ABSTRACT: Twenty years have passed since the first digital signal processing systems were used for real-time additive synthesis of sound [Chamberlin 76, DiCuspano 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 as efficient mechanisms 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/Opodee's MAX [Puckette & Zicarelli 90], Max/MSP, 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 (f0) with 1/f amplitude structure:

```c
#include "BYO.h"
typedef struct { float f0; } Base;
static void synthesizetable(BYOSynthsBase &Buzz *buzz)
{
  int i;
  for(i=0; i < Buzz->maxpartials; i++) {
    /* harmonic partial frequencies. integer multiples of f0 */
    double f0_freq = buzz->f0*f0s[i];
    /* amplitudes are reciprocals of these frequencies */
    a_details->ampl[i] = 1.0/f0_freq;
    }
  c->partials = 1;
}
```

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 [Tsevery et al 71] and the relatively new FFT [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, Sierra 86, Depalle&Rodel 90], is represented by a C struct excepted below:

```c
typedef struct sBYOSynthDesc {
  /* sound width (Partials) */
  unsigned numpartials;
  float max_partial_frequency; /* maximum synthesizable frequency */
  unsigned npartials;
  /* number of partials */;
  BYODOUBLE amplitude; /* linear, with 1.0f as full scale output */
};
```

ICMC PROCEEDINGS 1995 203
The next sections describe each field of the BYO synthesis description struct.

Sinusoidal Partials

The first fields in the above struct are for sinusoidal components. The synthesizer sets maxpartials, 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:

```c
typedef struct byovector {
  float freq;
  unsigned npartials;
} BYOVector;
```

The elements of these vectors are:

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

The use of stride gives flexibility for the layout in memory of this performance critical data. With modern computers [Hennessy & 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., POPs (Rodet et al. 84), filter banks or fortrans [Packets 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 1/2 amplitude structure:

```c
float noise_band_edge_frequencies[] = {100.0f, 200.0f, 400.0f, 500.0f, 600.0f, 900.0f, 1200.0f};
```

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

304  ICMC PROCEEDINGS 1995
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:

```java
RegisterBYOParameter("D", "Frequency of lowest partial of buzz", setD);
```

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:

```java
static void setD0( buzz *buzz, int byte_received, void *parameter_received)
{
  if(byte_received==sizedFloat())
    buzz->d0 = *(float *)parameter_received;
}
```

Delivery of these symbolically named parameters is not part of the BYO specification. Synthesizer implementation provide mappings from MIDI, ZIPI [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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>single</td>
<td>single tone</td>
<td>f0, amplitude</td>
</tr>
<tr>
<td>jitter</td>
<td>sinusoidal modulation of partial frequencies</td>
<td>rate, depth</td>
</tr>
<tr>
<td>harmonic</td>
<td>random modulation of partial frequencies</td>
<td></td>
</tr>
<tr>
<td>onover</td>
<td>band-limited harmonic partial expansion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scale amplitudes according to frequency</td>
<td></td>
</tr>
<tr>
<td></td>
<td>apply peaky spectral envelope</td>
<td></td>
</tr>
<tr>
<td></td>
<td>frequency, gain, bandwidth</td>
<td></td>
</tr>
</tbody>
</table>

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 [LeedWessel 92], temporal phrasing operators, reenactments, and formats.

Acknowledgments

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


