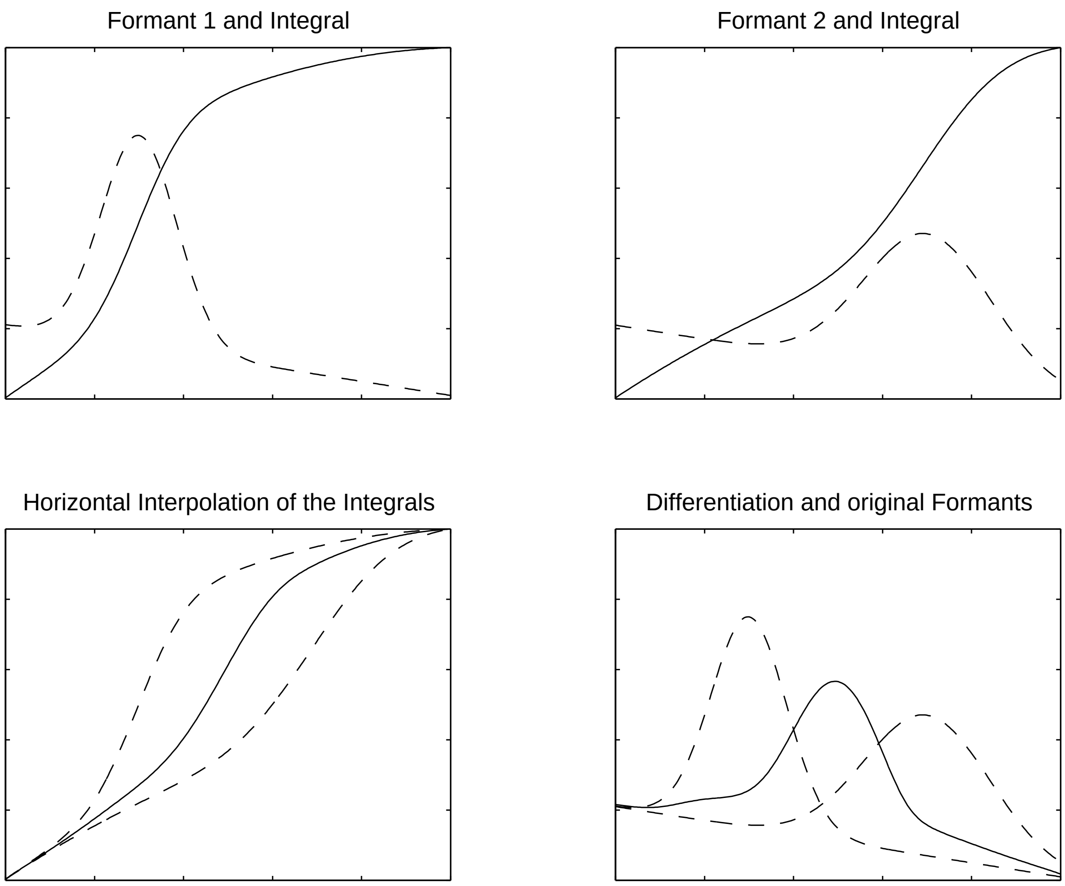

Figure 5.5: Interpolation of formants by horizontal interpolation of the integral. All amplitudes are normalized. The upper row shows the integrals (the cumulative sum) of the two formants shown as dashed spectral envelopes. The lower left figure shows the horizontal linear interpolation by a factor of 0.5 of the interpolated integrals, drawn again as dashed lines. Taking the derivative of the result in the lower right figure reveals the almost perfectly shifted formant. (The original formants are shown again in dashed lines for clarity.)

That the idea works can be seen in figure 5.5. Formally, the method can be described like this: Starting from two spectral envelopes $v_1(f)$ and $v_2(f)$, considered as continuous functions over $0 \leq f \leq f_s/2$, we construct the cumulative integral functions $V_1(F)$ and $V_2(F)$, and normalize:

$$V_i(F) = \frac{1}{s_i} \int_0^F v_i \, \mathrm{d}f \qquad \text{for } i = 1, 2 \tag{5.3}$$

With

$$s_i = \int_0^{\frac{f_s}{2}} v_i \, \mathrm{d}f \tag{5.4}$$

being the maximum values of $V_i(F)$. Then we invert these functions

$$V_i(F) = A \iff V_i^{-1}(A) = F \qquad \text{for } i = 1, 2 \text{ and } 0 \leq A \leq 1 \tag{5.5}$$

and interpolate by weighted sum with an interpolation factor m to receive $\bar{V}^{-1}(A)$:

$$\bar{V}^{-1}(A) = (1 - m)V_1^{-1}(A) + mV_2^{-1}(A) \tag{5.6}$$

After inverting $\bar{V}^{-1}(A)$ back to normal coordinates

$$\bar{V}^{-1}(A) = F \iff \bar{V}(F) = A \tag{5.7}$$

we differentiate to retrieve the interpolated spectral envelope $\bar{v}(f)$:

$$\bar{v}(f) = \frac{\mathrm{d}\bar{V}}{\mathrm{d}f} \tag{5.8}$$

Unfortunately, this works only for one formant to be interpolated, as can be seen in figure 5.6. Nevertheless, we can do better if we have the information of formant regions, i.e. we know where the two formants to be interpolated lie in their respective spectral envelopes. In this case, we can restrict the technique of horizontal interpolation of the integral to the given formant regions, with an appropriate fade-in and fade-out of the region borders.

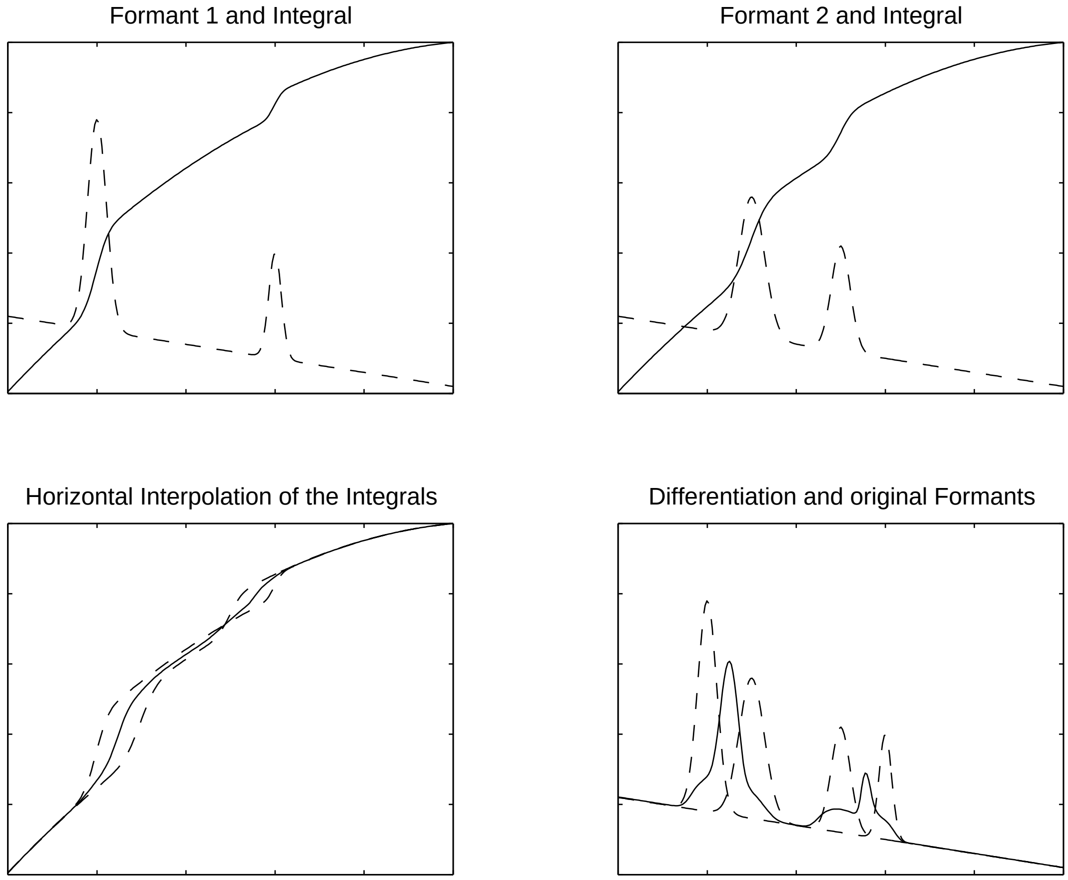

Figure 5.6: Interpolation of two formants by horizontal interpolation of the integral. Obviously, the result in the lower right graph does not correspond to the interpolation of the formant parameters.

5.1.4 Interpolation between Precise Formants

If both of the two spectral envelopes to be interpolated are given as precise formants with their index i, center frequency f_i, amplitude a_i, and bandwidth b_i as parameters, interpolation becomes trivial. Simply the formant parameters need to be linearly interpolated, using equation (5.1) accordingly.

5.1.5 Summary of Formant Interpolation

Summing up the different possibilities of interpolation of spectral envelopes, we can recognize a hierarchy in the spectral envelope representations in regard of formant interpolation. The hierarchy is, from highest to lowest:

1. Precise formants: can be interpolated perfectly
2. Fuzzy formants: can be interpolated reasonably well
3. Spectral representation of envelopes: can not be interpolated respecting formants

With each step down we lose some information necessary for formant interpolation. We can convert downwards step by step:

1. $\rightarrow$ 2. Convert precise formants to fuzzy formants by evaluating the spectral envelope given by the precise formants after equation (4.1), and storing it in spectral representation, while calculating the formant regions from the precise formant parameters.
2. $\rightarrow$ 3. Convert fuzzy formants to spectral representation by simply discarding the formant regions.

We cannot, however, convert upwards, because that would mean adding information (by simple calculations, that is—of course, methods to detect formant shapes in spectral envelopes exist, but these are the subject of a field of research of its own.)

This means that, when spectral envelopes in different representations have to be interpolated, we can't do better than going down to that representation of the two which is lowest in level, discarding the formant information of the higher one.

5.2 Amplitude Manipulations

An obvious type of manipulation of spectral envelopes is **amplitude manipulation**. To describe that whole class of manipulations, we only need to define the two basic operations of addition and multiplication of spectral envelopes. This is straightforward if we formally define a spectral envelope as a real function $v(f)$ with domain $0 \leq f \leq f_s/2$:

$$v_{add} = v_1 + v_2 \iff v_{add}(f) = v_1(f) + v_2(f) \quad \text{for all } 0 \leq f \leq f_s/2 \tag{5.9}$$

$$v_{mul} = v_1 \cdot v_2 \iff v_{mul}(f) = v_1(f) \cdot v_2(f) \quad \text{for all } 0 \leq f \leq f_s/2 \tag{5.10}$$

If the spectral envelope is in spectral representation (section 4.3), we get to the functional interpretation by linear interpolation as in equation (5.1). If it is in precise formant representation, we can evaluate the formant function (equation (4.1)) directly, if it is in filter coefficients, we convert it to spectral representation.

With these operations, we can effect manipulations such as amplification and attenuation, or spectral tilting, as explained in the following.

Amplification and attenuation

Amplification and attenuation of spectral envelopes is effected by a multiplication by a factor greater or less than one, respectively. Multiplication with a function over frequency (i.e. with a spectral envelope) instead of a constant factor, thus doing **frequency selective amplification or attenuation**, essentially amounts to applying a filter to the spectral envelope.

Tilting the spectrum

The overall slope of the spectrum of a speech or instrument signal is called **spectral tilt**. For speech, it is among others responsible for the prosodic feature of *accent*, in that a speaker modifies the tilt (raising the slope) of the spectrum of a vowel, to put stress on a syllable [Dog95]. For instruments, it can be dependent on dynamics, the relative force with which the instrument is played. Spectral tilt is measured in decibel per octave.

To apply a given spectral tilt t to a spectral envelope v, we have to first normalize the tilt per octave to a tilt t_m over the whole range of the spectral envelope up to $f_m = f_s/2$ (half the sampling rate):

$$t_m = t \frac{f_m}{\log_2 f_m} \tag{5.11}$$

Then, the additive decibel value t_m is converted to a multiplicative amplitude factor t_a:

$$t_a = 10^{\frac{t_m}{20}} \tag{5.12}$$

To effect the tilting of a spectral envelope $v(f)$, we multiply by a ramp spectral envelope $v_t(f)$ ramping from 0 to t_a:

$$\mathrm{tilt}_t(v) = v \cdot v_t \qquad \text{with} \tag{5.13}$$

$$v_t(f) = f \frac{t_a}{f_m} \tag{5.14}$$

5.3 Other Manipulations

The manipulations not covered by the preceding sections concern mainly manipulations of the frequency location of parts of a spectral envelope. One possibility I named **skewing** is to selectively displace a part of the spectral envelope in frequency. In order to avoid holes, the amount of displacement (the frequency shift) is faded in and out by a skew-function, as in figure 5.7.

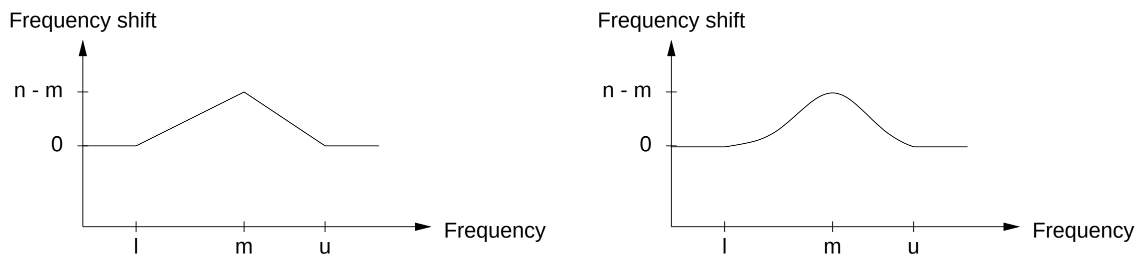

Figure 5.7: The left figure shows the linear skew function, determining the frequency shift dependent on frequency, while to the right, a smoother gaussian skewing function is shown.

Four parameters specify the skewing: The lower and upper frequencies l and u determine the range of the spectral envelope to be affected. The middle frequency m will be moved to the new frequency n, while the spectral envelope left and right from m will be shifted gradually less the further the distance to m. The result of skewing with a linear skew function is shown in figure 5.8. Many other skew functions and more parameters are possible.

Other possible frequency manipulations are a shift of the whole spectral envelope$v$, as well as **compression** or **dilatation** by a transposition factor t:

$$v_{transp}(f) = v\left(\frac{f}{t}\right) \qquad \text{for all } 0 \leq f \leq f_s/2 \tag{5.15}$$

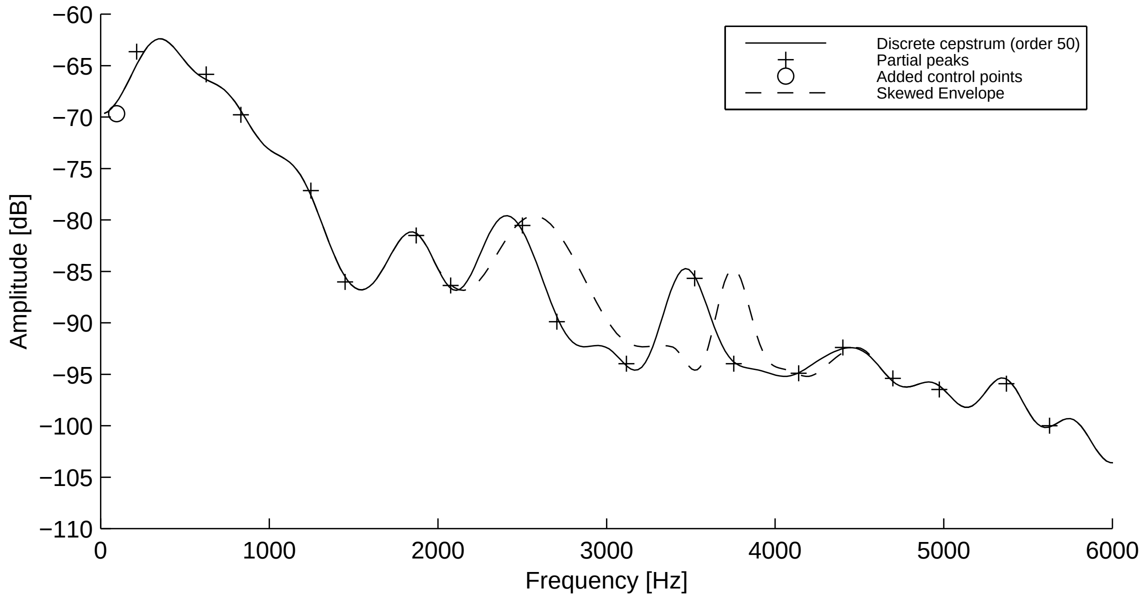

Figure 5.8: Manipulation by Skewing. The spectral envelope (of a voice) is skewed (displaced in frequency) between $l = 2000$ and $u = 4600$ Hz, with the mid-point of skewing displaced from $m = 3000$ to $n = 3400$ Hz. Outside the range from l to u, the spectral envelope is left untouched.

