Nyquist

Tutorial Text

Roger B. Dannenberg (CMU)

International Conference Center
Waseda University
September 10, 1993
Room D, 14:00~17:00
Nyquist Tutorial

Roger B. Dannenberg
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
dannenberg@cs.cmu.edu

Nyquist Overview

- A sound synthesis language
  - gives flexibility
  - extensibility
  - tailor to your own needs
- Manages signals in time
  - create instances of sound computations, e.g. notes
  - logical stop times for joining sequences
  - multiple sample rates
- Composition and structure
  - symbolic pitch and duration possible
  - sequential and parallel combination
  - transformation operators: transpose, stretch, etc.
Tutorial Overview

- About Nyquist – some basics
- Lisp Basics – enough to make you dangerous
- Nyquist – language elements
- Examples – small programs that work
- Leftovers

Differences from Music n, cmusic, Csound

<table>
<thead>
<tr>
<th>Music n</th>
<th>Nyquist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate Score and Orchestra languages</td>
<td>One unified language for both</td>
</tr>
<tr>
<td>Score preprocessors often needed</td>
<td>Powerful built-in scoring capability</td>
</tr>
<tr>
<td>No hierarchy: difficult to phrase</td>
<td>Naturally hierarchical</td>
</tr>
<tr>
<td>Two levels: instruments cannot invoke score language</td>
<td>Fully recursive, multi-level</td>
</tr>
<tr>
<td>Synthesis-oriented</td>
<td>Analysis and Synthesis support</td>
</tr>
<tr>
<td>Difficult to extend efficiently</td>
<td>Instrument compiler simplifies extension writing</td>
</tr>
</tbody>
</table>
Installing Nyquist

- ftp unix tar file from g.gp.cs.cmu.edu, directory /usr/rbd/pub/nyquist
- install the source:
  zcat nyquist.tar.Z | tar xf -
- print the manual:
  lpr nyquist.ps
- compile nyquist:
  make
- run it:
  ny

Quick Examples

- (play (osc C4))
- (mult (osc C4)
  (env .05 0.1 .5 1.0 .5 .4))
- (seq (note c4) (note c5))
- (seq (phrase) (trans 2 (phrase)))
- (snd-lp 400.0
  (snd-load "mysound.snd"))
Quick Examples (Behind the Curtains)

- (defun note (pitch)
  (mult (osc pitch) (env ...)))

- (defun phrase ()
  (seqrep (i 12)
    (stretch 0.15
      (note (+ C4 i)))))

Introduction to Lisp

- Lisp is interactive: read - eval - print loop

  > 2
  2

  > (+ 1 2)
  3
Lisp (cont'd): Applying a Function to an Argument

(osc C5)

Function  Argument (Parameter)

(play (osc C5 ))

(play (osc C5 ))

Lisp (cont'd): Multiple Function Arguments

(s-add a b)

Function  Arguments (Parameters)

(stretch 0.3 (osc C4))

(stretch 0.3 (osc C4 ))
Lisp (cont’d): Invoking a Function with No Arguments

(gc)

Function
No Arguments

(print (room))
(print room)

Lisp (cont’d): Defining a Function

(defun incr (n) (+ n 1))

Function Name
Parameter List
Function Body

(defun note (pitch)
  (mult (env)
    (osc pitch)))

(defun note (pitch)
  (mult (env)
    (osc pitch)))
Lisp (cont’d): Assigning Values to Variables

> (setq v 2)
2
> v
2
> (setq s (osc C4))
<sound: #2006a074>
> (play s)
nil

Lisp (cont’d): Introducing Local Variables

(let (x y)
   (setq x (osc C5))
   (setq y (osc G5))
   (sim x y)) # adds sounds

(let ((x (osc C5))
   (y (osc G5)))
   (sim x y))
Lisp (cont’d): Arrays

> (vector 0.1 0.2 0.3)
#(0.1 0.2 0.3)
> (setf x (vector 0.1 0.2 0.3))
#(0.1 0.2 0.3)
> (aref x 0)
0.1
> (aref x 3)
error: array index out of bounds - 3
> (setf (aref x 2) 0.4)
0.4
> x
#(0.1 0.2 0.4)

Lisp (cont’d): Immutable Values

- Data is rarely modified
- New data is often allocated or generated
- Never use parameters to return values
Lisp (cont’d): Syntax Review

- (function arg1 arg2 arg3 ...)
- (defun function (arg1 arg2 ...) expression1 expression2 ...
- (defun function (arg1 arg2 ...) (let (var1 var2 ...(var3 expression ...) expression1 expression2 ...
- (setf variable expression )

Lisp (cont’d): Debugging

- (trace function ) - adds function to trace list
- (untrace (function )) - removes function
- (setf *tracenable* t) - backtrace on error
- (setf *breakenable* t) - break on error
- (clean-up) - pop back to previous error
- (top-level) - pop up to top level
- (btrace n ) - print n-level backtrace
- (continue) - retry restartable error
Lisp (cont'd):
Debugging Example

