STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

# "PADSYNTH" SOUND SYNTHESIS ALGORITHM

PAUL NAŞCA

Abstract. The sound synthesizer has changed the way the musical sound is produced by offering nearly unlimited amount of musical timbres. Unfortunately, many of the current synthesizers lack the subjective "warmth" of the acoustic instruments. The current paper describes a new algorithm which produces aesthetically pleasant sounds. There is also described a simple approach for synthesizing sounds, optimized for the generation of the ensemble sounds. This approach can be used in other sound synthesis/sound processing algorithms to acheive a great degree of perceived quality.

## 1. Introduction

Nowadays there are many sound synthesizers and sound synthesis algorithms. Most of the synthesis algorithms are based on a handfull of approaches, like addition of many simple components or by simulating with the mathematical equations the acoustic instruments.

The music theory considers the musical sound as being composed of many sine waves with a frequency multiple of a fundamental frequency. For example, the note A with a fundamental frequency of 220 Hz has the harmonics with the frequencies of 440 Hz, 660 Hz, 880 Hz, and other frequencies multiples of 220 Hz. Each of these sine-wave components are called "harmonics".

The most used approach to generate harmonics is based on amplitude modulated by a stochastic process (called "micromodulations" [3]). However, using this approach it is very difficult to simulate the ensembles of instruments or choirs.

It seems it is complicated to use modulation because that the speed of modulation needs to be increased on higher harmonics [4] in "Simulating the ensemble effect".

Received by the editors: February 1, 2011.

2010 *Mathematics Subject Classification.* 00A65.

1998 *CR Categories and Descriptors.* H.5.5 [**Information Systems**]: Sound and Music Computing – *Signal analysis, synthesis, and processing*.

*Key words and phrases.* sound synthesis, harmonics, fast Fourier transform.

This paper describes a simple sound synthesis approach solves the above mentioned problem.

This approach adds another level between "harmonic"/"overtone" and the sine component of the sound. The proposed algorithm models a difficult category of sounds like those generated by the ensembles of instruments and choirs.

The basic idea is to consider the harmonics as being a narrowband signal instead of simple (or frequency/amplitude modulated) sine components.

The frequency-domain property of choirs is described in Jordi Bonada[1] paper, in which he tries to describe a technique to transform a solo sound to "unison choir" sound. His paper acknowledges that using sine components with pure amplitude modulation is not a good idea, because at higher frequency the harmonics overlap. His paper does not explain the cause, the how or the why, for example, the harmonics become wider and wider on higher frequencies according to a linear function. Also, even if he acknowledges that higher harmonics need to be smoothed in frequency, he does not explain how the slight detuning of lower sine components become a continuous frequency band on higher frequencies.

The algorithm described here considers that the harmonics become wider and wider (according to linear function) on higher frequencies, until they merge together into a single continuous band. If the fundamental frequency has a certain width (measured in Hz), the Nth harmonic contains multiples of the each sine frequency component of it. This causes the harmonic to have a bigger bandwidth.

Another paper [5] which describes that voices in a choir do not have exactly the same frequency (called "pitch scatter") and that voice's frequency changes very fast for the same perceived note (called "voice flutter").

In this paper I will present the algorithm "PADsynth" based on Ternstrom [5] definition of "pitch scatter" which - for more clarity - I will call from now as "bandwidth of each harmonic".

The first requirement of this algorithm is a frequency distribution of a single harmonic (usually a Gaussian distribution). Using this distribution, the desired harmonic are added to a large array which represent the signal amplitudes in the frequency domain. After this, it is done a single inverse Fourier transform (IFFT) by considering the phases of the signal as being at random. A long sample will result (usually few seconds long) which can be played at different speeds in order to obtain the desired pitch. This algorithm has been included in several software synthesizers, because of its simplicity and especially because of the high quality of musical instruments generated by it. It is also mentioned in other paper[2] where the algorithm is used in a low cost system-on-chip embedded system.

## 2. What PADsynth algorithm does

Most people perceive the sounds of ensembles, choirs and the sounds produced by slight detuning of instruments as being "pleasant" or "warm". But it is often believed that the pitched part of the instruments' timbre are composed only by a fundamental frequency and harmonics (another pure sine signals with different frequencies, usually on multiple frequencies).

The approach presented in this paper considers the harmonics (and overtones, in general) as being a "collection" of many sine components with very close frequencies and the phases of these components are random. This randomness of the phases makes the instrument have qualities that resemble the natural/acoustic instruments and ensembles.