Chapter 6

Sound Synthesis with Spectral Envelopes

In the application of spectral envelopes to sound synthesis, a spectral envelope called the **modulator** is applied to an input sound to yield an output sound, according to some parameters, as shown in figure 6.1. If the input and output sound are represented as sinusoidal partials, then additive synthesis is to be applied as described in section 6.1. If the sounds are given as a sampled signal, filtering has to be applied, as explained in section 6.2.

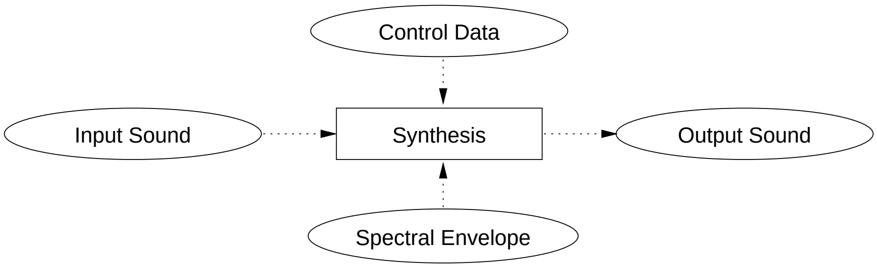

Figure 6.1: Synthesis with a modifying spectral envelope called the modulator

6.1 Additive Synthesis

In the domain of additive synthesis (see section 2.2), application of spectral envelopes consists of a simple change of the amplitude of the input partials, while leaving their frequencies untouched. The simplest way is to replace the amplitude of a partial at frequency f with the value of the envelope $v(f)$ at that frequency. If the input partials and the modulating spectral envelope are from different sounds, this is called *cross synthesis*, since it crosses the characteristics of two distinct sounds: the partial structure (presence and frequency location, or absence of partials and their development in time) of the input sound, and the modulating spectral envelope estimated from the other sound.

Instead of imposing the spectral envelope, a gradual mixture between the spectral envelope of the input sound (given by the partial amplitudes), and another spectral envelope is possible. This is governed by a mix factor m, between 0 for the original and 1 for the imposed spectral envelope. If a_i is the amplitude of partial i at frequency f_i, and $v(f_i)$ the value of the envelope at that frequency, then the new amplitude a_i' is

$$a_i' = (1 - m)a_i + mv(f_i) \tag{6.1}$$

One step further to flexibility is taken by considering the mix factor being a function $m(f)$ dependent of frequency. This way, e.g. only the high frequency part of the input sound can be changed to adopt the spectral envelope characteristics of the modulator.

Note that this can be generalized to n spectral envelopes $v_j(f)$ and n mix functions $m_j(f)$ with their sum $m_s(f) = \sum_{j=1}^{n} m_j(f) \leq 1$, such that the proportion of the original is $1 - m_s(f_i)$:

$$a'_i = (1 - m_s(f_i))a_i + \sum_{j=1}^{n} m_j(f_i)v_j(f_i) \tag{6.2}$$

If the envelope isn't represented directly but as filter coefficients, these have to be converted to the spectral envelope first, as shown in chapter 3 in equations (3.3) and (3.10) for LPC and cepstrum coefficients, respectively.

6.2 Filtering

If the sound to be changed is not represented by partials but comes as a time-domain signal, the modulating spectral envelope has to be converted to a filter that is to be applied to the signal. There are always two possibilities how to actually carry out the filtering:

- In the time-domain, the signal x is convolved with the impulse response h of the filter to yield the changed signal y:

  $$y(n) = x(n) * h(n) \tag{6.3}$$

- In the frequency-domain, the Fourier transform X of the signal x is multiplied with the transfer function H of the filter to yield the Fourier transform Y of the changed signal:

  $$Y(f) = X(f) \cdot H(f) \tag{6.4}$$

  Y will be converted to a time-domain signal y by inverse Fourier transform.

  In fact, the filter transfer function H is exactly the curve of the spectral envelope in the frequency–amplitude plane.

How to calculate either one of the forms of the filter will be described in the following, for each possible representation of the spectral envelope.

Spectral representation

If the spectral envelope is in canonical representation directly as a curve in the frequency–amplitude domain, the frequency-domain filter is given straightforward by the evaluation of the spectral envelope $v(f)$ at the frequencies f_i, which are the frequency bins used for filtering::

$$H(f_i) = v(f_i) \tag{6.5}$$

The time-domain filter is the inverse Fourier transform of the frequency-domain filter:

$$h = F^{-1}(H) \tag{6.6}$$

This time-domain filter will in general be more costly to apply than a time-domain filter derived from filter coefficients as shown below, since the impulse response can be as long as the number of bins in H.

Cepstral coefficients

If the spectral envelope is represented as cepstral coefficients c_i, for frequency-domain filtering, we have to reconstruct the spectral envelope, which defines the transfer function of H, by inversing the operations carried out in cepstrum estimation (section 3.3):

$$H = exp(F^{-1}(c_i)) \tag{6.7}$$

Which is equivalent to an evaluation of this formula at the desired frequency bins ω_j:[1]

$$H(\omega_j) = exp\left(\sum_{i=1}^{p} c_i \cos i\omega_j\right) \tag{6.8}$$

For time-domain filtering, the impulse response h is computed by the inverse Fourier transform of the transfer function H:

$$h = F^{-1}(H) = F^{-1}(exp(F^{-1}(c_i))) \tag{6.9}$$

LPC-coefficients

The predictor coefficients $a_i, i = 1..p$ are well suited to frequency-domain filtering. They describe the analysis filter A, which is inverse of the transfer function H, corrected by a gain factor g:

$$H(\omega_i) = \frac{g}{A(\omega_i)} = \frac{g}{1 - \sum_{i=1}^{p} a_i e^{-i\omega_i}} \tag{6.10}$$

The time-domain filter could be derived as usual by equation (6.6), however, it follows from the properties of LPC-estimation (section 3.2) that we can use the predictor coefficients directly:

$$y(n) = gx(n) + \sum_{i=1}^{p} a_i y(n-i) \tag{6.11}$$

What's more, by using the reflection coefficients k_i, we can apply the much more efficient filter structure of a **lattice filter**, and can implement the filter directly as shown in figure 6.2.

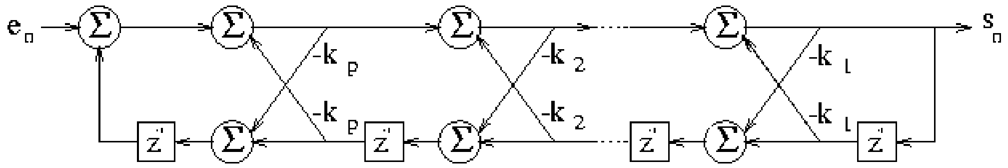

Figure 6.2: Lattice time-domain filter structure (from [Rob98]). The samples of the entry signal e_n pass via a network of summation stages ($\sum$), multiplication by the reflection coefficients k_i, and delay units z^{-1} to yield the filtered signal s_n

[1] In general, the Fourier spectrum is complex. The cepstrum, however, is evaluated from the magnitude spectrum which is the absolute value of the spectrum. Therefore, the recreation of the spectrum has no imaginary part.

Chapter 7

Implementation

This chapter describes the implementation of the methods for spectral envelope estimation, representation, manipulation, and application to synthesis documented in the previous chapters. The implementation proceeded in two steps: First, the spectral envelope and signal viewing program ViewEnv was developed in the Matlab programming language. It allows for prototyping, interactive testing, and evaluation of the algorithms for spectral envelope estimation and manipulation. The architecture and the usage of the ViewEnv program is described in appendix A.

Then, the spectral envelope function library was developed which combines the methods evaluated in the project, and makes them accessible to other programs. The principles of software engineering were applied to the development process.

This chapter is organised as follows:

First, some basic concepts of software engineering are introduced in section 7.1, and the notation used to document the design of the library and also of the ViewEnv program is explained in section 7.2. Then, some general considerations for the design and the implementation of the library are layed out in section 7.3, followed by the documentation of the design itself in section 7.4. Finally, the new SDIF file types used for permanent storage of spectral envelopes is described in section 7.5.

7.1 Some Words about Software Engineering

The central task of software engineering is to understand the **software development process**, to describe it, and to develop means of supporting it. These means comprise methods, (specification) languages, and tools. Some of them support single phases (see below) while others support the whole process. However, the integration of these means is still partly unsolved.

The software development process covers the whole **software lifecycle**, comprising the time from the development of software over the successive maintenance up to the point where the software isn't used anymore.

One of the earliest **process models** for the software development process is the **waterfall model** [GJM91]. This model distinguishes the phases of analysis, design, implementation, testing, and maintenance. It has two important drawbacks which software engineers have been trying to improve since then. First, the model supports only a unidirectional view of the software development process, even though it has soon become clear that this is far from being realistic. To the contrary, the phases are not completed one after the other, but there are recursions from later phases back to earlier phases. This problem has been addressed by the **extended waterfall model** (figure 7.2) which uses loops back from a phase to the predecessor phase. In reality, however, there are recursions back from any phase to all of its predecessor phases.

A second drawback is the separation of a naturally continuous process in discrete phases. This separation, however, is rather to be seen as a logical one, which helps to organise the efforts, than as a suggestion to actually proceed with the development of software following these steps one after

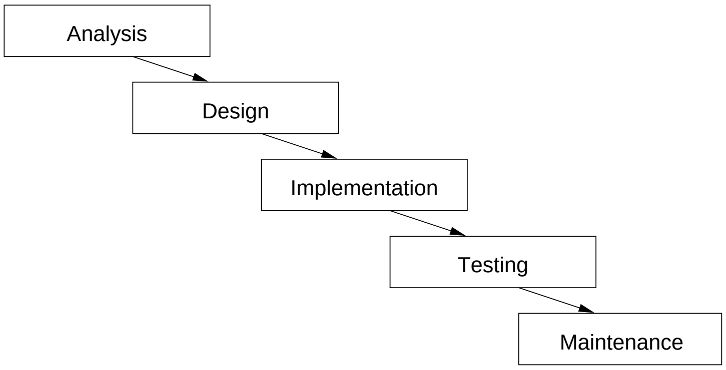

Figure 7.1: The classic waterfall model of software development

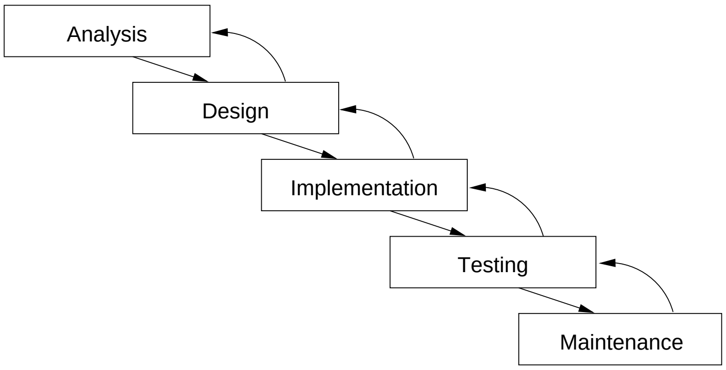

Figure 7.2: The extended waterfall model of software development with recursions to previous phases

the other. One possible attempt to tackle this second problem, choosing a finer granularity, will not overcome the main problem of discretization of a continuous process.

It is clear that, in order to overcome these drawbacks, an explicit modeling of directionality poses problems, and, if all possible directions are described, the model will become very tangled. There are approaches, however, which do try to avoid these problems. They do not emphasize the process itself, but they try to identify the working areas. That is, the phases which were related to a fixed point in time in the process are now discoupled from the time line and rather their intentional aspects are considered.

One of these approaches, the **working area model** [Nag90], identifies the areas of **requirements engineering** (RE), formerly analysis, **programming in the large** (PiL), formerly design, and **programming in the small** (PiS), formerly implementation. Testing and maintenance disappear as distinct phases for the following reasons: Testing is necessary in each area and also across these, to ensure consistency. Maintenance is considered not to be a separate area, but in fact is comprised of activities from the existing areas.

Moreover, to adequately describe the context of the software development process, additional working areas have been identified, which haven't been dealt with adequately up to know. These areas are **project management** (which comprises **version control** as well), **documentation**, and **quality assurance**. These socalled **nontechnical areas** interact with all of the above introduced technical areas, as shown in figure 7.3.

7.2 The Architecture Notation

The **architecture of a software system** is defined as its static structure, regarding packages, modules, and classes and the relationships among them. In the software development process, the

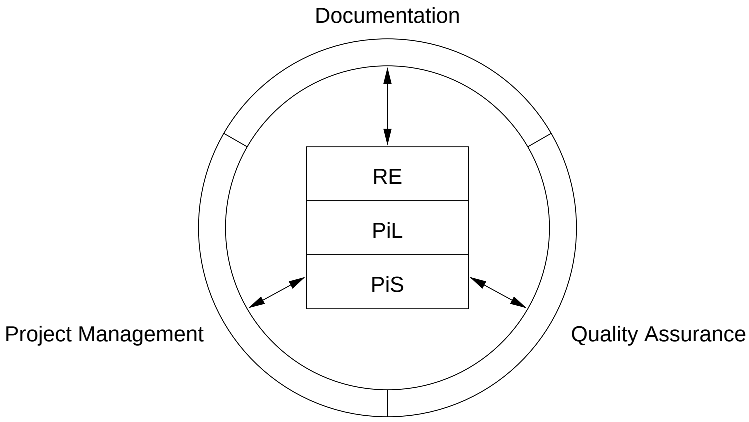

Figure 7.3: The working area model of software development

architecture is the output of the design phase (Programming in the large), and is the specification for the implementation (Programming in the small).

The best way to specify the architecture of software system is to use a graphical notation which shows the relations between the numerous units of the system. There are various standardized notations, such as *OMT* (*Object Modeling Technique*) after Rumbaugh [RBP+91], the *Booch Method* [Boo94], and the Jacobson method [Jac95], which have recently merged into *UML* (*Universal Modeling Language*) [Sof97].

Shortcomings of Existing Notations

When examining the applicability of these modeling languages to analysis and design of the spectral envelope library and the ViewEnv program, several shortcomings are apparent:

- Except for the now defunct OMT, these notations have no means of specifying data flow structures, but are specialized in describing object-oriented analysis and design. However, the spectral envelope library to be developed consists mainly of purely functional data transforming modules, where making the data flows explicit lends a very good insight into the structure of the system.
- It is advantageous to keep the distiction between abstract data types and functional module classes. In the above notations, principally all classes look the same. There are options to specify roles of classes, or to use adornments to express types of classes, but for the spectral envelope library, the distinction between data classes and functional modules is so important that it should be obvious on first sight. What's more, the diagrams can be kept simple and easier to understand.
- The object-oriented ViewEnv program would be amenable to one of the above standard notations while some other standard notation could be used for the mainly transformational spectral envelope library. However, this would undermine the consistency throughout this document.
- As the library is written in non-object-oriented C, it is unavoidable that the description of the architecture for the implementation will at some level of detail not match the structure of the implemented code anymore. That is, the notation for the architecture does not have to provide for means of adding more detailed information, it can be kept clean and simple.

