VowSynth:
A Synthesizer of Vowel Sounds based on Additive Synthesis

Esther Guerra i Novau
Audiovisual Institute, Pompeu Fabra University
Rambla 31, 08002 Barcelona, Spain
eguerra@iua.upf.es http://www.iua.upf.es

Abstract

This article presents the design and implementation of a non real-time voice synthesizer based on the analysis of a soprano voice singing the five spanish vowels. The analysis is based on the SMS technique and the synthesis uses additive synthesis.

1 Introduction

The synthesis of the singing voice has been investigated from two different fields, speech and music, and following mainly two different models: physical models, based on sound productions mechanisms, and spectral models, based on sound perception mechanisms. (For a general overview of singing voice synthesis methods see [1]).

One of the first methods to approach singing voice synthesis was LPC, which can be considered a mixture of both types of models. Other early approaches were mainly based on spectral models, like the vocoder, FM, CHANT, or MUSSE. SPASM can be included in physical models where the human vocal tract is modeled as a digital waveguide.

SMS (Spectral Modeling Synthesis) [2] is also a spectral model, and the basis for our synthesizer. A sound is analyzed with SMS and decomposed in a set of sinusoids and a spectral residual. The analysis procedure detects partials by studying the time-varying spectral characteristics of a sound and represents them with sinusoids. These partials are then subtracted from the original sound and the remaining residual can be approximated as a stochastic signal, for example, filtered white noise. Two recent methods which also use spectral methods are the software designed by K. Lomax [3] and LYRICOS [4].

In our first approximation to a voice synthesizer some restrictions have been made. On one hand, only vowels will be synthesized; consonants present problems that go beyond the current work. Because of this restriction the synthesizer will not use the residual part of the sound obtained by SMS, thus using SMS as an additive synthesis system. On the other hand, to also avoid some other problems, we will not deal with vibrato, thus analyzing and synthesizing voices without it.

Our software synthesizer has been developed under Windows'95, using Matlab 5.2, Visual C++ 5.0 and the library of SMS classes developed by the Music Technology Group of the Audiovisual Institute.

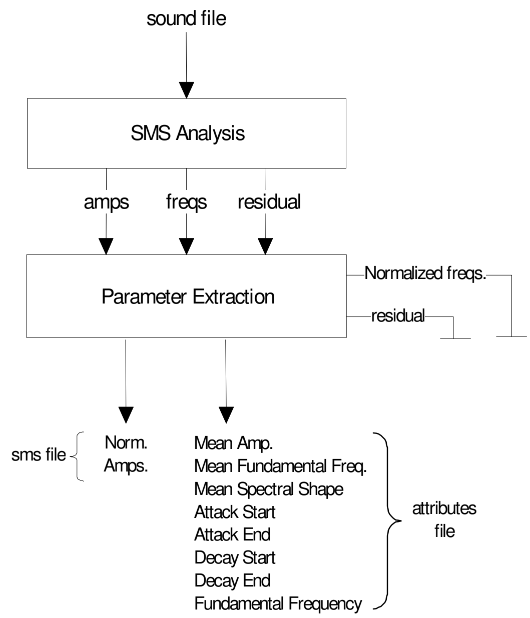

Figure 1. Diagram of the analysis process.

2 Analysis

The analysis of the original sounds is divided in two consecutive phases. In the first one, the basic SMS technique is applied [5] and in the second one, high level parameters are extracted from the previous information with some utilities written for Matlab [6]. In *Figure 1* a diagram of the analysis is shown.

2.1 SMS Analysis

In the SMS technique, initially developed by Xavier Serra [2], a sound is decomposed in deterministic and residual parts, and both will be used in the Matlab analysis. In the synthesis, the residual will be set aside for simplicity and because the human voice can be viewed as essentially a harmonic signal, where the residual is not essential in vowels (not as much as it would be, for example, in the flute, which has a great deal of residual in its sound).