> (defun add-n (i) (+ i n))
ADD-N
> (add-n 3)
error: unbound variable - N
if continued: try evaluating symbol again
Function: #<Closure-ADD-N: #20058e48>
Arguments:
  3
1> (setf n 3)
3
1> (continue)
[ continue from break loop ]
6

Lisp (cont’d):
More Debugging

> (trace add-n)
(ADD-N)
> (add-n (add-n 5))
Entering: ADD-N, Argument list: (5)
Exiting: ADD-N, Value: 8
Entering: ADD-N, Argument list: (8)
Exiting: ADD-N, Value: 11
11
>
The Sound Type

- A stream of samples, with
- a (fixed) sample rate
- a scale factor
- a time offset
- a logical stop time
- A genuine Lisp data type:
  \[ \text{type-of sound} \to \text{SOUND} \]

Naming Conventions

- s-fn – low-level sound function (avoid)
- snd-fn – common sound function
- no prefix – operation on behaviors
Sound Generation Functions

- \((\text{env } t_1, t_2, t_4, l_1, l_2, l_3, \{\text{dur}\})\)
- \((\text{pwl } t_1, t_2, l_2, l_3, \ldots, t_n)\)
- \((\text{1tofreq } \{\text{dur table phase}\})\)
- \((\text{mult beh1 beh2 } \ldots)\)
- \((\text{rest } \{\text{dur}\})\)
- \((\text{noise } \{\text{dur}\})\)

Sound Generation (Cont’d)

- \((\text{osc pitch } \{\text{dur table phase}\})\)
- \((\text{osc-note pitch } \{\text{dur env volume table}\})\)
- \((\text{amosc pitch mod } \{\text{table phase}\})\)
- \((\text{fmosc pitch mod } \{\text{table phase}\})\)
- \((\text{snd-lp sound cutoff})\)
- \((\text{snd-hp sound cutoff})\)
- \((\text{snd-reson sound center bandwidth})\)
Instruments

Defined by Lisp functions:

(defun fm-inst (pitch mod-index)
  (mult (my-env) (fmosc ...)))

(defun add-inst (pitch)
  (sim (partial-1 pitch)
       (partial-2 pitch)
       (partial-3 pitch)
       (partial-4 pitch)))

Scores

- using seq:
  (seq (note C4) (fm-inst D4 2.5))

- using sim:
  (sim (note C4) (fm-inst D4 2.5))

- using defun:
  (defun myscore ()
    (seq (note C4) (fm-inst D4 2.5))
   )
Linear Scores

(sim
(at 0.0 (flute 1.2 C5 ...))
(at 0.5 (bass 0.7 Ef3 ...))
(at 0.9 (bass 1.3 E3 ...))
(at 1.1 (clar 1.1 Ef4 ...))
...)

Hierarchical Scores

(defun phrase-1 ()
(sim (at 0.0 (flute 1.2 C5 ...))
    (at 0.5 (bass 0.7 Ef3 ...))))

(defun phrase-2 ()
(sim (at 0.0 (bass 1.3 E3 ...))
    (at 0.2 (clar 1.1 Ef4 ...))))

(sim
(at 0.0 (phrase-1))
(at 0.9 (phrase-2))
...)

Ngquist Tutorial -27- Roger D. Dannenberg
Computed Scores

(defun roll (dur)
  (let ((n (/ dur stroke-len)))
    (seqrep i n (stroke))))

(defun random-melody (n)
  (seqrep i n
    (let ((pitch (+ c4 (random 30))))
      (note pitch))))

Computed Scores (cont’d)

(defun roll ()
  (let ((n (/ *stretch* stroke-len)))
    (stretch-abs 1.0
      (seqrep i n (stroke))))))

(roll)  

(stretch 2.0 (roll))  

time —

(
Transformations

- (stretch 0.5 (note C4))
- (transpose 2
  (stretch 0.5 (myscore)))
- Transformation Operators are:
  - abs-env
  - at, at-abs
  - control-erate-abs
  - sound-erate-abs
  - extract, extract-abs
  - loud, loud-abs
  - stretch, stretch-abs
  - trans, trans-abs

Combination & Structure

- (seq beh1[ beh2 ... ])  
- (seqrep (var limit ) beh )  
- (sim beh1[ beh2 ... ])  
- (simrep (var limit ) beh )  
- (set-logical-stop beh time )
Behavioral Abstraction

- "instrument" defines whole class of sounds
- parameters allow you to specify some details
- transformations support "implicit"
  parameterization:
  (defun note (p)
      (mult (env ...) (osc p)))

(stretch 2.0 (note GS4))
(loud 4.0 (note GS4))
(trans 12.0 (note GS4))

Example 1 - Wavetable

(defun mkwave ()
  (let ((table
      (a-add (a-scale (build-harmonic 1.0) 1.0)
      (a-add (a-scale (build-harmonic 2.0) 0.5)
      (a-add (a-scale (build-harmonic 3.0) .25)
      (a-scale (build-harmonic 4.0) 0.12)))))
  (setf *wave*
    (list table
      (hz-to-step
       (/ *SOUND-SRATE* 2048.0))
      t)))))