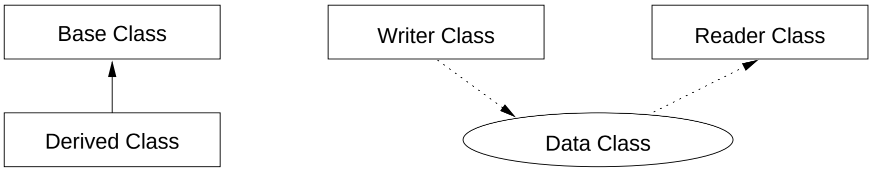

Figure 7.4: Inheritance relation between functional module classes (left), and data flow between a reader and a writer functional module (right)

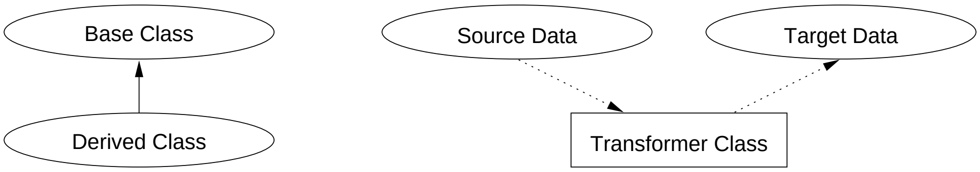

Figure 7.5: Inheritance relation between data classes (left), and data flow through a functional module (right)

The Adapted Notation

For these reasons, I decided to use an adapted notation which is very close to the standard notations mentioned above. It has two types of classes (data classes and functional module classes), two relations between classes of the same type (the inheritance relation and the "uses" relation), and data flows between functional modules which are typed by a data class.

Figure 7.4 shows the inheritance relation between functional classes and a data flow (always via an intermediate data class which specifies the type of the data flow). Figure 7.5 shows the inheritance relation between data classes and data flow which is transformed by a functional class. Figure 7.6, finally, demonstrates the usage relationship which expresses either a reference being held by the using class or an aggregation, i.e. the using class actually contains an object of the used class.[1]

Note that the data flows in the diagrams are on *class* level, not on *object* level. That is, e.g. the diagram in figure 7.11, page 69, does *not* say that there are three *Sound Data* objects (*Partials, Spectrum, Signal*) that are eventually combined into one *Sampled Spectral Env* object, but it does simply show the possible data flows between the classes.

7.3 General Considerations for Library Development

For the special case of a library, some of the design goals for software are particularly important. These are:

- Flexibility
- Expandability
- Practicality (ease of use)
- Re-usability

Since re-use is the very reason for existence of a library, some principles of re-use will be given in the next section, and how the spectral envelope library fulfills these.

[1]For the design of a complex software system it is important to express the decision between a reference and containment, especially because it makes a statement about liftimes of objects. In our case, however, where the system is rather a batch system with a linear flow of information, it is advantageous to abstract from that decision for reason of simplicity.

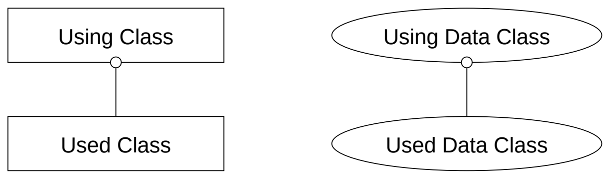

Figure 7.6: Usage relation for functional module classes (left), and data classes (right)

7.3.1 Principles of Re-use

After [Utt93], software is reusable if:

1. **It works.**
   This is assured by adopting the testing mechanism described in section 7.3.2.

2. **It is comprehensible.**
   This is cared for by the documentation of the architecture and the functions, and by a clear structure.

3. **It can co-exist with other software.**
   This point is accomplished in our case mainly in regard of compilation by using ANSI-C as a programming language, and not C++, since we want to keep compatibility with systems that are not object-oriented. Another effort is keeping the name-space proper by using the prefix `se` for all globally visible entities. In regard of functionality, the interfaces are clear and simple, and the usage of system ressources (memory, CPU) is moderate.

4. **It is supported.**
   This is a political question I can not comment on.

5. **It is economical.**
   Software is economical, when its use brings more benefit than the effort of setting it up or writing it oneself.

6. **It is available.**
   Within IRCAM, this is assured, outside, see point 4.

7. **It has been re-used.**
   The significance of this point is that only in the first re-uses the last incompatibilities and errors show up. The spectral envelope library has been re-used once so far (see section 8.6), but more applications are necessary.

7.3.2 Testing

An automated testing mechanism has been adopted for the development of the spectral envelope library which will be outlined in this section:

All the functions of the library are also accessible from command-line programs. These programs can be run from a test script, defining a number of test cases with their input data and program parameters. The generated output data will be automatically compared to stored reference data, which has been checked manually for validity. The test cases and the reference data are of course under version control.

It has to be taken care that the test cases cover all possible control paths in the program, especially the "unexpected" cases of error handling code and the like.

7.3.3 Error Handling in a Library

Although it is rather a question of implementation, some thoughts will be given to the error handling in a library in general, after which the mechanism adopted in the spectral envelope library will be explained.

Error handling in a function library is always a difficult point. Ideally, all errors will be detected and handled in the library, if possible. If not, they will be passed to the calling program. In any case, a library must never call the system functions `exit()` or `abort()` and thus stop the process if any unexpected situation occurs, because it is up to the caller to decide what to do in such cases, not up to the library. Who knows, maybe the calling program has a method to handle the error and can recover from it. What's more, detecting the cause of an error is made more difficult, when suddenly the program stops in some low level library function, without knowing when and by whom it was called. (Example: Compare the usefulness of the error message “error reading 4 bytes” with “error reading file header”.)

In an environment without exception handling such as ANSI-C, however, this behaviour of passing on error conditions requires some discipline on the side of the user of the library. It is mandatory that all return codes of library functions are checked for their status, since there may have been an error after which the program can not continue.

In the spectral envelope library, the error behaviour is controlled by the function `seSetErrorBehaviour` which can take a member of the enumeration type `seErrorBehaviour` as parameter. If the bit `seEbAbort` is set in the parameter, the library aborts (and core dumps) on any error[2]. If the `seEbPrint` is set, an error message is printed on the console (standard error). If the `seEbLog` is set, a detailed error message is also written to a log file. Various predefined combinations of the error behaviour flags exist: `seEbFull`, the default, has all three flags set; `seEbNorm` is the normal behaviour for well-behaved programs, and means: continue on error, but print and log the error message (`seEbLog | seEbPrint`).

7.4 The Development of the Spectral Envelope Library

This section describes how the spectral envelope library was developed. First, the architecture will be described as the central design documentation in 7.4.1. Then, a few words will be said about how we got there from analysis, and some more words about how we're going from there to implementation in 7.4.2.

7.4.1 The Architecture

The architecture of the spectral envelope library is described using the notation introduced in section 7.2. The top-level data flow diagram in figure 7.7 shows the different transformations between sound data and the spectral envelope representation. Subsequently, the data and functional classes will be further decomposed into subclasses. An additional data class *Control Data*, which provides input to every functional class, is left out for the sake of lisibility.

The two main data classes, *Sound Data* and *Spectral Envelope*, the representation class, and their subclasses are shown in figures 7.8 and 7.9, respectively.

The diagrams in figures 7.10 to 7.15 show the functional classes *Estimation*, *Synthesis*, and *Interpolation*. They are grouped in pairs of two. The first diagram of each pair shows the subclasses of the respective functional class. The second diagram of each pair shows the input and output data flows of the subclasses. (The data flows of the base classes are all shown in the top-level diagram in figure 7.7.)

[2] Because the discipline required to check all return codes may cause problems to unexperience programmers and is not adequate for a quick test program, where you don't want to write an error check for every second statement, there is also this option to have the library abort on error, although it is not a good behaviour for a library in general.

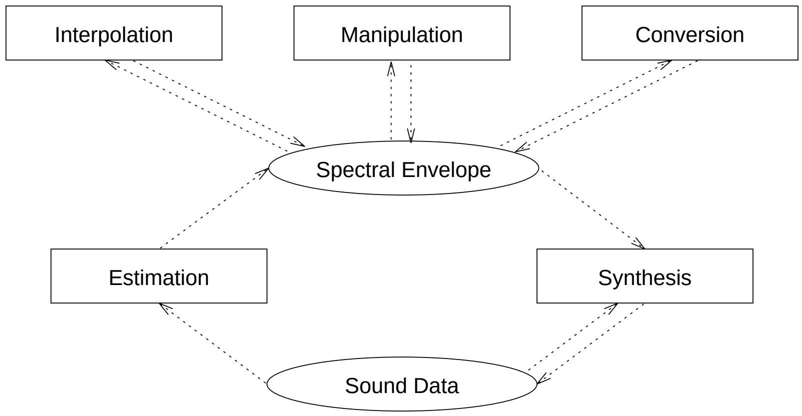

Figure 7.7: Top level data flow diagram of the base classes of the spectral envelope library. The base classes will be refined in the subsequent diagrams.

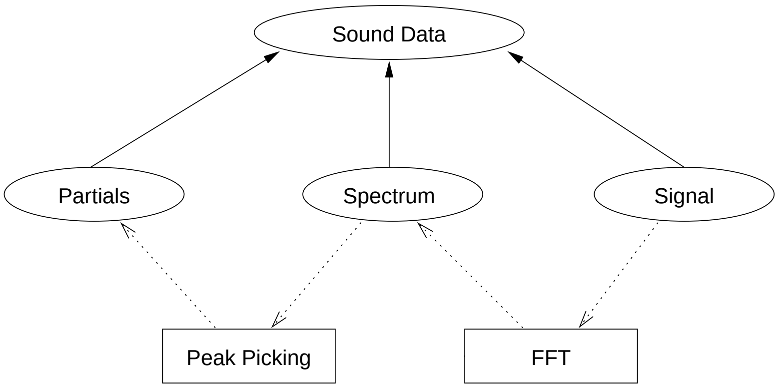

Figure 7.8: The *Sound Data* class, and its different subclasses, with the transformations between them

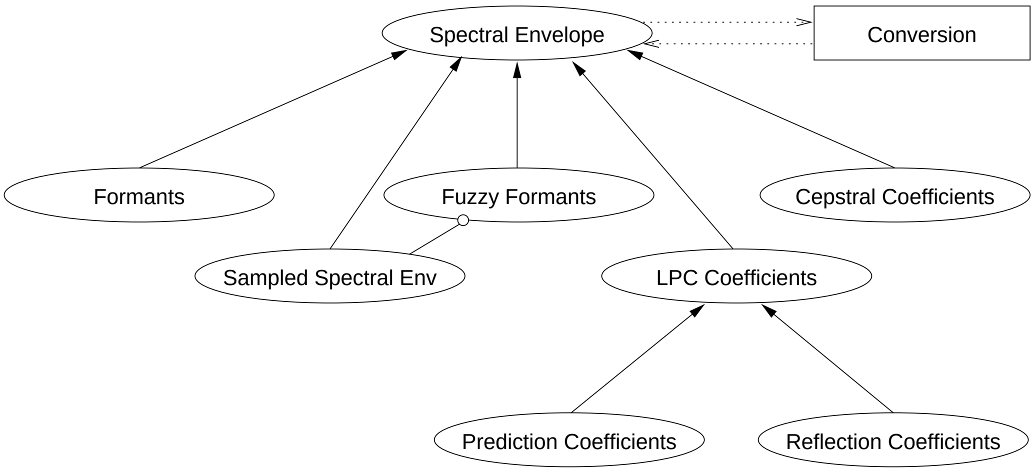

Figure 7.9: The representation data class *Spectral Envelope* and its subclasses, the different representations of spectral envelopes. The *Conversion* functional class does all conversions from one spectral envelope representation in another and is not further refined.

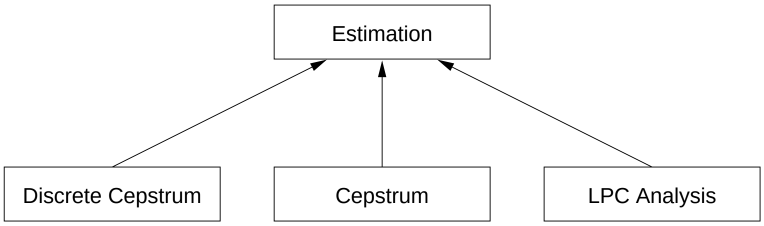

Figure 7.10: The *Estimation* class and its subclasses, the implemented methods for spectral envelope estimation

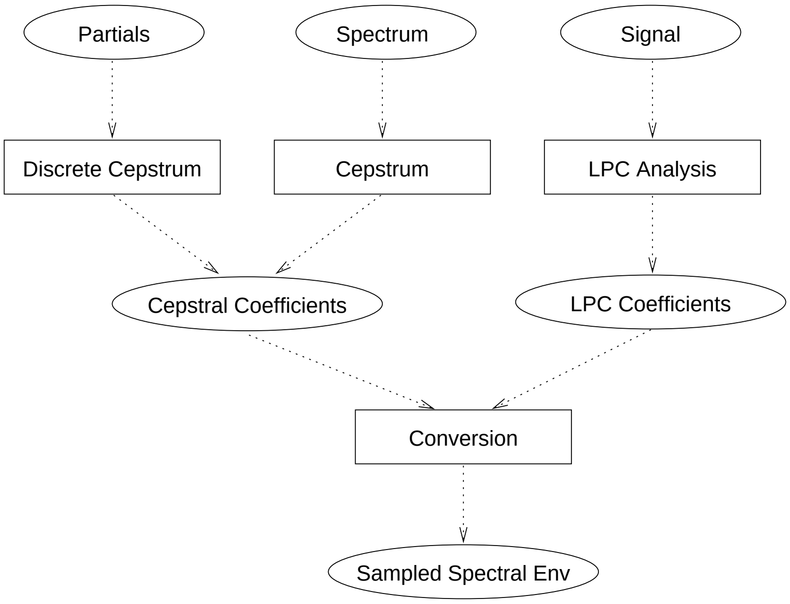

Figure 7.11: The subclasses of the *Estimation* functional class with their *Sound Data* input data flows and *Spectral Envelope* output data flows.

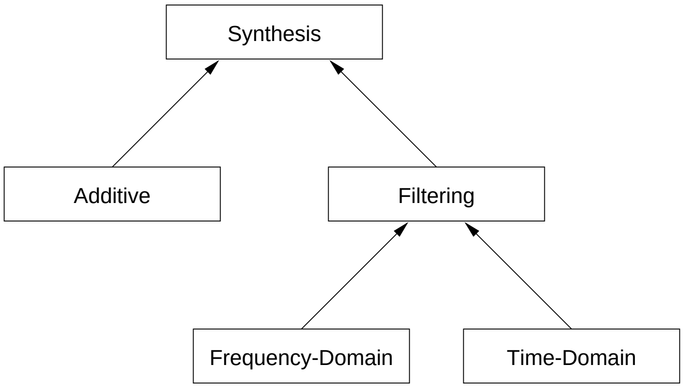

Figure 7.12: The *Synthesis* functional class and its subclasses, the different methods for synthesis with spectral envelopes

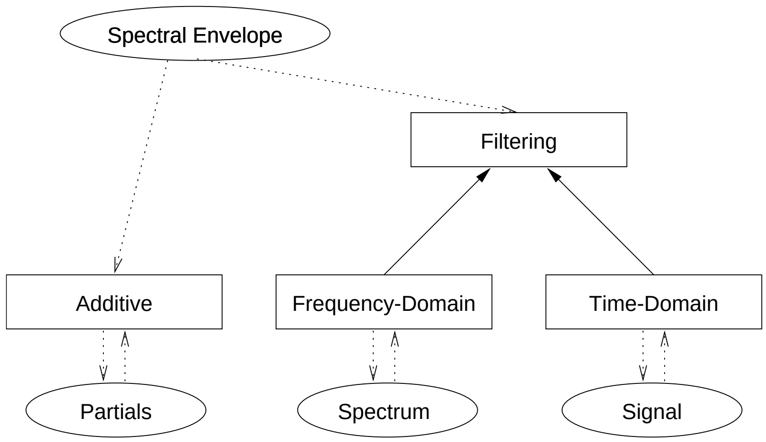

Figure 7.13: The subclasses of the *Synthesis* functional class with their input/output data classes (the subclasses of *Sound Data* and a spectral envelope as the data class *Spectral Envelope*)

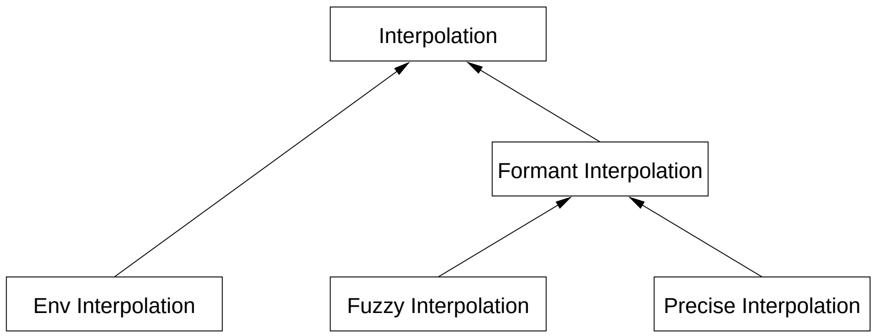

Figure 7.14: The *Interpolation* functional class and its subclasses, the methods for interpolation of spectral envelopes. *Formant Interpolation* is a separate base class because the matching of formant indices is handled there.