From the basic SMS analysis, the sinusoidal data, and residual will be calculated and saved in a .m file, which is simply a text file compatible with Matlab.

2.2 Matlab Analysis

From the .m file obtained, the process continues with the extraction of high-level attributes which characterize the sound. Utilities based on Matlab with dll's compiled in VC++ 5.0 and developed at the IUA [6] have been used. This software reads the data written in the .m file (amplitudes, frequencies and residual) and processes it, extracting high-level parameters. The results are:

- *Mean amplitude*: Average amplitude of the steady state of the sound.
- *Mean fundamental frequency*: Average fundamental frequency of the steady state of the sound.
- *Mean spectral shape*: Average spectral shape of the steady state of the sound.
- *Begin and end of the attack and release*.
- *Normalized amplitudes.*
- *Normalized frequencies.*

Normalized amplitudes and frequencies are stored in a new .sms file, and the rest in an attributes file. This work was done before the new file format for SMS was developed, in which the attributes are also stored in the same file.

3 Synthesis

To design this synthesizer, I started with the C++ classes developed by the Music Technology Group of the IUA [5], modifying and adapting them to the voice synthesizer.

The synthesizer is initialized loading a set of analyzed sounds. Then, from a score, the user controls and modifies the analyzed sounds, so the desired notes are generated after the additive synthesis stage. In *Figure 2* a diagram of the synthesis is shown.

3.1 Initialization

The first thing the synthesizer does is to open all the .sms files (one for each analyzed sound) and load them in memory as a data base of analysis data. The current version of the synthesizer uses the 5 Spanish vowels: a, e, i, o, u. Then, the score will select which one is going to be used for each event. This step will be very important when the synthesizer works in real time [10].

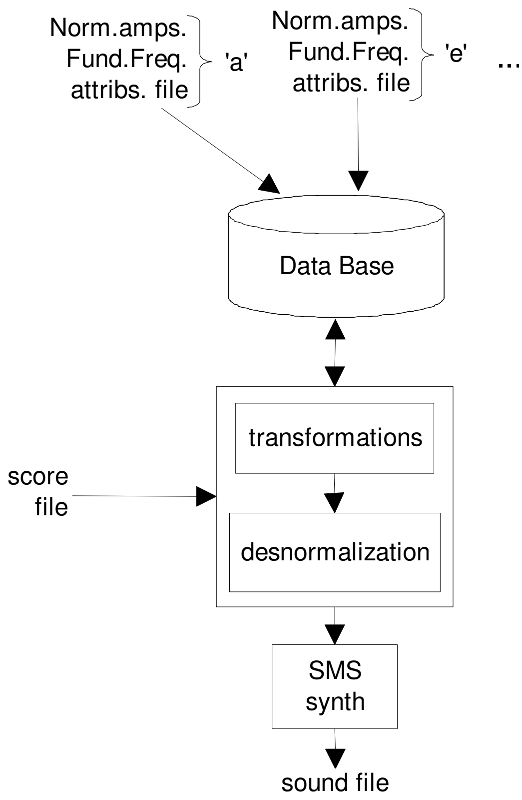

Figure 2. Diagram of the synthesis process.

3.2 Control

The interface of control and communication with the synthesizer is a text file, a score where the control parameters are specified. This interface is quite simple, it's the one used with the application developed by Bonada [5], a text file with a control event written in every line. In parallel to this project, a new score file format is being developed that uses a more advanced interface [7].

The original synthesis software had a long list of control parameters. To this list I added new parameters that were needed for the synthesizer:

- Tempo, Meter
- STime, Sduration, Spitch, SetPitch

- KeepSpectralShape (bool)
- Vowel [a,e,i,o,u]
- Articulation (bool)
- ArticulationType (1,2)
- ArticulationDuration (sg)
- DecayDuration (sg)
- Dynamics (f, m, p)
- MeanAmp (dB)

In the next section a little explanation of the more important parameters will be done.