Due to this result, it must be defined a parameter of the sound, called "the bandwidth of each harmonic" (BoEH) which represents the frequency spread of the sine components into each harmonic. For example, if we define BoEH as the highest frequency minus lowest frequency from a certain harmonic, for note A-4 (440 Hz) if there are sine components of 435Hz, 438Hz, 442 Hz and 445 Hz the BoEH is 10 Hz. If one harmonic contains many sine components, there are other ways to define BoEH, for example as the standard deviation of frequency values of the components.

In natural ensembles one of the property of the BoEH is that it is proportional to the overtone's frequency. For example the A-4 has harmonics 880 Hz and 1320 Hz, the bandwidth of them will be 20Hz and 30Hz. As a result, BoEH can be described as a single number: the bandwidth of the fundamental frequency, expressed in cents (where 1 cent is one hundredth of a halftone). From this parameter, we can express the BoEH in Hz for each overtone.

From this approach it is not difficult to realise what the implication of the ensembles and detuning on higher harmonics are. Also, BoEH approach predicts that ensembles of pitched sounds with many harmonics(like a huge choirs of singers) can result a hissing sound on the high part of the spectrum. This prediction was confirmed by analyzing the recording of a large number of singers when they pronounced different wowels ("10,000 Voices, The World Choir" - EMI Classics, 1992). The next spectrograms (Fig.2) shows vowel "A" analyzed using PRAAT [6] software.

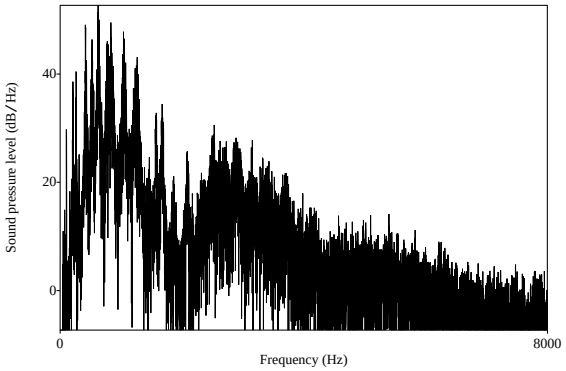


FIGURE 1. Spectrum of "A" vowel

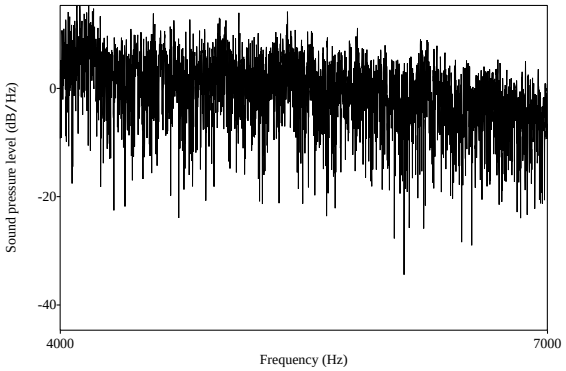


FIGURE 2. Closeup of the spectrum, to show the "hissing" part of the vowel

If the wovel "A" from this recording is passed through a highpass filter, the "hissing" becomes very noticeable for the listener.

## 3. How PADsynth algorithm works

The PADsynth algorithm generates subjectively pleasant sounds, even if its idea is more simple than of the other sound synthesis algorithms. It generates a perfectly looped wave-table sample which can be played back on different speeds in order to obtain the desired musical pitch. It easily generates sounds of ensembles, choirs, metallic sounds (bells) and many other types of sound. Also, this algorithm is a direct consequence of the BoEH approach. It was created by me at the end of 2003, and it was released on-line with example

implementations [7] under Public Domain. One of the interesting property of PADsynth is that it generates the "hissing" sound for large BoEH, similar to the "hissing" discussed before.

3.1. **PADsynth steps.** The basic steps of this algorithms are:

(1) Make a very large array which represents the amplitude spectrum of the sound (default all values are zero)
(2) Generates the distribution of each harmonic in frequency and add it to the array
(3) Randomize the phases to each frequency of the spectrum
(4) Do a single Inverse Fourier Transform of the whole spectrum. Usage of overlapping windows is not necessary, because there is only one single IFFT for the whole sample.

The resulting sample can be used as a wave-table to generate the desired note. These four steps are represented graphically in fig.3.