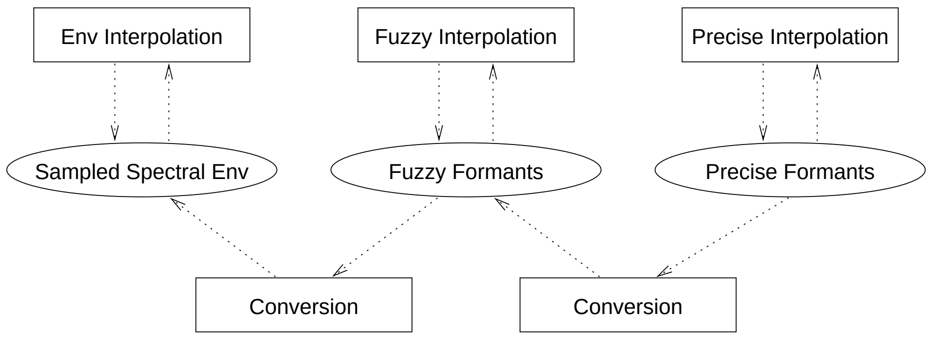

Figure 7.15: The subclasses of the *Interpolation* functional class with their input/output data classes (the subclasses of *Spectral Envelope*). As can be seen, conversion can only be done in the direction to the less structured representation.

7.4.2 Going from Analysis to Implementation

The architecture diagrams in figures 7.7 to 7.15 specify the design of the spectral envelope library. However, they are directly derived from the analysis of the requirements for the library. In fact, the diagrams that are the output of the analysis phase (the requirements engineering) look the same, except that some details like usage-relationships are missing. This is the reason that the analysis phase is not explicitly documented. This supports the claim, by the way, that the adapted notation is indeed useful, because it offers a transparent transition from the analysis of the software system to its design.

In going from design to implementation, however, it is impossible to make a transparent transition, since there we reach the boundary of the object-oriented paradigm. While in analysis and design, it was still possible to describe the world as consisting of interacting objects, each pertaining to one of several classes, in our implementation in non-object-oriented C, we have to talk about functions, modules, and data structures. Because we can't transfer the object-oriented paradigm completely to implementation, some desirable properties of object-oriented programming will not hold. Specifically, the aspects of encapsulation, inheritance and type-polymorphism will be lost, as explained in the following:

Encapsulation

The C-structures that will be used instead of classes offer no compiler-enforced protection mechanism regarding the access to the data fields. However, for the data classes explained below, **access functions** can be provided, that set the value of one data field, or return its value. If the user of the library by convention uses these access functions, sufficient encapsulation between the interface and the implementation of the data structures is provided. This means that, for example in the case the internal data structures change, the code using these structures will not have to be changed.

Inheritance

The inheritance relations (also called subtype or generalization/specialization relations) used in the analysis and design will not be visible in the implementation. Instead, the classes will be mapped to C-structures as shown in section 7.4.3.

Polymorphism

Polymorphism is closely linked to inheritance and means the access to a certain functionality that applies to all subclasses of a base class by one method name. This name maps to the appropriate method[3] of the actual subclass.

Polymorphism could be mimicked in C by storing pointers to member functions with each object. Apart from the complexity and usage of space of that procedure, type polymorphism is, on closer scrutiny, not really necessary for the spectral envelope library. This is because the number of generic functions (e.g. *save*) which have to be applied to all subtypes of one general type are fairly few. Instead, specialized functions for the classes exist. For example, the three different subclasses of sound data (*Signal, Spectrum, Partials*) are processed by three different estimation methods (*LPC Analysis, Cepstrum, Discrete Cepstrum*) to yield two different output data types (*LPC Coefficients, Cepstral Coefficients*). I.e. the data flows on parallel "tracks" that rarely cross. In another case, formant interpolation, it is necessary to know the actual subtype of spectral envelope representation (*Sampled Spectral Envelope, Precise Formants, Fuzzy Formants*) to choose the specialized method to perform the formant shift (see section 5.1.2). Therefore, it is advantageous to keep the subtype information explicit in the code and not generalize to a superclass.

[3]A method is a *member function*, i.e. a function that is a member of a class and can therefore be called only for an object of that class and its subclasses.

7.4.3 Example of a Data Structure: The Spectral Envelope Class

As an example of the implementation of the data classes, the definition of the structures for the *Spectral Envelope* class, from the class diagram in figure 7.9, are given below:

The sampled spectral envelope, or spectral representation, is:

```
typedef struct
{
    float       maxfreq;     /* upper border of env (maxfreq = sampling rate/2) */
    int         numenv;      /* number of points in envelope                    */
    float       *env;        /* numenv spectral envelope amplitude values       */

    float       fstep;       /* size of bin in env  (fstep = sr / numenv)       */
    seScale     scale;       /* frequency scale for storage                     */
    float       breakfreq;   /* break frequency for log scale                   */
}   seSpecEnv;
```

Where `seScale` gives the type of the scaling and is defined as:

```
typedef enum { seScLinear, seScLog, seScNum }   seScale;
```

For the spectral envelope representation as precise formants, we use a list of structures for formants, and a residual spectral envelope, which represents all components that are not of the shape of a formant:

```
typedef struct
{
    float       centerfreq;     /* precise formant parameters */
    float       amplitude;
    float       bandwidth;
}   seFormant;

typedef struct
{
    int         numformants;    /* number of precise formants   */
    seFormant   *formants;      /* numformants precise formants */
    seSpecEnv   residual;       /* residual spectral envelope   */
}   sePreciseFormants;
```

For the fuzzy formant representation, we use a list of formant regions within a sampled spectral envelope:

```
typedef struct
{
    float       lower;          /* region boundary frequencies (Hz) */
    float       center;
    float       upper;
    float       salience;       /* how sure we are it is a formant  */
}   seRegion;

typedef struct
{
    int         numformants;    /* number of precise formants   */
    seRegion    *formants;      /* numformants precise formants */
    seSpecEnv   specenv;        /* spectral envelope            */
}   sePreciseFormants;
```

7.5 The Spectral Envelope File Format

The file representation of spectral envelopes is based on SDIF (*Sound Data Interchange Format*) [Vir98]. This section gives a brief overview of the SDIF format, followed by a presentation of the types that will be added to the format for spectral envelopes. The final definition is subject to change because it is still being discussed with CNMAT, IRCAM's partner in the SDIF project.

An SDIF-file is organised in *chunks*. It starts with an opening chunk containing the header, followed by some optional chunks giving file-global user-defined data in field-name/value pairs. These are called *Name-Value Tables* (NVTs). The body of the file is a contiguous sequence of time-tagged frames (which are themselves chunks), sorted in ascending temporal order, with multiple kinds of frames allowed in a single file or stream. A library of standard frame types defines formats for storing common sound representations that are part of the SDIF standard. The data in a frame are stored in matrices (or vectors) of floating point numbers, with each column corresponding to a parameter like frequency or amplitude and each row representing an object like a filter, sinusoid, or noise band.

SDIF allows the definition of new frame and matrix types, even in a file. For this end, a small data-definition language has been defined. The new types necessary for spectral envelopes are now given in this format, for each of the representations.

Spectral (Sampled) Envelopes

The following statements define the matrix and frame types for the samples spectral envelope representation (see section 4.3):

```
1MTD 1ENV { amplitude }
1MTD 1ENI { SamplingRate, FrequencyScale, FrequencyScaleParameter }
1FTD 1ENV {
            1ENI envelope-info;
            1ENV envelope;
          }
```

The codes `1MTD` defines two matrix types. The first, `1ENV`, holds one column and n rows of amplitude values v_i for the bins of the spectral envelope at frequency $i/n*f_s/2$ (for the linear case). The second, `1ENI`, (where `ENI` stands for envelope information), holds the sampling rate, frequency scale (linear or logarithmic), and the break point parameter for the case of logarithmic frequency scale.

`1FTD` defines a frame type also called `1ENV`, which contains one matrix of type `1ENV` and one of type `1ENI`. The frame and matrix types have distinct name spaces and can bear the same name. The leading `1` is a version indicator. When a frame and matrix types is expanded by more data items, the version indicator will augment.

These values in `1ENI` are also given in the name-value table at the beginning of the file, but in order to keep the file applicable for streaming (where the header might have been missed by a reader), they are repeated with each frame.

Filter Coefficients

The following statements define the matrix and frame types for the LPC and cepstrum filter coefficients (see section 4.2):

```
1MTD 1ARA { a }
1FTD 1ARA {
            1ENI envelope-info;
            1ARA AR-coefs;
          }

1MTD 1ARK { k }
```

```
1FTD 1ARK {
          1ENI envelope-info;
          1ARK AR-coefs;
        }

1MTD 1CEC { c }
1FTD 1CEC {
          1ENI envelope-info;
          1CEC cepstral-coefs;
        }
```

The number n of rows of each coefficient matrix `1ARA`, `1ARK`, or `1CEC` is the order of the LPC, or cepstrum.

Alternative Definition

At the moment, the method described above has been adopted. Nevertheless, it has the disadvantage that defining the 4 frame/matrix types above clutters the SDIF name space with structurally identical and semantically very similar types. Alternatively, one could imagine a single frame type with an info matrix and a data matrix, the info matrix giving the type of the data and all parameters necessary to interpret them:

```
1MTD 1FIF { CoefficientType, SamplingRate,
          FrequencyScale,  FrequencyScaleParameter }
1MTD 1FCF { coefficients }
1FTD 1FCF {
          1FIF filter-info;
          1FCF coefficients;
        }
```

The disadvantage of this method is that the actual type of the data is hidden in a floating point field, even for something as unambiguous as LPC coefficients. What's more, this method repeats the typing mechanism of SDIF, but losing the clarity of the 4-letter type signatures.

Formant Description

For the representation of a spectral envelope as formants (see section 4.5), we need two types: precise formants and fuzzy regions, where a formant is suspected. The precise formants would be defined by:

```
1MTD 1FRM { frequency, bandwidth, amplitude, label }
1FTD 1FRM { 1FRM formants; }
```

The fuzzy formants would be a combination of an envelope `1ENV` and a formant region `1FRR`:

```
1MTD 1FRR { lowerfrequency, upperfrequency, centerfrequency,
          confidence, label}
1FTD 1FRR {
          1ENI envelope-info;
          1ENV envelope;
          1FRR formant-regions;
        }
```

If the center frequency is not known, the column could be missing or a value of $-\infty$ could be given. The labels can be used to identify and track formants, necessary for interpolation, the confidence parameter gives a hint of the how sure it is that there is a formant in that region.

Chapter 8

Applications

This chapter describes some applications of the spectral envelope methods and tools developed in this project. It covers both already existing applications and possible applications. These are grouped into applications around additive analysis–synthesis (section 8.1), synthesis of the singing voice (section 8.2), enhancements of the DIPHONE program (section 8.3), applications to real-time synthesis and signal processing (section 8.4), conversion between spectral envelopes and images (section 8.5), and application in voice synthesis by rule (section 8.6).

8.1 Controlling Additive Synthesis

Additive analysis–synthesis is a powerful way to completely parameterize a sound event into sinusoidal partials with their frequencies, amplitudes, and phases. This benefit is also its curse: It puts every minute detail of a sound event at our disposal, but leaves us with the task to control and manipulate this mass of parameters in a sensible way. So far, control is done by specifying the change of every single parameter over time by break point functions (see section 4.4). Since the number of partials can easily rise into the hundreds, modifications are tedious. Moreover, doing valid manipulations in regard of signal processing and from a musical perspective is not obvious, and, what's more, the parameters are interdependent (e.g. changing the frequency of the partials changes the spectral envelope, often with undesirable results, as shown in section 2.3.1).

In [FRD92] it is suggested to use spectral envelopes to control the amplitudes of the partials for resynthesis. This drastically reduces the number of parameters, provides us with parameter sets which are easily understandable (e.g. formants), and renders frequency and amplitude control independent from each other.

Also, for the residual noise, the modeling by filtering of a white noise with spectral envelopes (estimated with linear prediction) renders this component of sound accessible to manipulation. This has not been possible before in the sampled signal representation of the residual.

The most significant advantage, however, lies in the unified handling of noise and harmonic part, because the spectral envelope of the residual noise is represented in the same way as the spectral envelope of the sinusoidal part. Therefore, the very same manipulation can affect both parts synchronously, if this is desired [RDG95].

8.2 Synthesis of the Singing Voice

One of the primary applications of the spectral envelope handling developed in this project is the high quality synthesis of the singing voice. In the additive synthesis paradigm, synthesis is a resynthesis of the previously analysed and modified voice. To effect the modifications in a sensible manner, the constraints posed by the speech organs have to be taken into account.

For example, as demonstrated in section 2.3.1, transposition of the voice quickly sounds very unnatural when the spectral envelopes are not corrected, because they reflect the configuration, especially the length, of the vocal tract. To avoid this, the transposition program which is part of the spectral envelope library has the possibility to automatically estimate the spectral envelope of the original sound and reconstitute it by applying it to the transposed sound.

Also, many aspects of the expressivity of the singing voice (as well as prosody in speech—see section 5.2) depend on the spectral envelope, i.e. on timbral variations, rather than on pitch and amplitude alone.

With the methods of interpolation between spectral envelopes and formants, a new type of high quality additive synthesis of voice is possible. To preserve the rapid changes in transients (e.g. plosives), and the non-formant shaped noise spectral envelopes in fricatives, these are best synthesised with the harmonic sinusoids + noise model, controlled by envelopes in spectral representation. For precise formant locations in the steady part of vowels, the formant representation of spectral envelopes can be specified.

8.3 Integration in Diphone

It is planned that the spectral envelope library is ported to the Apple Macintosh system, allowing the integration of spectral envelope manipulation in the DIPHONE program. This will be in the form of an extension of the additive synthesis plugin, such that the manipulations of spectral envelopes that are possible with the library, will be controlled by the intuitive graphical user interface of DIPHONE.

What's more, spectral envelopes allow for the convenient modeling and manipulation of the residual noise, which could be made accessible to the additive synthesis plugin of DIPHONE, and to the CHANT plugin.

With the interpolation capabilities between precise formants and spectral envelopes with marked formant regions (fuzzy formants, see section 5.1.2), it is even possible to bridge the gap between the two synthesis methods, and to interface the excellent generation of vowels in CHANT with the flexibility of ADDITIVE in DIPHONE.

8.4 Application to Real-Time Signal Processing

In the framework of the FTS/*jMax* system for real-time signal processing, a module to estimate spectral envelopes, manipulate them, and apply them to synthesis would widen greatly the range of applications of the system. Flexible, musically controllable manipulation paradigms in real-time using spectral envelopes could be developed using the graphical programming environment of *jMax*. However, the demands in processing power for real-time application of the spectral envelope handling algorithms—especially of estimation—would have to be evaluated.

The already existing real-time implementation of additive synthesis would benefit greatly from the addition of a noise model for the residual part of a sound, allowing sharp transients like e.g. in the attack of a sound to be reproduced or generated exactly. Also, for many sounds, a noise component is characteristic and always present, as in the turbulence noises of wind instruments.

Another possible application is in the ESCHER system, based on *jMax*, designed to provide an intuitive and expressive control of sound synthesis in real-time by all types of gestural input devices [WSR98].

