Music 209
Advanced Topics in Computer Music
Lecture 6 – Real-Time Control

2006-2-23

Professor David Wessel (with John Lazzaro)
(cnmat.berkeley.edu/~wessel, www.cs.berkeley.edu/~lazzaro)

www.cs.berkeley.edu/~lazzaro/class/music209
Today, we show software examples of real-time control in concatenative synthesis.
Topics for today ...

- Systems architecture
- Structured Audio tutorial
- Example: A 185 MB piano
- Concatenative coding techniques
MIDI cable

Piano Samples

Stereo Audio Out
MIDI : A network protocol for musical instrument control

Unidirectional serial link - 31,250 Hz
Sends commands for key press, key release, knobs.
MIDI: Commands sent on a wire

Command sent on one of 16 voice channels

<table>
<thead>
<tr>
<th>Channel Voice Messages</th>
<th>Bitfield Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoteOff (end a note)</td>
<td>1000cccc 0nnnnnnn 0vvvvvvvv</td>
</tr>
<tr>
<td>NoteOn (start a note)</td>
<td>1001cccc 0nnnnnnn 0vvvvvvvv</td>
</tr>
<tr>
<td>PTouch (Polyphonic Aftertouch)</td>
<td>1010cccc 0nnnnnnn 0aaaaaaa</td>
</tr>
<tr>
<td>CControl (Controller Change)</td>
<td>1011cccc 0xxxxxxx 0yyyyyy</td>
</tr>
<tr>
<td>PChange (Program Change)</td>
<td>1100cccc 0pppppppp</td>
</tr>
<tr>
<td>CTouch (Channel Aftertouch)</td>
<td>1101cccc 0aaaaaaa</td>
</tr>
<tr>
<td>PWheel (Pitch Wheel)</td>
<td>1110cccc 0xxxxxxx 0yyyyyy</td>
</tr>
</tbody>
</table>

128 notes, 60 = Middle C
127 strike velocities, 0 = NoteOff
Controllers send 7-bit values. Ex: controller 7 is channel volume.

Program change: 7-bit number maps channel to a timbre.
Due Next Weds!

<table>
<thead>
<tr>
<th>Title</th>
<th>Due Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Abstract</td>
<td>March 1, 11:59 PM</td>
<td>A short (one or two page) description of the project. PDF or plain text format is fine -- please, no .doc files. Collaborative projects should include information on how the work will be split between team members. Email this abstract to the instructors (wessel [at] cnmat [dot] berkeley [dot] edu, lazzaro [at] eecs [dot] berkeley [dot] edu).</td>
</tr>
</tbody>
</table>

You are free to propose a project topic of your own creation. Alternatively, you may choose one of the project ideas below (click on the link for a complete description).

- **Drum-related Projects**
  - Creating Electronic Drum Samples from Acoustic Drum Samples
  - Tools for Automating Drum Track Arrangements
  - Timbre-Space Browsers for Drum Loops and Individual Hits
  - Realistic Retuning of Drum Sounds
  - Real-Time Performance by Retiming Drum Loops
  - Fusing Multiple Drum Hits into a Single Percept
- **Wind Instrument Projects**
  - Playing Horns from a Keyboard with Improved Articulation
  - Automatic Horn Phase Selection to Match a Track
  - Real-time Timbre Selection with a Wind Controller
- **Computer Systems Projects**
  - CoreSample: Kernel Database Services for Concatenative Synthesis
- **Vocal Projects**
  - Synthesis, Analysis, and Algorithmic Composition of Glossolalia Vocals
  - Lyric Design for Phrase-Based Vocal Synthesis
Structured Audio
SAOL (pronounced "sail")

ISO/IEC JTC 1/SC 29/WG 11
N2503-sec5
Date: 1999-3-10

ISO/IEC FDIS 14496-3 sec5
ISO/IEC JTC 1/SC 29/WG11
Secretariat: Narumi Hirose

Information technology - Coding of audio-visual objects
Part 3: Audio
Section 5: Structured audio

Eric Scheirer (MIT Media Lab)
MPEG-4 Structured Audio: Developer Tools

By John Lazzaro and John Wawrzynek, CS Division, UC Berkeley.

MPEG-4 Structured Audio

MPEG-4 Structured Audio (MP4-SA) is an ISO/IEC standard (edited by Eric Scheirer) that specifies sound not as sampled data, but as a computer program that generates audio content for MPEG 4 Structured Audio.

The MP4-SA Book

We wrote an online book to show how to create audio content for MPEG 4 Structured Audio.

Links

Introductory Example

COMPILING MPEG 4 STRUCTURED AUDIO INTO C

John Lazzaro and John Wawrzynek

CS Division
UC Berkeley
Berkeley, CA, 94720
{lazzaro, johnw}@cs.berkeley.edu

ABSTRACT

Structured Audio (SA) is an MPEG 4 Audio standard for algorithmic sound encoding, using the programming language SAOL. The paper describes a SA decoder, sfront, that translates a SAOL program into a C program, which is then compiled and executed to create audio. Performance data shows a 7.6x to 20.4x speedup compared to the SA reference MPEG decoder.

sfront

Download the latest version of sfront, a translator that converts MP4-SA files into efficient C programs that generate audio for rendering, interactive and network applications.

Sfront is written by John Lazzaro and John Wawrzynek, and is freely redistributable under the terms of the GNU Public License.