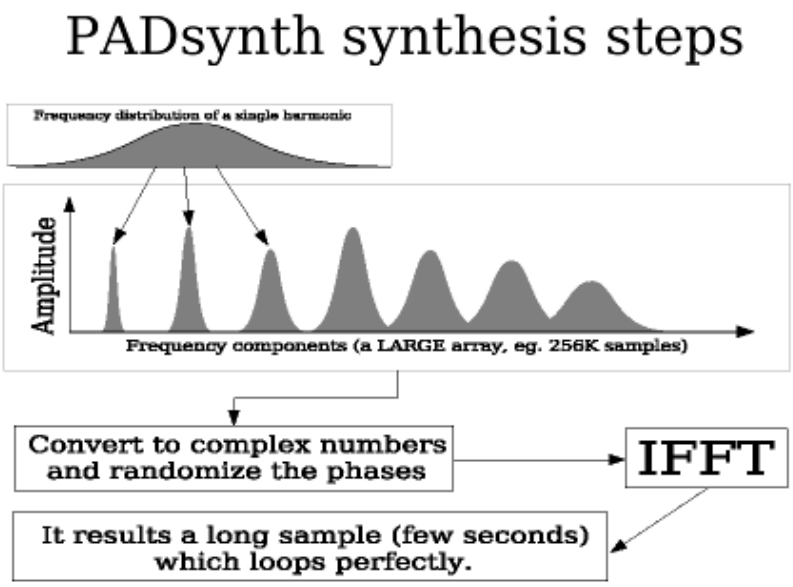


FIGURE 3. PADsynth Steps

There are some important facts of the PADsynth algorithm:

- The bandwidth of each harmonic which was described earlier into this paper is the parameter which gives the subjective quality of "warmth" or "ensemble".

- Another important parameter is the frequency distribution of each harmonic. For example, the sine components of the harmonic can be evenly spread or a Gaussian distribution can be used.

3.2. **PADsynth algorithm described in pseudo-code.** For a better understanding and for helping the implementation of this algorithm, the steps will be described in pseudo-code:

```
Input:
  N - wave-table size. It's recommanded to be a power of 2. This is,
      usually, a big number (like 262144)
  samplerate - the sample-rate (eg. 44100)
  f - frequency of the the fundamental note (eg. 440)
  bw - bandwidth of first harmonic in cents (eg. 50 cents); must be
       greater than zero
  number_harmonics - the number of harmonics; number_harmonics<(sample-rate/f)
  A[1..number_harmonics] - amplitude of the harmonics
Output:
  smp[0..N-1] - the generated wave-table

Internal variables:
  freq_amp[0..N/2-1] = {0,0,0,0,...,0}
  freq_phase[0..N/2-1], etc...

Functions:
  RND() returns a random value between 0 and 1
  IFFT()it is the inverse Fourier transform
  normalize_sample() normalizes samples between -1.0 and 1.0

  profile(fi,bwi){
      x=fi/bwi;
      return exp(-x*x)/bwi;
  };

Steps:
  FOR nh = 1 to number_harmonics
      bw_Hz=(pow(2,bw/1200)-1.0)*f*nh;
      bwi=bw_Hz/(2.0*samplerate);
      fi=f*nh/samplerate;
      FOR i=0 to N/2-1
          hprofile=profile((i/N)-fi,bwi);
          freq_amp[i]=freq_amp[i]+hprofile*A[nh];
```

```
    ENDFOR
ENDFOR

FOR i=0 to N/2-1
    freq_phase[i]=RND()*2*PI;
ENDFOR

smp=IFFT(N,freq_amp,freq_phase);
normalize_sample(N,smp);

OUTPUT smp
```

The frequency domain array ("freq_amp") data is represented below in fig. 4. Notice that on the highest frequencies the overtones merge and this cause the "hissing" discussed above.