The real-time additive synthesis component in ESCHER is based on an n-dimensional timbral parameter space[1], spanned up by points of different combinations of the n parameters. At each point, a note of a real instrument played with these timbral parameters is subjected to additive analysis, and the resulting partial sets are stored. Now, when playing the ESCHER system, the partial sets are interpolated according to the current timbral parameters.

[1] Timbral parameters being e.g. dynamics, pitch, harmonicity, noise, etc. They are completely user-defined.

There are now two possible applications of spectral envelopes in ESCHER: First, data compression could be used, since the amount of additive analysis data is not negligible (remember that a whole note comprising several hundreds to thousands of time-frames of data is stored at each point of the multi-dimensional parameter grid). Storing the development of the partial amplitudes as a spectral envelope, while the partial frequencies are completely determined by the pitch, would drastically reduce the amount of data to be stored.

Second, along the same lines as in section 8.1, the representation of the instrument as spectral envelopes would allow for better manipulation of the sound (so far, only interpolation is possible). These manipulations could include various partial transformations, which will even amount to *extrapolating* the sound, i.e. creating sound characteristics which are not restricted to that of the instrument that was originally modeled, resulting in more flexibility and expressiveness.

8.5 Conversion between Spectral Envelopes and Images

A playful application which came to my mind and which was easy to implement is the conversion between spectral envelopes over time (spectrograms) and bitmap images. A **spectrogram** is a two-dimensional presentation in the time–frequency plane of the spectrum of a sound over time. Here, of course, it displays the course of the spectral envelope over time. Usually, time runs from left to right, frequency runs from bottom to top, and amplitude is coded by brightness (on screen) or blackness (on paper). I.e. one column x in the image corresponds to one time-frame $t = x'$, and the brightness or blackness values for a pixel (x, y) in that column are read from the spectral envelope $v_t(f)$ at time t and frequency $f = y'$, with the tick operator $'$ being some mapping from image coordinates to the time–frequency plane.

The generation of the bitmap is trivial, and for input/output in an image file format, the PBMPLUS library was used. This library generates an intermediate file format, which can be subsequently converted to and from all other existing image file formats.

There is already an interest in converting spectral envelopes to images, namely **visualization** of spectral envelopes. In figure 8.1, the upper left image shows the spectral envelopes of a mongolian chant over time (the blacker the colour, the higher the amplitude). Sometimes it is interesting to see how the spectral envelope changes over time, how the formants move, etc. But, the other way seems even more interesting to me: reading a bitmap image and converting it to a spectral envelope. This works analogously, just that the brightness of a pixel (x, y) of the image is converted to the amplitude of the spectral envelope: $v_t(f)$ at time $t = x'$ and frequency $f = y'$. The resulting spectral envelope is then applied to some sound's partial structure and an additive synthesis of the resulting sound is performed.

Now that we have both directions of conversion, we can apply the vast manipulation techniques developed for graphic design and visual arts to images of spectral envelopes, and listen to the results. Two examples of such manipulations can be found in figure 8.1. Interesting enough, the visually more spectacular *twirl* in the upper right image sounds rather dull (it doesn't change the original sound much), much less interesting than the *spread* effect in the lower left of the figure, a granulation-like sonic event, but across the whole spectrum.

Finally it is possible to take an arbitrary image and apply it as a spectral envelope to additive synthesis. This opens up a vast space of new sonic possibilities. For example in the lower right image of figure 8.1, which depicts a long-time exposure of a nightly London street, you can hear every pillar of the building in the background as a pulse, while the white streaks of the lights of passing buses sound like a resonant frequency sweep. (In this image, amplitude maps to brightness, contrary to the other three images, but preserving the image.)

I agree that the serious applicability of this conversion is dubious, because the image manipulations have nothing to do with the properties of the data as a signal, so there is the risk of arbitraryness. Nevertheless, the results are surprising and stimulating. I don't claim that it is more than playing around, but isn't playing around—or *exploring*—at the very heart of creativity?

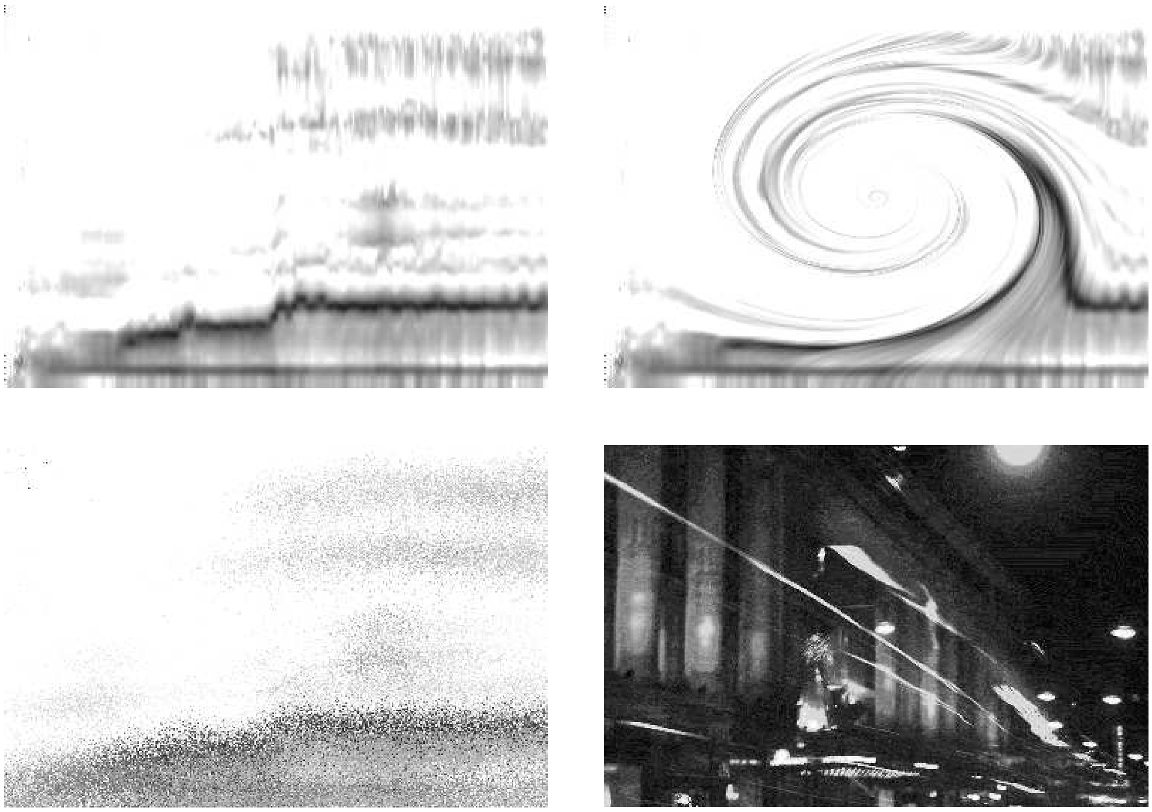

Figure 8.1: Example for converting between spectral envelopes and images: The spectrogram of a mongolian chant to the upper left is put through various manipulations, a *twirl* filter in PHOTOSHOP in the upper right, a *spread* effect in XV in the lower left. The lower right image shows an arbitrary photograph to make the reader's mind wonder what it might sound like when interpreted as a spectral envelope and applied to the mongolian chant (see in the text).

8.6 Voice Synthesis by Rule

Finally, both the VIEWENV program and the spectral envelope library and programs have already been applied in a project in the analysis–synthesis team [Hen98]. The topic was analysis and synthesis-by-rule of the singing voice. The spectral envelopes of recordings of a singer were estimated with the spectral envelope library, after which the displaying and measuring capabilities of the VIEWENV program were used to retrieve the formants and their parameters to set up rules describing the dependencies of the formants on pitch. Also, the SDIF file format for spectral envelopes was used for the first time.

Chapter 9

Conclusion and Perspectives

This chapter will give a summary and conclusion of the spectral envelopes project, followed by perspectives both for future research and application of spectral envelopes, as well as for artistic application.

9.1 Summary of the Project

In the project *Spectral Envelopes in Sound Analysis and Synthesis*, various methods for estimation, representation, file storage, manipulation, and application of spectral envelopes to sound synthesis were evaluated, improved, and implemented. The prototyping and testing environment ViewEnv has been developed, and a function library to handle spectral envelopes was designed and implemented.

For the **estimation** of spectral envelopes (chapter 3), after a definition of the requirements, the LPC, cepstrum, and discrete cepstrum methods were examined, all of which were implemented. Various separate possibilities of improvements of the discrete cepstrum method (regularization, stochastic (or probabilistic) smoothing, logarithmic frequency scaling, and adding control points to the envelope) were examined and combined. A comparison of the effects of these improvements was carried out and an evaluation with a large corpus of sound data showed the feasibility of discrete cepstrum spectral envelope estimation.

After defining the requirements for the **representation** of spectral envelopes (chapter 4), filter coefficients, spectral representation, geometric representations (break-point functions and splines), formant representation, and high resolution matching pursuit were examined. After a comparison of the methods in regard of the requirements, spectral representation, filter coefficients, and formant representation was chosen. A combined spectral representation with indication of the regions of formants (called "fuzzy formants") was defined to allow for integration of spectral envelopes with precise formant descriptions.

For **file storage**, data types to store each of the representations used in the project were defined in the framework of the SDIF standard sound description file format (see section 7.5).

Various types of **manipulations** were examined in chapter 5. Special attention has been given to interpolation between spectral envelopes, and between spectral envelopes and formants. Other manipulations, based on primitive operations on spectral envelopes, affecting the amplitude and the frequencies of spectral envelopes have been covered.

For the application of spectral envelopes to sound **synthesis**, the two cases of additive synthesis and filtering have been examined in chapter 6. For the latter, the conversio of the different representations to time-domain or frequency-domain filters is given.

In order to easily develop the algorithms for spectral envelope estimation, and to compare the effects of the different parameter settings, the ViewEnv spectral envelope **viewing application** was developed in the course of the project under the Matlab programming environment.

Finally, the **spectral envelope library**, which combines all of the methods evaluated in the project, and makes them accessible to other programs, has been developed. The principles of software engineering were applied to the requirements analysis, the design of the software architecture, to the implementation, and to the testing mechanisms devised, as described in chapter 7.

9.2 Perspectives of Future Research

Apart from the prospective applications of spectral envelopes mentioned in chapter 8, which would all be possible with the results of the project as is, other interesting ideas and enhancements would need more basic research. Some of these are described in the following.

9.2.1 Optimization of Spectral Representation

Further reasearch is possible in the optimization of the representation of spectral envelopes as a sampled curve in the frequency–amplitude plane. The number of points neccessary to precisely represent a spectral envelope depends on the frequency range to cover, and on the complexity of the envelope. It could be automatically checked if the number of points is high enough by an analysis of the spectral envelope to be represented, and the frequency grid could be adapted to the requirements. Even with the current spectral representation, nothing constrains us to store the same number of points at each frame. Thus, for frames containing silence, only one point would be stored to indicate that a spectral envelope is present but on the level of silence. This would take up almost no space, while frames with complicated spectral envelopes could be stored more accurately.

9.2.2 Wavelets for Representation

An interesting perspective, which has not been explored in this project, is the application of the theory of **wavelets** to spectral envelopes. Without going into details, wavelets are a way to analyse a signal on multiple time-frequency resolutions simultaneously. While Fourier analysis uses sinusoids with an infinite time support, wavelet analysis uses different possible sets of basis functions with limited time support. See [Cha95] for an introduction to wavelets, [Mal97] for a detailed description, and [Hub97] for a insightful account of the history of wavelets.

Wavelets could be applied to the representations of spectral envelopes in two ways. First, they could be used for data compression in the framework of sampled (spectral) representation of spectral envelopes.

Second, after an idea of Rémi Gribonval, a completely wavelet-based representation might be possible, which could render manipulation very easy: Figure 9.1 shows an example spectral envelope. After the analysis of the spectral envelope, interpreted as a signal, with wavelets at different scales [MZ92], the inflection points (where the curving of the spectral envelope changes direction) are available, equally at different scales, as shown in figure 9.2. This information is sufficient to reconstruct the spectral envelope with good precision.

It can be seen that each peak in the spectral envelope in figure 9.1 is identified by two inflection lines in the analysis in figure 9.2.[1] It is now easy to imagine moving these lines to shift formant peaks, or changing the spacing to modify their bandwidth.

[1] Contrary to spline representation (section 4.4), where there are inflection and extrema *points*, the wavelet analysis yields information about the curve behaviour at multiple resolutions.

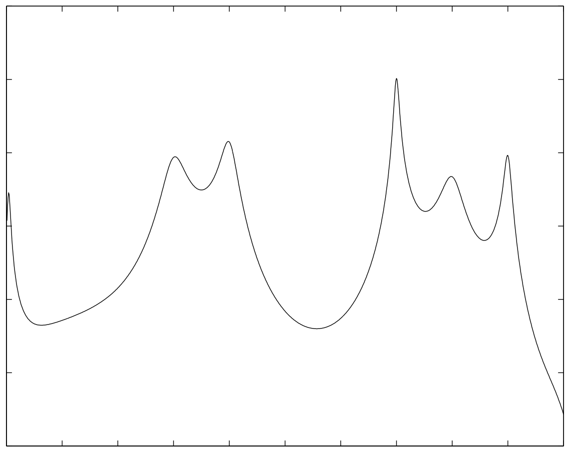

Figure 9.1: An artificial spectral envelope

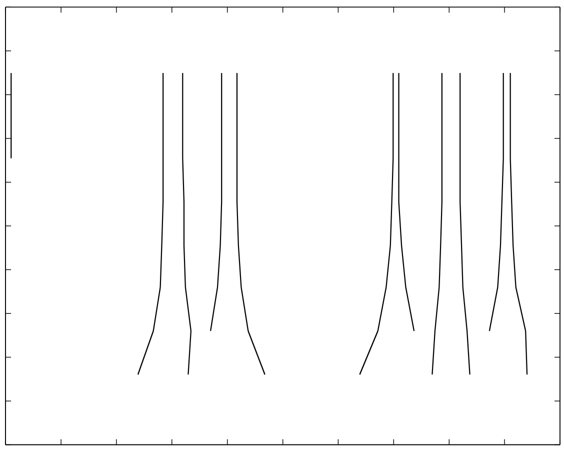

Figure 9.2: The inflection points of the spectral envelope of figure 9.1. While the x-axis is in the same frequency scale as in figure 9.1, the y-axis gives the resolution scale of the wavelet analysis.

9.3 Possibilities of Artistic Application

In the context of computer music, the control of spectral envelopes offers the possibility to influence the timbre of a sound to a great degree, allowing composers to obtain a desired effect or characteristic of a sound by the use of a flexible, unconstrained representation (the spectral representation gives complete freedom to modify a spectral envelope in any way). To the performer, the possible real-time application of spectral envelope manipulation would greatly enhance expressivity through easily understandable and "musical" parameters, i.e. parameters that pertain to a model (the source–filter model) which is valid for many instruments.

Between the creation of completely new sounds and the modification of existing sounds lies the combination of features of different sounds. For example, *cross synthesis* with spectral envelopes crosses the characteristics of two distinct sounds: The partial structure is taken from one sound, and the spectral envelope from another.

However, the application of the methods and tools developed in this project will not only enrich musical creation, but also the demands of art will in turn influence further research conducted on this topic:

Whenever I asked computer music composers and performers what they would do if they had the possibilities to manipulate the spectral envelope of sounds, they inevitably came up with ideas about varying spectral envelopes in time.

For a worthwile artistic application, we have to raise our point of view above the one-dimensional perspective adopted in most of this project, in that the richness and complexity of sound—which is an inherently time-based phenomenon—were only viewed through the keyhole of one time-frame.

Alas, this is out of the scope of this project. Here, the mechanisms for effecting these changes have been developed, the question how to apply and control them has to be answered elsewhere.

Appendix A

Envelope Viewing Application

The spectral envelope and signal viewing application ViewEnv was written to allow for easy interactive testing and evaluation of the algorithms for spectral envelope estimation. All of the important parameters for the different estimation algorithms are directly accessible, either by numerical entry or by intuitive sliders, and changes are reflected immediately in the graphical output. Special care has been taken to facilitate comparing different algorithms, or different parameter settings for the same algorithm.

ViewEnv was written using the Matlab programming environment, taking advantage of the high-level mathematical functions which are provided, and the rapid prototyping style of programming, due to Matlab being an interpreted language.

