

Bring Your Own Control to Additive Synthesis

Adrian Freed

CNMAT

(510) 643 9990, adrian@cnmat.berkeley.edu

ABSTRACT: Twenty years have passed since the first digital signal processing systems were used for real-time additive synthesis of sound [Chamberlin 76, DiGiugno 76]. This year, new algorithmic and implementation developments have resulted in passing the symbolic milestone of 1000 sinusoidal partials at 44.1kHz sampling rate on a desktop computer (SGI Power Indigo 2). Unlike the systems of two decades ago, this milestone was achieved with frequency and amplitude interpolated partials, and an efficient mechanism by which partials can be controlled during musical performance. This control mechanism, BYO (Bring Your Own), is the subject of this paper.

Introduction

BYO is a layer between high level programming tools (such as IRCAM/Opcode's MAX [Puckette & Zicaralli 90], Matlab, and the Apple Newton) and interpolated additive synthesis [Freed et al. 93]. The primary goal of BYO is to facilitate the development of interesting, new control paradigms for musically expressive sound synthesis. The novelty of the BYO design over more general extension mechanisms (e.g., Max externals, C++ objects, or NewtonScript proto's) is that by being optimized for manipulations of spectral descriptions (partials and noise) of sound, excellent run-time efficiency is achieved without loss of expressiveness.

An Example BYO Control Function

The following C language function describes a sound with harmonically spaced partials from a given fundamental frequency (f_0) with $1/f$ amplitude structure:

```
#include "BYO.h"
typedef struct { float f0;} Buzz;
static void synthesisframe(BYOSynthDescp c, Buzz *buzz)
{
    BYOVector f = c->frequency, a = c->amplitude;
    int i;
    for(i=0; i < c->max_partials; ++i) {
        /* harmonic partial frequencies. integer multiples of f0 */
        f.data[i*f.stride] = buzz->f0*(i+1);
        /* amplitudes are reciprocal of their frequencies */
        a.data[i*a.stride] = 1.0f/f.data[i*f.stride];
    }
    c->npartials = i;
}
```

The above function is called by a real-time synthesis program at the control frame rate, every few milliseconds. Synthesizers for BYO controls interpolate frequency and amplitude parameters as they compute output samples at the much higher sample rate. Synthesizers for BYO controls are available using the classical digital oscillator [Tierney et al 71] and the relatively new FFT⁻¹ [Depalle & Rodet 90] methods. Care in implementation is required to smoothly interpolate partial and noise amplitudes and maintain phase constraints [Freed et al. 93].

BYO control functions manipulate descriptions of the sound to be created for each synthesis frame. This description, intended to embody sinusoidal and noise signal models [McAulay&Quatieri 92&86, Serra 86, Depalle&Rodet 90], is represented by a C struct excerpted below:

```
typedef struct BYOSynthDesc
{
    /* Narrow Band (Partial) */
    unsigned max_partials;
    float max_partial_frequency;          /* maximum synthesizable frequency */

    unsigned n_partials;                  /* number of partials */
    BYOVector amplitude;                  /* linear, with 1.0f as full scale output */
}
```

```

    BYOVector frequency;          /* positive frequencies in Hz */
    BYOVector phase;              /* in radians */
    BYOVector bandwidth;          /* positive frequencies in Hz */

/* Broad Band (Noise) */
    unsigned max_bands;
    float max_noise_frequency;
    unsigned n_bands;              /* number of bands of noise in freq. domain */
    /* upper edge! 0Hz is implicitly first band edge */
    BYOVector noise_band_edge_frequencies;
    BYOVector noise_amplitude;
}BYOSynthDesc, *BYOSynthDescp;

```

The next sections describe each field of the BYO synthesis description struct.