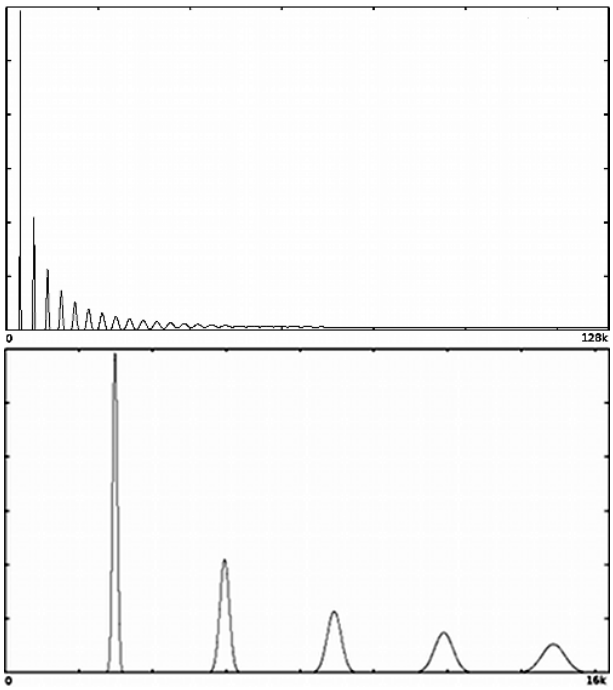


FIGURE 4. Frequency domain data (full and closeup)

3.3. **Suggestions regarding usage of PADsynth algorithm.** For best results, there are some recommendations of using PADsynth algorithm into a synthesizer.

- The algorithm produces a sample which is perfectly looped. It is recommanded to make the samples few seconds long to avoid making noticeable the repetition of the sound.
- For each new musical note the playing should start from a random position of the sample.
- For generating stereo sounds, a single sample is enough, but the sample have to be played from different positions for left/right channels.

## 4. Implementations of PADsynth

Since the day this algorithm was made public it was implemented into several open-source and commercial synthesizers. The first synthesizer which used this algorithm is ZynAddSubFX[16], an open-source audio synthesizer, mostly written by me. This synthesizer is widely distributed in most Linux software repositories. It has a large number of users, for example, on YouTube[15] there are available hundrieds videos related to ZynAddSubFX. PADsynth algorithm is used into one of its three synthesizer engines (other engines are based on additive and substractive synthesis methods). Other software synthesizers which use this algorithm are WhySynth[14], Kunquat[11], BR404[8], KarmaFX[10] and discoDSP Discovery Pro[9]. There is also, available a module [12] for SynthMaker [13] which implements PADsynth algorithm.

## 5. Conclusions

PADsynth algorithm is a simple but very versatile synthesis method. Despite of its low complexity, the sounds generated by it are subjectively pleasant and "warm", a characteristic seldom met in many digital synthesizers. Also, the "Bandwidth of each harmonic" approach to the musical instruments is a productive approach for creating new sound synthesis and sound processing algorithms.

## References

[1] Jordi Bonada, *Voice solo to unison choir transformation*, Music Technology Group, Institut Universitari de lAudiovisual Universitat Pompeu Fabra, Barcelona, 2005

[2] Akansha Pilley, N.G.Bawane, Jagruti Sanghavi *A Stand-alone and Less Power Consumption Digital Music Synthesizer using a Low Cost SoC*, International Journal of Electronics Engineering, 2010

[3] J. C. Risset, *Stochastic Processes in Quantum Theory and Statistical Physics, chapter "Stochastic processes in music and art"*, Springer Berlin / Heidelberg, 1982

[4] Ternstrom S, *Choir acoustics - an overview of scientific research published to date*, TMH-QPSR vol.43, 2002

[5] Ternstrom S, *Perceptual evaluation of voice scatter in unison and choir sounds*, STL-QPSR vol.32, 1991

[6] PRAAT, `http://www.fon.hum.uva.nl/praat`, retreived Jan 2011

[7] PADsynth algorithm with example implementations, http://zynaddsubfx.sourceforge.net/doc/PADsynth/PADsynth.htm, retreived Jan 2011
[8] BR404, http://www.kvraudio.com/get/3679.html, retreived Jan 2011
[9] discoDSP Discovery Pro, http://www.discodsp.com/discoverypro/, retreived Jan 2011
[10] KarmaFX Synth Modular, http://www.karmafx.net, retreived Jan 2011
[11] Kunquat, https://blueprints.launchpad.net/kunquat/+spec/kunquat-padsynth-generator, retreived Jan 2011
[12] Padpal, http://rekkerd.org/rock-hardbuns-updates-padpal-3/, retreived Jan 2011
[13] SynthMaker, http://synthmaker.co.uk/, retreived Jan 2011
[14] WhySynth, http://www.smbolton.com/whysynth.html, retreived Jan 2011
[15] YouTube, http://www.youtube.com/, retreived Jan 2011
[16] ZynAddSubFX, http://zynaddsubfx.sourceforge.net, retreived Jan 2011

"Petru Maior" University, Faculty of Science and Letters, Targu Mures, Romania
*E-mail address*: nascapaul@yahoo.com