In the following, I will describe the program from two different viewpoints. First, in section A.1, aimed at the user who wants to view envelopes or visualize the effect of parameters, second, in section A.2 for the developer who wants to extend the program by new algorithms or functions.

A.1 Usage

This user documentation of the spectral envelope and signal viewing application ViewEnv is structured as follows: After explaining how to start the program in section A.1.1, the basic entities the program handles, files and curves, are introduced in section A.1.2. This section will also explain how to select what to display, and how to navigate in the data displayed. Section A.1.3 will explain several other functions associated with the display window. Section A.1.4 will explain the control window where the parameters for the various spectral envelope algorithms are specified. Each parameter will be described in detail. Section A.1.5 will explain the manipulation window, section A.1.6, finally, will explain the possibilities for interactive evaluation of the spectral envelope estimation methods.

A.1.1 Running ViewEnv

To start ViewEnv, you have to enter its directory (`src/viewenv`) and start Matlab by typing `matlab5`. This will call the script file `startup.m`, which sets the correct search paths and starts the program by calling the function `run`. This function can also be called from the Matlab prompt to reinitialize the program.

This should bring up several windows, the display window (figure A.1) for display of data and general settings, the control window (figure A.2) for loading data and setting parameters, and the manipulation window (figure A.6) for manipulation of spectral envelopes.

If you're using Matlab 5.0 or 5.1, you'll have to live with some bugs which are fixed in version 5.2: First, when more characters are entered in a text edit field than fit in the width of the field, the text becomes invisible. However, it is still there and can be used (just not seen), and when characters are deleted, it becomes visible again. Second, ViewEnv is programmed using the object-oriented programming features of MATLAB5. In the early versions, the object and class handling was not implemented correctly, leading to strange errors upon user actions, which would go away, when tried a second time. Just try again.

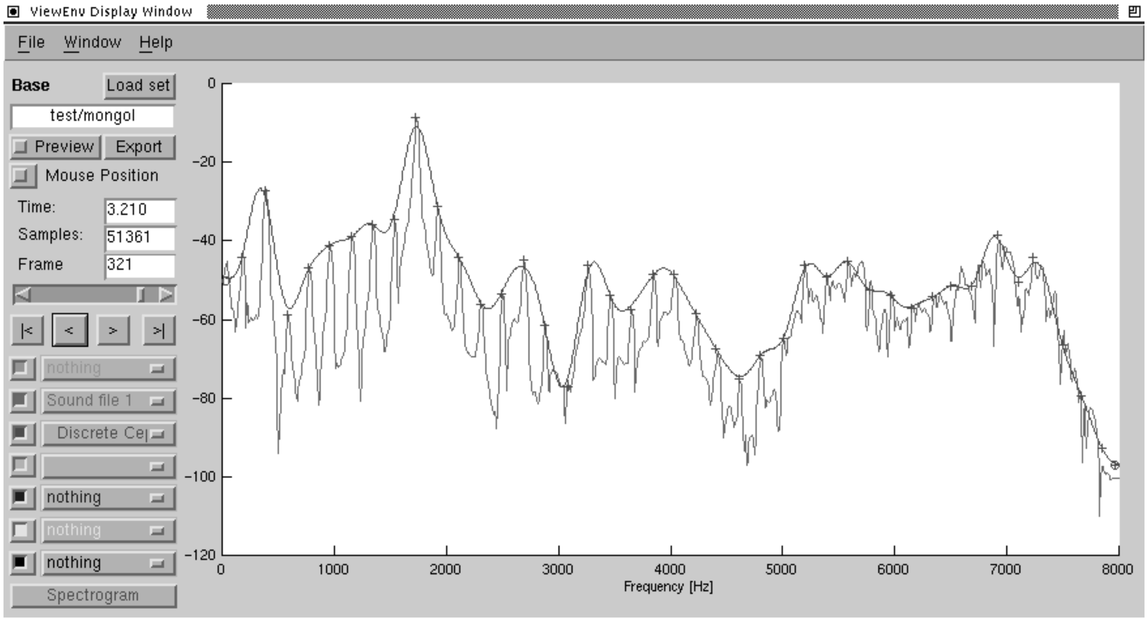

Figure A.1: The VIEWENV display window. The main part of the window is taken up by the *display axes*. To the lower left are the *display pop-ups*, to the upper left is the *navigation area* and *interaction area*.

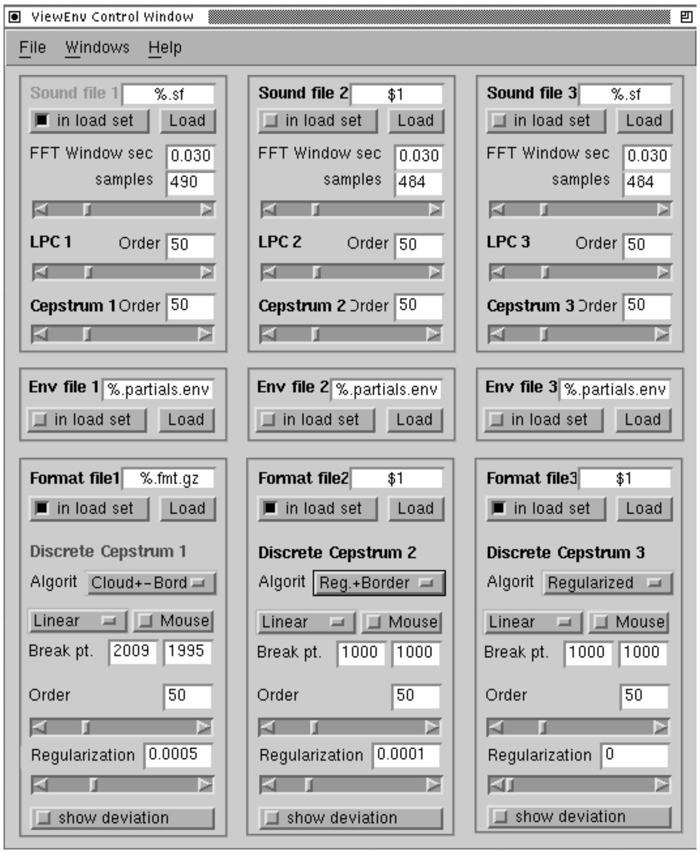

Figure A.2: The VIEWENV control window.

A.1.2 Files and Curves

ViewEnv can load files containing various types of data and display them directly, as well as computing data from loaded files and display that. Everything that can be displayed is called a *curve*. Every file, together with its derived curves is grouped into a rectangular frame in the control window (figure A.2, see also figure A.3 for a graphical presentation of the derivation relation). Every curve has a title which is displayed in bold in the control window. The different curves will be described later in section A.1.4.

A.1.2.1 File names

File names can be specified in three ways:

As a complete file path
The path (directory + filename) can be absolute or relative to the current directory.

As an extension to the base name
If the file name contains a "%" character, this will be replaced by the string entered in the Base name edittext in the display window. That way, the various files usually related by being generated from a single source can be accessed rapidly. (E.g. from some file `sound.sf` you'll have `sound.format`, `sound.sdif`, `sound.synt.sf`, etc.)

As a reference to an already loaded file
If the filename is of the form $n, where n is between 1 and 3, this file will share the data of the file n of the same type, i.e. from the frames in the control window left or right of the files frame. For example, if the name of Sound file 2 is specified as $1, pressing load (of sound file 2), will tell it to get its data from sound file 1, whatever will be loaded there, while the parameters stay independent. This allows to compare different settings of parameters for the algorithms on the same data set. See figure A.3 for a graphical presentation of the reference relation.

A.1.2.2 Loading files

Files can be loaded in two ways: individually, by clicking the Load button underneath the file name edittext in the control window, or all files in a group called the **load set** at once. All files whose box in load set is checked will be loaded when the Load set button next to the Base name in the display window is pressed. This combines nicely with the "extension to the base name" way of specifying file names, when all related files with the same base name are in the load set. Then, after changing the base name, the new files can be loaded with one mouse click.

A.1.2.3 Choosing the Display

In the lower left corner of the display window (figure A.1), 7 checkboxes with a pop-up menu can be seen. These will be referred to as the *display pop-ups*. Initially, when no files are loaded, all the 7 pop-ups will only have one entry: nothing, so no curve will be displayed. Every curve that is present (be it data loaded from a file or calculated data) will be listed in each of the 7 pop-up menus. By selecting a desired curve in one of the pop-ups, it will be displayed in the display window. Thus, up to 7 curves can be viewed simultaneously.

The 7 checkboxes are similar to the *mute* function on audio mixing desks: When switched off, the curve selected in the pop-up will not be displayed. This serves for quick switching off of a curve, when it obscures some detail in the other curves displayed, without having to take the somewhat longer way of selecting nothing in the display pop-up. Also, different combinations of curves can be selected and configured quickly.

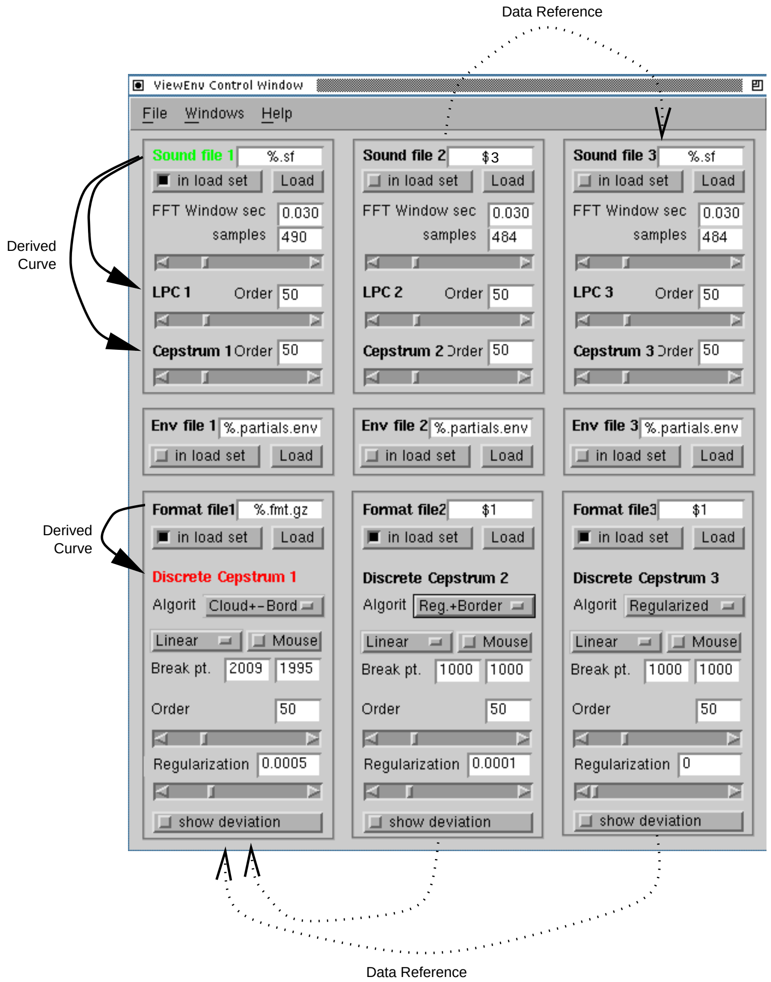

Figure A.3: The derivation and reference relations in the control window

Consistent Colour Coding

An important point is the colour that is associated in a fixed and immutable way with each slot. Thus, choosing a slot for a curve also chooses the colour with which the curve will be displayed in the axes of the display window. This colour coding has been kept consistent across all the windows of the program. When a curve is selected in a pop-up, and thus the colour chosen, the heading of the curve in the control window (figure A.2) is displayed in that colour, also. Also, the Spectrogram button in the display window, and the checkbox in the manipulatoin window (figure A.6) will change to that colour, to indicate the curve being affected by their actions (see below).

A.1.2.4 Navigation

The **navigation area** in figure A.1 shows the time-position in the data we're looking at, and offers controls to change it. The display is via three text fields, that give the position in seconds, samples, and frames. (Samples are only valid, if there is a sound file loaded.) All of these three fields can be edited, and new positions entered. Below the edit fields, there is a slider and four buttons to change the current frame. The slider can be dragged or set to a position by using the middle mouse button. Clicking on the slider arrows increments/decrements the frame by one. Clicking next to the slider handle increments/decrements by 10. The four buttons below serve the functions "go to first frame", "go to previous frame", "go to next frame", and "go to last frame".

How is the connection drawn between frames, samples and time positions? All loaded data have a time base (because time tags are stored with the file, or because it is clear how to convert position into time and vice versa), but only one data set can determine the relation between time and frame number, i.e. what time position to jump to, when a frame number is entered. This data set is called the **time master**. Whenever it changes, this is printed on the terminal. To tell which data set is the time master, priorities have been given. The data set with the highest priority will alsways be time master. The lowest priority assigned to the sound file and its derived curves. Then comes the envelope file, and the format file has highest priority. Within the maximally three data sets of one type, the one with the lowest number has highest priority.

When a time number is entered, the frame of the time master closest to that time is selected.

A.1.3 Other Functions in the Display Window

The Interaction Area

The interaction area above the navigation area in the display window figure A.1 offers miscellaneous functions. There is the edit field for the base name (see A.1.2.1), the Load set button (see A.1.2.2) which have already been explained.

Print Preview

The Export button writes the content of the display axes to a black-and-white PostScript file `export.eps` that can be printed or included in paper documents such as the spectral envelope report. All curves lose their colours in the PostScript-file. Because this is not convenient when several curves are displayed, the Preview checkbox switches to the **preview mode** for generating black-and-white curves with different line styles (dashed, dotted, etc.) to distinguish them. As with the colour, each display pop-up has a fixed line style associated to it, so different combinations of line styles can be tried out. Also, in preview mode a legend appears, which shows the name of each displayed curve along with a short example of the line style or symbol of that curve.

Position and Formant Measurement

Below the Preview and Export elements, a checkbox for position measurement with the mouse is situated. When checked, clicking in the display axes shows the position in the frequency–amplitude plane in Hz and dB. Moreover, when clicking and dragging, a special tool for manual formant measurement in spectral envelopes is available, to retrieve the frequency, amplitude and bandwidth of a formant. It measures and displays the distance of the mouse from the point first clicked on (marked with a cross) in frequency and amplitude. For frequency, two symmetric vertical lines will appear on equal distance from the starting point, as shown in figure A.4.

Formant measurement proceeds as follows: The starting point is chosen to be the peak of a formant apparent in the spectral envelope displayed. After clicking and holding down the left mouse button, the cursor is displaced to -3 dB amplitude distance. Then, the cursor is moved horizontally until the frequency lines cross the spectral envelope. The frequency distance displayed is the bandwidth of the formant. Moreover, because of the two symmetric frequency cursors, the symmetricity of the formant can be checked.

Zooming

When the mouse position checkbox is switched off (which is the default), **zoom mode** is selected. By clicking the left mouse button in the display axes, a zoom in of 50% is activated. The right mouse button zooms out. By dragging with the left mouse button, a zoom rectangle can be openend, and by double clicking the left mouse button, we return to the normal full view.

Spectrogram

Finally, a bit astray from the interaction area, the Spectrogram button at the very bottom to the left selects a spectrogram view (of the time–frequency plane, where intensity is coded by colour) of the curve that was manipulated last in the display popups.