Sinusoidal Partial

The first fields in the above structure are for sinusoidal components. The synthesizer sets `max_partials`, the maximum number of partials that may be described in subsequent BYOVectors. The `max_frequency` field is typically set to the Nyquist frequency corresponding to the output sample rate. The `n_partials` field is initially set to 0 and is increased in value as BYO control functions add new partials. The heart of the partial specification consists of the two BYOVectors, amplitude and frequency.

A BYOVector is a structure containing two fields:

```

typedef struct byovector
{
    float *data;
    int stride;
} BYOVector, *BYOVectorp;

```

The elements of these vectors are:

```
p->data[0], p->data[p->stride], p->data[2*p->stride], ... p->data[(c->max_partials-1)*p->stride]
```

The use of `stride` gives flexibility for the layout in memory of this performance critical data. With modern computers [Hennessey & Patterson 94] and their optimizing compilers [Bacon et al. 94], no performance penalty for index operations is introduced by using strides [Freed 93].

The `bandwidth` field is for frequency domain description with synthesis methods that require more than just frequency and amplitude, e.g., FOF's [Rodet et al 84], filter banks or formants [Puckette 95].

Noise Control Parameters

The frequency spectrum is divided into adjacent bands from 0Hz to `max_noise_frequency`. The number of bands must not exceed `n_bands`. Edge frequencies of the bands are specified in `noise_band_edge_frequencies` and the mean amplitude of the noise in `noise_amplitude`. The example below creates noise with a 1/f amplitude structure:

```

float bands[] = {100.0f, 200.0f, 400.0f, 800.0f, 1600.0f, 3200.0f, 6400.0f, 12800.0f, 0.0f };
static void synthesisframe(BYOSynthDescp c, const Buzz *buzz)
{
    int i;
    if(c->n_bands==0) {
        for(i=0;bands[i]>0.0; ++i)
            c->noise_band_edge_frequencies.data[i*c->noise_band_edge_frequencies.stride]
                = bands[i];
        c->n_bands = i;
    }
    for(i=0; i < c->n_bands; ++i) {
        /* amplitudes are reciprocal of their frequencies */
        c->noise_amplitude.data[i*c->noise_amplitude.stride] = 1.0f/
            c->noise_band_edge_frequencies.data[i*c->noise_band_edge_frequencies.stride];
    }
}

```

BYO synthesizers interpolate the amplitude of noise in each band from frame to frame.

External Control Events and Parameters

BYO controls handle the arrival of high level control events by binding a parameter name to a call-back function. The binding is registered to the synthesizer as follows:

```
RegisterBYOParame ter("f0","frequency of lowest partial of buzz", setf0);
```

The following call-back function will be executed when the synthesizer receives a message destined for this BYO control, and when such execution avoids concurrent operations on BYO control object instance variables:

```
static void setf0(Buzz *buzz, int bytes_received, void *parameter_received)
{
    if(bytes_received==sizeof(float))
        buzz->f0 = *(float *)parameter_received;
}
```

Delivery of these symbolically named parameters is not part of the BYO specification. Synthesizer implementations provide mappings from MIDI, ZIP1 [Mcmillen et al. 94] or Ethernet UDP sources [Freed 92, 94].

The BYO Execution Environment

BYO synthesizers implement a small interpreter that instantiates, schedules and manages BYO control functions, instance variables, synthesis descriptions and their assignment to output voices. The following schedule of BYO control functions illustrates that the strength of the BYO interface lies in its promotion of a modular view to the design of synthesis control strategies. The example implements a simple singing voice synthesizer:

Name	Description	Parameters
single	single tone	f0, amplitude
vibrato	sinusoidal modulation of partial frequencies	rate, depth
jitter	random modulation of partial frequencies	rate, depth
harmonic	band-limited harmonic partial expansion	npartials
oneoverf	scale amplitudes according to frequency	
formants	apply peaky spectral envelope	frequency, gain, bandwidth

Conclusion

The individuals acknowledged below are responsible for a diverse and expanding collection of BYO control functions that includes loudness and roughness manipulations, timbre spaces [Wessel 79] by geometric range query [Matousek 94], timbre spaces by connectionist methods [Lee&Wessel 92], temporal phrasing operators, resonances, and formants.