The sfront reference manual describes how to install and use the program. Developers can learn how to add control and audio drivers to sfront, as well as learn about the internals of sfront and the C programs it creates.
A SAOL “Instr”

Each MIDI NoteOn launches a new instance of the instr bound to the program number.

Instance passed in NoteOn number and velocity.

Sets program number for a MIDI channel

NoteOff schedules instance to terminate

Where MIDI meets SAOL

MIDI program number

// code goes here

instr sine (pitch, vel) preset 0 {

	// code goes here

}

<table>
<thead>
<tr>
<th>Channel Voice Messages</th>
<th>Bitfield Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoteOff (end a note)</td>
<td>1000cccc 0nnnnnnn 0vvvvvvv</td>
</tr>
<tr>
<td>NoteOn (start a note)</td>
<td>1001cccc 0nnnnnnn 0vvvvvvv</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>PChange (Program Change)</td>
<td>1100cccc 0pppppppp</td>
</tr>
</tbody>
</table>
Benefits

The language runtime does real-time scheduling for you. All you do is supply behavior code.

No extra code for polyphony.

The language makes parallelism explicit: SAOL code is multi-core ready.
Execution model

// Variable declarations

Global block sets audio and control sample rates.

Benefit: Keeps code for all timescales in one place.

// Code that runs once, at instantiation.

Birth

No ipass

Benefit: Keeps code for all timescales in one place.

// Code that runs at the start of each control cycle.

Birth

No ipass

Benefit: Keeps code for all timescales in one place.

// Code that runs at the audio sample rate.

truefalse

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Cycle</th>
<th>Pass</th>
<th>X-#</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.990000</td>
<td>k-cycle</td>
<td>a-cycle</td>
<td></td>
</tr>
<tr>
<td>0.990025</td>
<td>a-cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.990050</td>
<td>a-cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.990075</td>
<td>a-cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>a-cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.999925</td>
<td>a-cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.999950</td>
<td>a-cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.999975</td>
<td>a-cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.000000</td>
<td>k-cycle</td>
<td>i-pass</td>
<td>--</td>
</tr>
<tr>
<td>1.000025</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.000050</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.000075</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.009975</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.010000</td>
<td>k-cycle</td>
<td>k-pass</td>
<td>--</td>
</tr>
<tr>
<td>1.010025</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.010050</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.010075</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.019975</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.020000</td>
<td>k-cycle</td>
<td>k-pass</td>
<td>--</td>
</tr>
<tr>
<td>1.020025</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.020050</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.020075</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
<tr>
<td>1.029975</td>
<td>a-cycle</td>
<td>a-pass</td>
<td></td>
</tr>
</tbody>
</table>
Example 1: One Note Fits All ...

Plays one piano sample across entire keyboard. Each key plays same pitch.

```
global {
    srate 44100;
    krate 1050;
    table right_060_mf(sample, -1, "060_C3KM56_M.wav");
}
```

Samples read from disk and locked into RAM during startup.

Variable name used in SAOL program code.

Piano sample file on disk.
Instr declaration + i-rate code

instr piano_proto (pitch, \texttt{vel}) preset 0 {

How global \{\} variables are made visible in an instr.

\textbf{imports exports table} \texttt{right\_060\_mf};
\textbf{ivar} rel\_time, full\_scale, volume;
\textbf{ksig} k, rel;
\textbf{asig} i;

// ********************
// computed during i-pass
// ********************

rel\_time = 0.250;
full\_scale = 0.25;
volume = full\_scale*(\texttt{vel}/127); // NoteOn velocity scales volume

Assignments to a variable happen at the rate indicated by these keywords (ivar, ksig, asig).

“\texttt{vel}” is the velocity value of the NoteOn that created the instance.
Instr  k-rate code

A “standard name” (built-in variable).

released is 1 if instance is slated for termination before next k-pass.

if (released && !rel) // Add release time when NoteOff occurs
{
    rel = 1;
    extend(rel_time);
}

if (!rel &&
    (k > flen(right_060_mf) - rel_time*s_rate - 2*(s_rate/k_rate)))
{
    turnoff; // Force NoteOff before we run out of samples
}

k = k + (s_rate/k_rate);

We postpone termination so that we can fade note out.
extend() is a SAOL command.

If the sample ready to run out?
If so, we “force” a NoteOff by using the turnoff command. Next k-pass, released will be 1.
**Instr a-rate code**

```plaintext
// *******************************
// computed during a-pass
// *******************************

if (!rel)    // Attack and sustain portion of note
{
    output(volume*tableread(right_060_mf, i));
}
else        // Release envelope after NoteOff
{
    output(aline(volume, rel_time, 0)*tableread(right_060_mf, i));
}

i = i + 1;
```

---

**Demo**

- **output** Sum audio sample value onto the instr's "output bus".
- **tableread(...)** Interpolated read of the i’th sample of the table.
- **NoteOn**: Scale by velocity constant.
- **NoteOff**: Scale by fadeout envelope.
Sample Databases
Sample database

But how do we conveniently access 100s of samples in a SAOL program?

global {

    table left_024_mf
        (sample, -1, "024_C0KM56_M.wav");

    table right_024_mf
        (sample, -1, "024_C0KM56_M.wav");

    table left_031_mf
        (sample, -1, "031_G0KM56_M.wav");

    table right_031_mf
        (sample, -1, "031_G0KM56_M.wav");

    [...]

tablemaps

instr full (pitch, vel) preset 0 {

imports exports table low;
imports exports table mid;
imports exports table hi;

tablemap set(low, mid, high);

output(tableread(set[1], i));

}