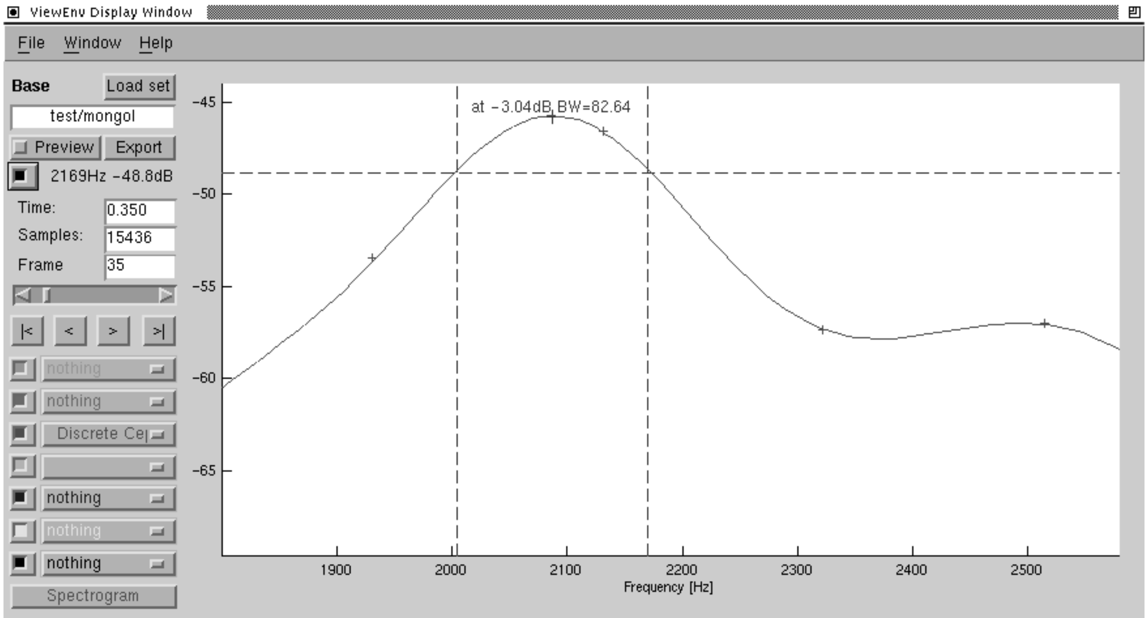

Figure A.4: Formant measurement with ViewEnv

A.1.4 Parameters and the Control Window

The control window (figure A.2) contains three identical columns of parameters for the various algorithms, grouped in boxes by the file they are derived from. The values of almost all numerical parameters are entered and displayed either directly in the edittext next to their name, or graphically by using the slider below. As usual, the arrows allow an increment or decrement of one "step" (whatever a step is for that parameter), clicking next to the slider handle changes the value by a bigger amount, and the slider handle can be dragged directly to the desired position.

The settings made can be recorded for later use by saving the window. To do this, select the Save As entry from the File menu and enter or select `control.m` as the file name, overwriting the previous settings.

The curves and their respective parameters are explained in the following sections.

A.1.4.1 Sound File

Displays the Fourier spectrum of a sound file. Besides the standard file parameters, the length of the window, from which the spectrum will be computed, can be specified both in seconds and in number of samples[1].

LPC
: Displays the spectral envelope computed by linear prediction (see section 3.2). The order parameter specifies the number of poles to use to approximate the spectrum. Higher orders yield a closer adaptation to the spectrum, but a less smooth spectral envelope.

Cepstrum
: Displays the spectral envelope computed by the continuous cepstrum method (see section 3.3). The order parameter specifies the cutoff frequency of the smoothing filter. A higher order means that more high frequency components, i.e. rapid changes, are left in the spectral envelope.

[1]If the number of samples is a power of two, the Fast Fourier Transform (FFT) algorithm will be used, otherwise a Discrete Fourier Transform (DFT) will be performed.

A.1.4.2 Envelope File

Displays the spectral envelope taken directly from a file. Only the standard file parameters are present. The two possible formats of an envelope file are:

- ASCII spectral envelope format (extension `.env`), where the first number of the file specifies the number of points n of the envelope, followed by one line for each frame, containing the time of the frame and the n amplitude values of the spectral envelope for equidistant frequencies, spaced f_s/n Hertz apart. Note that this format is obsolete and incomplete. It was only used in an early test stage and lacks the information about the sampling rate f_s. It is now superseded by the:
- SDIF spectral envelope format (extension `.sdif`), which contains the envelope in binary form, plus all necessary header information. See section 2.5 for a detailed description of the spectral envelope SDIF file format.

A.1.4.3 Format File

The format file itself contains the harmonic partials of a sound, as generated by the ADDITIVE program (see section 2.2) or HMM. Both, the ASCII format (extension `.format`), and the binary format (extension `.fmt`) are recognized, as well as files compressed with `gzip` (extensions `.format.gz` and `.fmt.gz`, respectively). Each partial is displayed as a little cross at its frequency and amplitude.

Discrete Cepstrum
Displays the spectral envelope computed by the discrete cepstrum method (see section 3.4) from the partials of the format file, which are displayed, too. The parameters are:

Algorithm Choice between Galas, Galas cloud, algorithms developed by Thierry Galas, Regularized, Reg.+Border, Reg.+-Border, regularized discrete cepstrum, Cloud, Cloud+Border, Cloud+-Border, regularized discrete cepstrum with statistical cloud smoothing, where the +Border version adds points at the low and high border of the frequency range at half the amplitude of the highest/lowest partial to force the envelope going down, and the +-Border version adds these points only when there is enough space. For more details, see the description of the algorithms in section 3.5.4.

Frequency scale Choice between

Linear: The frequencies of the partials are used as is.

Log freq: A logarithmic scale is applied to the frequencies of the partials higher than the break point to increase the resolution of low frequency details to the expense of high frequency details, which aren't perceptually important.

Log norm: Use logarithmic scale as above, which is normalized (scaled back) to cover the range of 0 to $f_s/2$ in order to avoid range errors.

Log corr: Apply normalized logarithmic scale *after* adding points in the Cloud algorithm. For the other algorithms, this is the same as Log norm.

Break point If a logarithmic frequency scale is selected, the first value specifies the break point between the linear part and the logarithmic part of the scale. The second value is the output frequency at the break frequency. For the normalized and corrected logarithmic scale, the second parameter has no influence (see section 3.5.3 for more details).

Mouse input If checked, the Break point parameter can be entered interactively using the mouse. As can be seen in figure A.5, a crosshair (the two intersecting dashed lines at the break input/output frequencies) is drawn, which can be moved by clicking the mouse or moving the mouse while a button is pressed (dragging). Also, the resulting partial frequencies are plotted as an input/output relationship (the y-axis is is to be interpreted as ranging from 0 to $f_s/2$ Hertz, for that matter), along with a grey dotted identity line for reference.

The mouse input should always be active for only one of the three discrete cepstrum curves.

Order The order parameter commands the accuracy with which the given partials are approximated by the spectral envelope. A high value will lead to a curve hitting the partials exactly, but is more demanding computationally.

Regularization The regularization factor is a constraint on the shape of the spectral envelope. It introduces an additional punishment of irregularities due to to too strong an inclination of the curve (see section 3.5.1 for more details).

The values that can be entered range from 0 (no constraint) to 1 (heavy constraint), with intermediate values of the form $d \cdot 10^n$, where $d = 0..9$ and $n = -5.. - 1$. Clicking the slider arrows results in an increment/decrement of d, with a jump to the next higher/lower power of ten (e.g. 0.0009 will increment to 0.001, then 0.002). Clicking next to the slider handle multiplies/divides by ten. This way, a wide range of regularization factors with sufficient precision can be entered conveniently.

Show deviation When this checkbox is on, the absolute deviation of the spectral envelope from the curve obtained by linear interpolation is displayed in dB. This is used for the evaluation of spectral envelope estimation as described in section 3.6 and shown in figure 3.13.

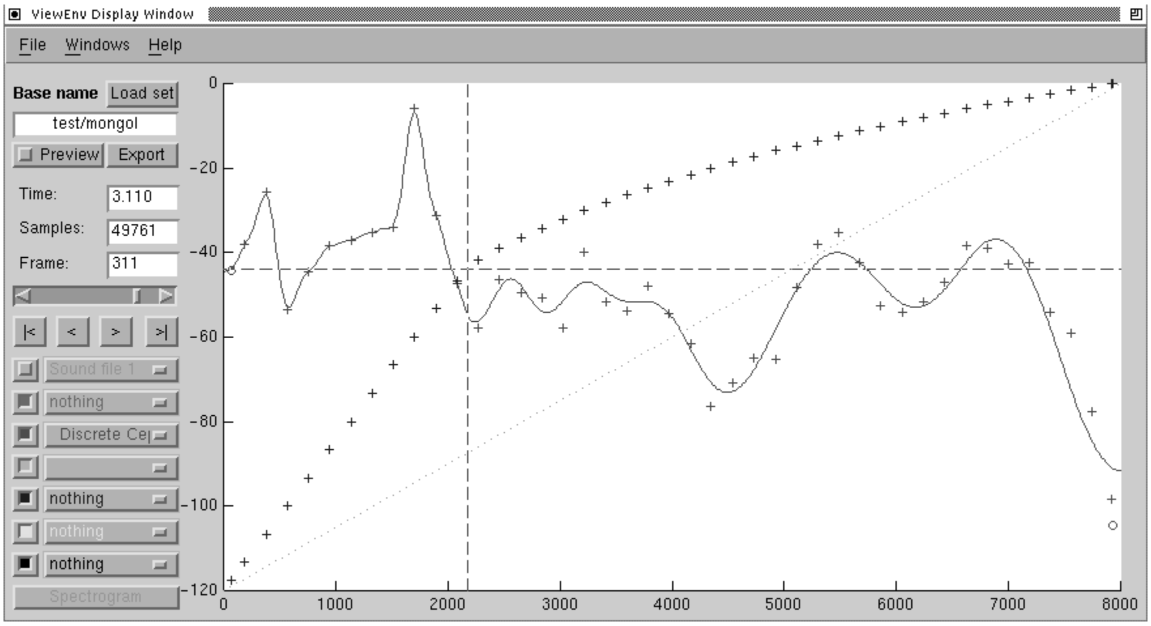

Figure A.5: Display of the logarithmic frequency scale with mouse input of the break point for logarithmic frequency scale enabled

A.1.5 The Manipulation Window

The manipulation window (figure A.6) so far serves only one manipulation, the skewing. It is described in section 5.3. The four parameter of skewing can be entered numerically in the edit fields, or by sliders. With the Skew Range checkbox, skewing can be switched on or off. Always the last curve handled in the display pop-ups is the one to be skewed, which is indicated by the colour of the Skew Range checkbox. Figure 5.8 shows an example of the skewing manipulation.

A.1.6 The Evaluation Window

The evaluation window serves a rather specific task (to the spectral envelope project), and the user interface is therefore not very elaborate. The task is the evaluation of a large corpus of audio data, described in section 3.6. After the corpus has been analysed, the result file giving the maximum deviations for each frame can be loaded and displayed, as can be seen in figure A.7. The upper line

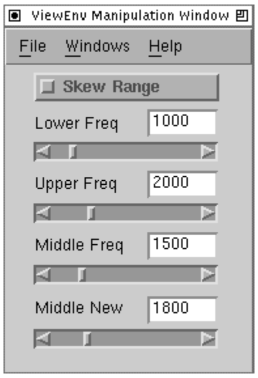

Figure A.6: The ViewEnv manipulation window.

shows, for each frame, the maximum absolute deviation between the estimated spectral envelope and the linear interpolation. The lower line shows the amplitude at which the maximum deviation occured. The text display below the axes shows the frame number at which the cursor is positioned (frame), maximum deviation for that frame (maxdiff), amplitude value (env), and frequency (freq) at which the maximum difference occured. By means of a frequency cursor, and zooming into the interesting parts (toggle zoom mode/cursor mode by pressing 'z'), the peaks in the deviation curve can be examined (see figure A.8). If it is a significant peak (if it is not at a very low amplitude or at the borders of the frequency range) we can jump directly to the display of the frame producing that error in the display window by pressing 'd'. By pressing 'f', these peaks are filtered out beforehand.

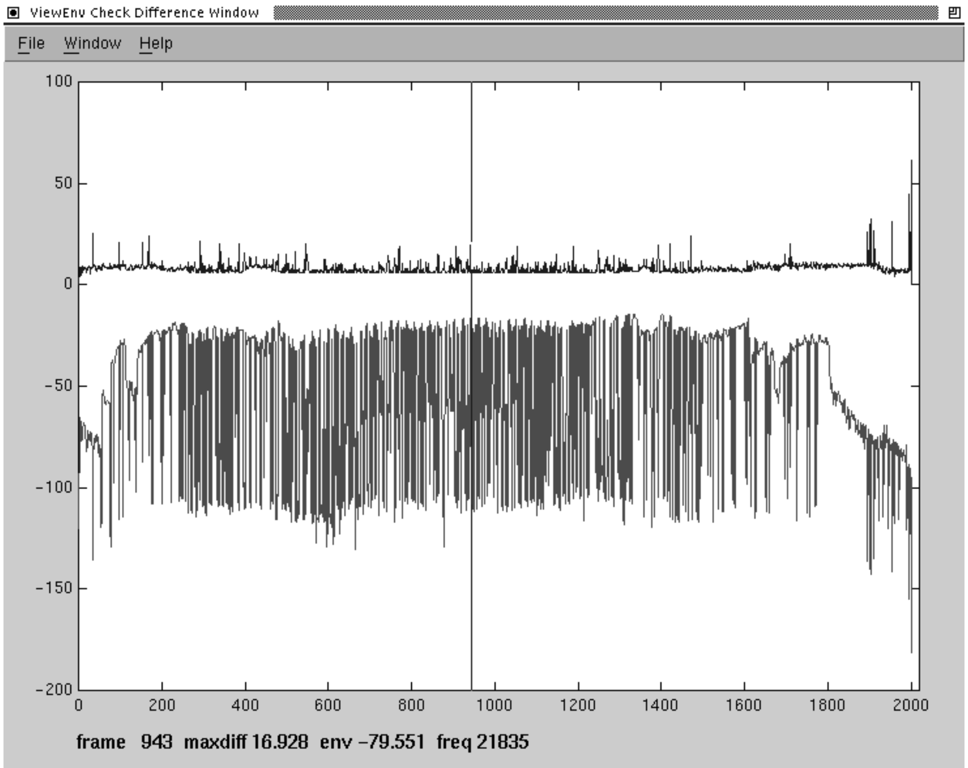

Figure A.7: Evaluation window, full view

A.2 Architecture

This section will briefly explain the internal structure of the envelope viewing application ViewEnv to allow for easier extension or modification.

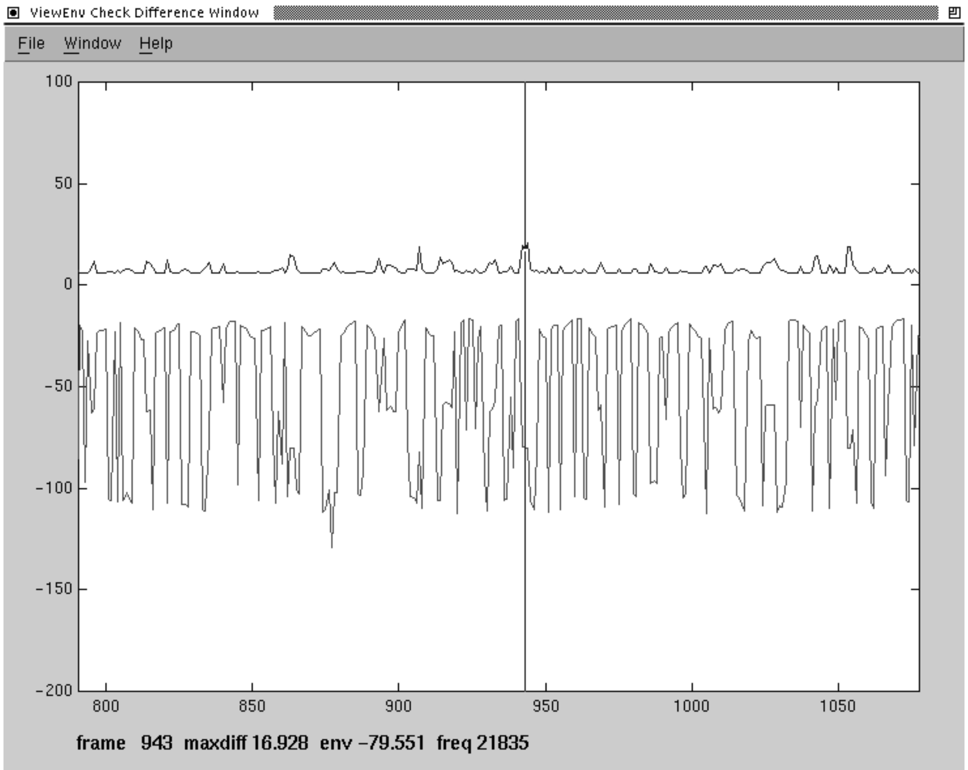

Figure A.8: Evaluation window, zoomed in. Here, single peaks of high deviation can be individually examined.