Acknowledgments

Paul Debevec
Gibson Guitar Company
Mark Goldstein
Michael Goodwin
Mike Lee
David Petru
Xavier Rodet
Silicon Graphics Inc.
David Wessel
Matt Wright

Bibliography

- Bacon, DF; Graham SL; Sharp OJ., "Compiler Transformations for High-Performance Computing", ACM Computing Surveys, 1994 DEC, V26 N4:345-420.
- Chamberlin, H.A., Jr. "Experimental Fourier series universal tone generator", Journal of the Audio Engineering Society, May 1976, vol.24, (no.4):271-6.
- P. Depalle & X. Rodet, "Synthèse additive par FTT inverse", Rapport Interne IRCAM, Paris 1990.
- G. DiGiugno, "A 256 Digital Oscillator Bank," Presented at the 1976 Computer Music Conference, Cambridge, Massachusetts: M.I.T., 1976
- Freed, A., "Tools for Rapid Prototyping of Music Sound Synthesis Algorithms and Control Strategies", in: Proc. Int. Comp. Music. Conf., San Jose, CA, USA, Oct. 1992
- Freed, A., Rodet, X. Depalle, P, 1993, "Synthesis and Control of Hundreds of Sinusoidal Partial on a Desktop Computer without Custom Hardware", Proceedings of ICSPAT, 1993, DSP Associates, Boston, MA.
- Freed, A. "Guidelines for signal processing applications in C", *C Users Journal* v11, n9 (Sept, 1993):85
- Freed, A. "Codevelopment of User Interface, Control and Digital Signal Processing with the HTM Environment," *Proceedings of The International Conference on Signal Processing Applications & Technology*, Dallas, Texas, 1994.
- John Hennessey and David A. Patterson, "Computer organization and design : the hardware/software interface", Morgan Kaufmann, 1994.
- M. Lee & D. Wessel, "Connectionist Models for Real-Time Control of Synthesis and Compositional Algorithms", Proceedings of ICMC, 1992.
- Matousek, J. "Geometric Range Searching", ACM Computing Surveys, 1994 Dec, V26 N4:421-461.
- R.J. Mc Aulay and Th. F. Quatieri, "Speech analysis/synthesis based on a sinusoidal representation", IEEE Trans. on Acoust., Speech and Signal Proc., vol ASSP-34, pp. 744-754, 1986.
- McMillen, K.; Wessel, D.; Wright, M., "The ZIPI Music Parameter Description Language", Computer Music Journal, Winter 1994, vol.18, (no.4):52-73.
- Puckette, M., Zicarelli, D., 1990, "MAX – An Interactive Graphic Programming Environment", Opcode Systems, Menlo Park, CA, 1990.
- Puckette, M., "Formant-based audio synthesis using nonlinear distortion.", Journal of the Audio Engineering Society, Jan.-Feb. 1995, vol.43, (no.1-2):40-7.
- Th. F. Quatieri and R. J. McAulay, "Shape Invariant Time-Scale and Pitch Modification of Speech", IEEE Trans. on Signal Processing, Vol. 40 No. 3, March 1992.
- X. Rodet, Y. Potard, J.B.B. Barrière, 1984, "The CHANT Project: From the Synthesis of the Singing Voice to Synthesis in General", *Computer Music Journal*, 8(3):15-31.
- X. Serra, "A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition", PhD dissertation, Stanford Univ., 1986.
- J. Tierney, C. M. Rader, and B. Gold, "A digital frequency synthesizer," IEEE Trans. Audio Electroacoustics, vol AU-19, pp 48-57, March 1971.
- D. Wessel, "Timbre space as a musical control structure," *Computer Music Journal* 3(2):45-32, 1979.