3.3 Using and Processing Parameters

Frame by frame denormalization and transformation will be controlled by the score. The first parameter to be read is *Vowel*, which specifies the vowel to be synthesized [a, e, i, o, u]. Once it is identified, its spectral shape is loaded. The denormalization of the harmonic l in frame number k is done adding the mean amplitude to the denormalized amplitude (which is stored in the .sms file):

$$Amp'_k(l) = NormAmp_k(l) + A_{mean}$$

where A_{mean} is the mean amplitude of all the harmonics in the steady state, k is the frame number and l the harmonic number, with values among 0 and (*numharm–1)*, being *numharm* the greatest number of harmonics.

At this moment, the sound is not vocalic yet, since all the harmonics still have the same amplitude. To become a concrete vowel, the spectral shape amplitude must be applied to each harmonic in every frame.

To obtain the denormalized frequency, an easier process is as follows. We approximate the voice as perfectly harmonic, thus the l-th harmonic can be generated multiplying the 1st harmonic (fundamental frequency) by l. So, to denormalize the frequency of the harmonic l in frame k, the fundamental frequency should be multiplied by k.

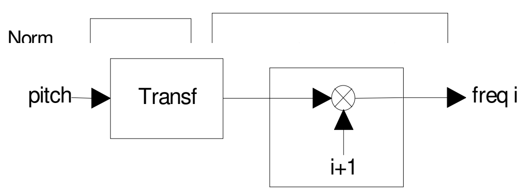

Figure 3. Diagram of the amplitude and frequency denormalization in a frame.

3.3.1 Attack

In the attributes file, where there is each note's information, the parameter *AttackStart* is read, and it shows which is the note's starting frame (that is, where it is the first harmonic different from zero). When a note is to be synthesized, it is from this frame, because if the synthesis would begin from frame 0, there would be a silence at the beginning of the sound that is useless, and it should be avoided.

3.3.2 Decay (or release)

A restriction in terms of note duration is made: the synthesized note will never be longer than the analyzed one. When a note has a concrete duration and it is being synthesized, when the end is reached the note must be handled in such a way that it doesn't finish abruptly, the right release must be applied. Our release is a generated amplitude envelope going from 1 to 0 in the temporal domain. An exponential amplitude envelope could be applied, but a straight line is chosen because perceptually the difference is minimum. *DecayDuration* is a parameter indicating the duration (in seconds) of this decay. By default, decay duration is $t_d = 0.1$ sec.

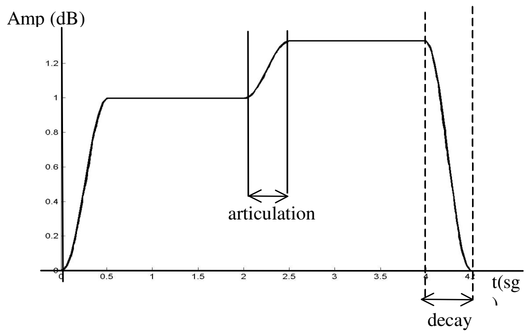

Figure 4. Amplitude envelope in time domain. Decay and articulation are specified.

3.3.3 Articulation

Maybe this is the most delicate point in all the process. Articulation is the transition between two successive notes (two events), and this transition includes amplitude, frequency and vowel change. In the score the articulation is controlled by three parameters: *Articulation* (a value of 0 implies no articulation and 1 that there is articulation), where by default its value is 0, *ArticulationDuration* denotes how much time it must last in seconds (by default, the value is 0.1) and *ArticulationType*, that may be 1 o 2, depending on what kind of articulation we desire. A value of 1 implies linear interpolation, in amplitudes and in frequencies, whereas a value of 2 implies a linear interpolation in amplitude but sinusoidal in frequency. By default, the value is 2. The current program only accepts the two extreme values, but ideally it should accept any value in between, where 1 would be a *legato* and 2 *staccato*.