Note that the term *derived* has already been used in section A.1 in the sense of *computationally derived*. In this section, as in object-oriented programming in general, it is used in both this sense and in the sense of *derived by inheritance* in descriptions of relations between classes.

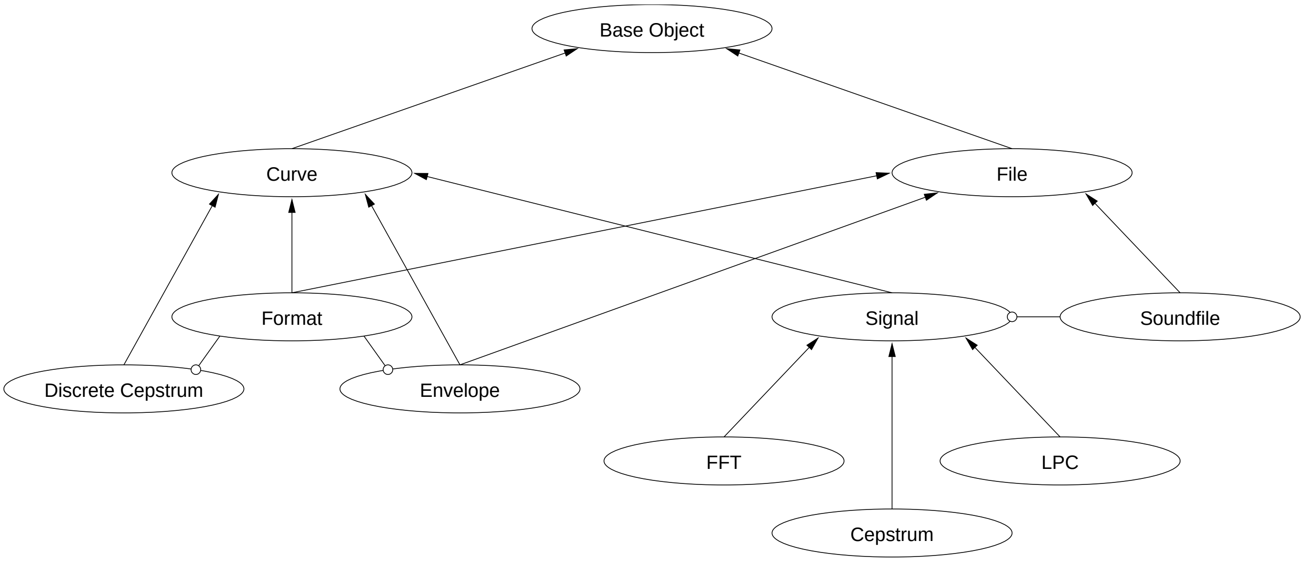

Figure A.9: Optimal class design for ViewEnv.

The intended object-oriented class hierarchy is shown in figure A.9. The different tasks of a curve (something to be displayed) and a file (loaded data) are separated, since some curves don't own their proper data, but are derived from data loaded in a different class. This type of derivation is expressed as a usage relation, e.g. between *Discrete Cepstrum* and *Format*. Those classes who directly display the loaded data use multiple inheritance from *Curve* and *File*, e.g. *Envelope*.

Unfortunately, the realization of the principles of object-oriented programming is far from perfect in the MATLAB programming language. Indeed, it is quite cumbersome to write classes, e.g. the inheritance has to be "hand made" and a separate directory is needed for each class, with a separate source file for each method. Thus, for reasons of feasibility, the class design had to be stripped down to the one shown in figure A.10.

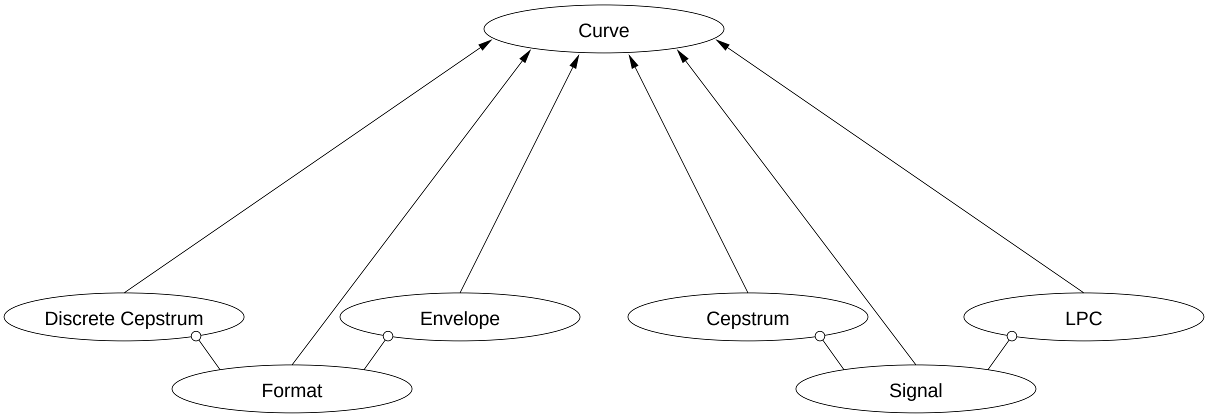

Figure A.10: Feasible class design for VIEWENV.

The new basic class *Curve* is a fusion of the curve and file functionalites, although some of them will not be used by certain derived classes. The *Curve* class handles all commonalities of the different derived classes, such as the callbacks from the file handling buttons of the user interface.

To define a new class derived from *Curve*, all that has to be done is to write a constructor which calls the *Curve* constructor, provide a `load.m` method, a `paint.m` method, and possibly a `callback.m` method, to handle user interface elements to input parameter values.

Bibliography

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin–Cummings, Redwood City, Calif., 2nd edition, 1994.

[Cha95] Y. T. Chan. *Wavelet Basics.* Kluwer Academic Publ., Boston, 1995.

[COM97] O. Cappé, M. Oudot, and E. Moulines. Spectral Envelope Estimation using a Penalized Likelihood Criterion. In *IEEE ASSP Workshop on App. of Sig. Proc. to Audio and Acoust.*, Mohonk, October 1997.

[CY96] John E. Clark and Colin Yallop. *An Introduction to Phonetics and Phonology.* Blackwell, Oxford, 1996.

[DDPZ94] François Dechelle, Maurizio DeCecco, Miller Puckette, and David Zicarelli. The IRCAM "Real-Time Platform": Evolution and Perspectives. In *Proceedings of the International Computer Music Conference (ICMC)*, 1994. Available online[2].

[DGR93] Ph. Depalle, G. Garcia, and X. Rodet. Tracking of Partials for Additive Sound Synthesis Using Hidden Markov Models. In *IEEE Trans.*, pages 225–228, April 1993. Abstract[3].

[DGR94] Philippe Depalle, Guillermo García, and Xavier Rodet. A Virtual Castrato (!?). In *Proceedings of the International Computer Music Conference (ICMC)*, 1994. Available online[4].

[Dog95] Grzegorz Dogil. Phonetic correlates of word stress. *AIMS Phonetik (Working Papers of the Department of Natural Language Processing)*, 2(2), 1995. Contents[5].

[FRD92] Adrian Freed, Xavier Rodet, and Phillipe Depalle. Synthesis and Control of Hundreds of Sinusoidal Partials on a Desktop Computer without Custom Hardware. In *ICSPAT*, 1992. Available online[6].

[Gar94] Guillermo García. *Pm: A library for additive analysis/transformation/synthesis*, July 1994. Available online[7].

[GBM+96] R. Gribonval, E. Bacry, S. Mallat, Ph. Depalle, and X. Rodet. Analysis of Sound Signals with High Resolution Matching Pursuit. In *Proceedings of the IEEE Time–Frequency and Time–Scale Workshop (TFTS)*, 1996. Available online[8].

[GDR+96] R. Gribonval, Ph. Depalle, X. Rodet, E. Bacry, and S. Mallat. Sound Signal Decomposition using a High Resolution Matching Pursuit. In *Proceedings of the International Computer Music Conference (ICMC)*, August 1996. Abstract[9] PostScript[10].

[2] http://mediatheque.ircam.fr/articles/textes/Dechelle94a/

[3] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ICASSP93HMM/ICASSP93HMMabstract.html

[4] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ICMC94CASTRAT

[5] http://www.ims.uni-stuttgart.de/phonetik/aims.html

[6] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ICSPAT92/ICSPAT92.html

[7] http://www.ircam.fr/equipes/analyse-synthese/pm/

[8] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/TFTS96/tfts96.ps.gz

[9] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ICMC96HRMP/abstract.txt

[10] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ICMC96HRMP/ICMC96HRMP.ps.gz

[GJM91] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice–Hall, Englewood Cliffs, NJ, 1991.

[GR90] Thierry Galas and Xavier Rodet. An Improved Cepstral Method for Deconvolution of Source–Filter Systems with Discrete Spectra: Application to Musical Sound Signals. In *Proceedings of the International Computer Music Conference (ICMC)*, Glasgow, September 1990.

[GR91a] Thierry Galas and Xavier Rodet. Generalized Discrete Cepstral Analysis for Deconvolution of Source–Filter Systems with Discrete Spectra. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, New York, October 1991.

[GR91b] Thierry Galas and Xavier Rodet. Generalized Functional Approximation for Source–Filter System Modeling. In *Proc. Eurospeech*, pages 1085–1088, Geneve, 1991.

[Hen98] Nathalie Henrich. *Synthèse de la voix chantée par règles*. IRCAM, Paris, France, July 1998. Rapport de stage D.E.A. Acoustique, Traitement de Signal et Informatique Appliqués à la Musique.

[Hub97] Barbara Burke Hubbard. *The World According to Wavelets: The Story of a Mathematical Technique in the Making*. A K Peters Ltd, 1997.

[Jac95] Ivar Jacobson. *Object-Oriented Software Engineering: a Use Case driven Approach*. Addison–Wesley, Wokingham, England, 1995.

[Mal97] Stephane Mallat. *A Wavelet Tour of Signal Processing*. AP Professional, London, 1997.

[Mel97] Jason Meldrum. The Z–Transform, 1997. Online tutorial[11].

[MG80] J.D. Markel and A.H. Gray. *Linear Prediction of Speech*. Springer, 1980.

[MZ92] S. Mallat and S. Zhong. Characterization of Signals from Multiscale Edges. *IEEE Trans. Pattern Anal. Machine Intell.*, 40(7):2464–2482, July 1992.

[Nag90] Manfred Nagl. *Softwaretechnik: methodisches Programmieren im Großen*. Springer compass. Springer, Berlin, 1990.

[Opp78] Alan V. Oppenheim, editor. *Applications of Digital Signal Processing*, chapter Digital Processing of Speech, pages 117–168. Prentice–Hall, 1978.

[OS75] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice–Hall, 1975.

[Puc91a] Miller Puckette. Combining Event and Signal Processing in the MAX Graphical Programming Environment. *Computer Music Journal*, 15(3):68–77, Winter 1991. Available from[12].

[Puc91b] Miller Puckette. FTS: A Real-Time Monitor for Multiprocessor Music Synthesis. *Computer Music Journal*, 15(3):58–67, Winter 1991. Available from[13].

[RBP$^+$91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice–Hall, Englewood Cliffs, NJ, 1991.

[RDG95] Xavier Rodet, Philippe Depalle, and Guillermo García. New Possibilities in Sound Analysis and Synthesis. In *ISMA*, 1995. Available online[14] PostScript[15].

[RH91] Stuart Rosen and Peter Howell. *Signals and Systems for Speech and Hearing*. Academic Press, London, 1991.

[11] `http://www.spd.eee.strath.ac.uk/~interact/ztransform`
[12] `http://man104nfs.ucsd.edu/~mpuckett/`
[13] `http://man104nfs.ucsd.edu/~mpuckett/`
[14] `http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ISMA95/ISMA95.html`
[15] `http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ISMA95/ISMA95.ps.gz`

[RL97] Xavier Rodet and Adrien Lefevre. The Diphone program: New features, new synthesis methods and experience of musical use. In *Proceedings of the International Computer Music Conference (ICMC)*, September 1997. Abstract[16], PostScript[17].

[RM69] J.C. Risset and M.V. Mathews. Analysis of musical-instrument tones. *Physics Today*, 22(2):23–30, February 1969.

[Rob98] Tony Robinson. Speech Analysis, 1998. Online tutorial[18].

[Rod84] Xavier Rodet. Time-Domain Formant-Wave-Function Synthesis. *Computer Music Journal*, 8(3):9–14, Fall 1984. reprinted from [Sim80].

[Rod97a] Xavier Rodet. Musical Sound Signals Analysis/Synthesis: Sinusoidal+Residual and Elementary Waveform Models. In *Proceedings of the IEEE Time–Frequency and Time–Scale Workshop (TFTS)*, August 1997. Abstract[19], PostScript[20].

[Rod97b] Xavier Rodet. *The Additive Analysis–Synthesis Package*, 1997. Available online[21].

[RPB84] Xavier Rodet, Yves Potard, and Jean-Baptiste Barrière. The CHANT–Project: From the Synthesis of the Singing Voice to Synthesis in General. *Computer Music Journal*, 8(3):15–31, Fall 1984.

[RPB85] Xavier Rodet, Yves Potard, and Jean-Baptiste Barrière. CHANT: de la synthèse de la voix chantée à la synthèse en général. *Rapports de recherche IRCAM*, 1985. Available online[22].

[Sim80] J. C. Simon, editor. *Spoken Language Generation and Understanding*. D. Reidel Publishing Company, Dordrecht, Holland, 1980.

[Sof97] Rational Software. Unified modeling language, version 1.1. Online documentation[23], September 1997.

[TAW97] Keith A. Teague, Walter Andrews, and Buddy Walls. Enhanced Modeling of Discrete Spectral Amplitudes. In *IEEE Workshop on Speech coding*, Pocono Manor, September 1997.

[UAE93] Michael Unser, Akram Aldroubi, and Murray Eden. B–Spline Signal Processing: Part I—Theory. In *IEEE Transactions on signal processing*, volume 41, pages 821–833, 1993.

[Utt93] Ian A. Utting. *Lecture Notes in Object-Oriented Software Engineering*. University of Kent at Canterbury, Canterbury, UK, 1993.

[vH13] Hermann L. von Helmholtz. *Die Lehre von den Tonempfindungen: als physiologische Grundlage für die Theorie der Musik*. Vieweg, Braunschweig, 6th edition, 1913.

[vH54] Hermann L. von Helmholtz. *On the Sensations of Tone as a Physiological Basis for the Theory of Music*. Dover, New York, 1954. Original title: [vH13].

[Vir97] Dominique Virolle. *La Librairie CHANT: Manuel d'utilisation des fonctions en C*, April 1997. Available online[24].

[Vir98] Dominique Virolle. *Sound Description Interchange Format (SDIF)*, January 1998. Available online[25].

[16] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ICMC97/ICMC97DiphoneAbstract.html
[17] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/ICMC97/ICMC97Diphone.ps.gz
[18] http://svr-www.eng.cam.ac.uk/~ajr/SA95/SpeechAnalysis.html
[19] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/TFTS97/TFTS97abstract.html
[20] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/TFTS97/TFTS97.ps.gz
[21] http://www.ircam.fr/equipes/analyse-synthese/additive/index-e.html
[22] http://www.ircam.fr/equipes/analyse-synthese/listePublications/articlesRodet/CHANT85/Chant85Tout.html
[23] http://www.rational.com/uml/documentation.html
[24] http://www.ircam.fr/equipes/analyse-synthese/libchant/index.html
[25] http://www.ircam.fr/equipes/analyse-synthese/sdif/index.html

[WRD92] Peter Wyngaard, Chris Rogers, and Philippe Depalle. *UDI 2.1—A Unified DSP Interface*, 1992. Available online[26].

[WSR98] Marcelo M. Wanderley, Norbert Schnell, and Joseph Rovan. ESCHER—Modeling and Performing composed Instruments in real-time. In *IEEE Systems, Man, and Cybernetics Conference*, October 1998. To be published.

[Zwi82] Eberhard Zwicker. *Psychoakustik*. Springer, 1982.

[26] `http://www.ircam.fr/equipes/analyse-synthese/udi/udi.html`

Index

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

____

(Diemo Schwarz)