The articulation also includes a spectral change. The natural amplitude change of the articulation is not the same for all the harmonics. For example in the release the level of the highest harmonics drops more than that the one of the lowest ones.

3.3.4 Mean Amplitude

MeanAmp is a parameter used to change the mean level (in dB's) of the sound. When a value is given to *MeanAmp,* the value called *mean amplitude* (see section 3.2) is changed by this new one, so if *MeanAmp* is bigger than the original amplitude, the synthesized sound will be louder than the original one.

3.3.5 Dynamics

In music, the difference between a *pp* and a *ff* is not only the amount of dB's, but there are also differences in their spectra. In a *pp*, the spectral tilt is steeper, that is, when somebody sings loudly, the high harmonics are proportionally louder that when somebody sings softly. This effect has been studied and quantified by Sundberg [8] and Rodet [9]. We have implemented 3 dynamic levels: *f*, *m* and *p* (strong, medium, soft), each one with a spectral slope steeper than the previous one. There is also another parameter -*x*- that allows to adjust the tilt directly, useful to adjust the dynamic level of the analyzed sound.

4 Conclusions

The initial goal of this project was to build a singing vowel synthesizer. As the project went on, some restrictions had to be done, in order to make it feasible. So, there's still some work to do, like:

- Update the SMS analysis software to last version.
- Vibrato analysis and synthesis.
- Use of loop points to make notes as long as desired.
- Addition of residual to vowels.
- Incorporate consonants.
- Synthesis in real time, make use of the SMS software [10].

As a conclusion, we could say that the initial aims have been accomplished, but there's still work to be done. This is the first step of a project that has to go further, but the results obtained show the validity of this approximation to the problem.

5 Acknowledgments

I'd like to thank everybody in the Music Technology Group of the IUA and also Joana Clotet for her voice.

References

[1] P. Cook. "Speech and Singing Synthesis Using Physical Models: Some History and Future Directions". *Greek Physical Modeling Conference*, 1995.

[2] X. Serra. "Musical Sound Modeling with Sinusoids plus Noise". C. Roads and others, editors, *Musical Signal Processing,* Swets & Zeitlinger Publishers, 1997.

[3] K. Lomax. *The Analysis and Synthesis of the Singing Voice*. Ph. D. Thesis, Oxford University, 1996.

[4] M. Macon and others. "A Singing Voice Synthesis System based on Sinusoidal Modeling". Proc. *ICASSP,* 1997. [available at http://www.cse.ogi.edu/CSLU/publications/abstracts/icassp97/macon.html]

[5] J. Bonada. *Desenvolupament d'un Entorn Gràfic per a l'Anàlisi, Transformació i Síntesi de Sons Mitjançant Models Espectrals.* Graduate Thesis, ETSETB, Polytechnical University of Catalonia, 1997. [available at http://www.iua.upf.es]

[6] J. Soler. *Obtenció dePparàmetres d'Alt Nivell Aptes per a Realitzar Modificacions Musicals a Partir d'una Tècnica d'Anàlisi de Sons Basada en Models Espectrals*. Graduate Thesis, ETSETB, Polytechnical University of Catalonia, 1998. (to be published)

[7] X. Amatriain. "A Musical Data Definition Language and Class Structure for a Spectral Modeling Based Synthesizer". *Proceedings of the Digital Audio Effects Workshop*, 1998.

[8] J. Sundberg. *The Science of the Singing Voice.* Debalk, IL; Northern Illinois University, 1987.

[9] X. Rodet and G. Bennett. "Synthesis of the Singing Voice". M. Mathews and J. Pierce, editors, *Current Directions in Computer Music Research,* Cambridge MA.: MIT Press, pp 19-44, 1989.

[10] A. Loscos and E. Resina. "SMSPerformer: a Real-Time Synthesis Interface for SMS". *Proceedings of the Digital Audio Effects Workshop*, 1998.