(if (not (boundp 'wave*)) (mkwave))
Example 2 - Table-lookup osc

(defun note (pitch dur)
  (osc pitch dur 0 *wave*))

Example 3 - Note Sequence

(play (seq (note c4 i)
           (note d4 i)
           (note f4 i)
           (note g4 i)
           (note d4 q)))

note: i = 0.3 (eighth note at MM=100),
q = 0.6 (quarter note)
Example 4 - Envelope & Duration

(defun env-note (p)
  (mult (note p 1.0)
       (env .05 0.1 0.5 1.0 0.5 .4)))

; try it out
(play (env-note c4))

; use it with stretch to change duration
(play (seq (stretch 0.25
            (seq (env-note c4)
                 (env-note d4)))
       (stretch 0.5
                (seq (env-note f4)
                      (env-note g4)))
       (env-note c4)))

Example 5 - Sounds vs. Behavior

; load a sound
(setq a-snd
     (sf-load "demo-snd.nh" 22050.0))

; play it
(play a-snd)

; two sounds in sequence:
(seq a-snd a-snd) ; WRONG!
(seq (cue a-snd) (cue a-snd)) ; RIGHT!
Example 4 - Envelope & Duration

(defun env-note (p)
  (mult (note p 1.0)
        (env .05 0.1 0.5 1.0 0.5 .4)))

; try it out
(play (env-note c4))

; use it with stretch to change duration
(play (seq (stretch 0.25
            (seq (env-note c4)
                  (env-note d4)))
            (stretch 0.5
                  (seq (env-note f4)
                        (env-note g4)))
            (env-note c4))))

Example 5 - Sounds vs. Behavior

; load a sound
(setq a-snd
     (sf-load "demo-snd.nh" 22050.0))

; play it
(play a-snd)

; two sounds in sequence:
(seq a-snd a-snd) ; WRONG!
(seq (cue a-snd) (cue a-snd)) ; RIGHT!
Example 6 - Sim and At

(play (sim (at 0.0 (cue a-snd)))
   (at 0.7 (cue a-snd)))
   (at 1.0 (cue a-snd)))
   (at 1.2 (cue a-snd))))

Example 7 - Nested Transformations

(sim (cue a-snd)
   (loud 2.0 (at 3.0 (cue a-snd)))))

(loud 2.0 (sim (at 0.0 (cue a-snd))
   (at 0.7 (cue a-snd))))
Example 8 - Defining Behaviors

(defun snds (dly)
  (sim (at 0.0 (cue a-snd))
    (at 0.7 (cue a-snd))
    (at 1.0 (cue a-snd))
    (at (+ 1.2 dly) (cue a-snd)))
  (play (snds 0.1))
  (play (loud 2.5
    (stretch 0.9 (snds 0.3)))))

Example 9 - Stretching Sounds

(play (stretch 3.0 (sound a-snd)))

(defun down ()
  (seq (stretch 0.2 (sound a-snd))
    (stretch 0.3 (sound a-snd))
    (stretch 0.4 (sound a-snd))
    (stretch 0.5 (sound a-snd))
    (stretch 0.6 (sound a-snd)))
  (play (down))
  (play (seq (down) (up) (down))))
Example 10 - Saving Sounds

; save sound to a file
(sf-save a-snd "a-snd-file.snd")

; execute Unix command to play file
(system "play a-snd-file.snd")

; simple string manipulation
(setf snd-file-name "a-snd-file.snd")
(system
  (strcat "play " snd-file-name))

Example 11 - FM Sound

(fmosc pitch mod table phase)

(play (fmosc c4 (pwl 0.1))
; one octave frequency sweep:
(play (fmosc c4
  (pwl 0.5 (step-to-hz C4) 0.501))

(fmosc c4
  (pwl 0.5 (step-to-hz Cs4) 0.501)
  *fm-voice* 0.0)
; vibrato:
(fmosc Cs2 (*-scale (lfo 6.0) 10.0)
  *fm-voice* 0.0)
Multichannel Sound

- An array of sounds is a multichannel sound:
  (vector sound1 sound2 ...)
  (mult (env) multi-sound)
  (s-lp multi-sound 400.0)
- Sample rates need not match
- Functions generalize to multichannel operands:

Extending Nyquist

(mult
  (arguments ("sound_type" "s1")
             ("sound_type" "s2"))
  (inner-loop "output = s1 * s2")
  (commutative ("s1" "s2"))
  (linear "s1" "s2")
  (start (max "s1" "s2"))
  (logical-stop (min "s1" "s2"))
  (terminate (min "s1" "s2")))

*translate.lsp: mult.alg ↔ mult.c
Future Plans

- MIDI file interface
- Pitch analysis
- Synthesis techniques:
  - FOF synthesis
  - Reverberation
  - Special Location
  - Sampling (with looping)
- Analysis/Synthesis techniques:
  - LPC
  - Phase Vocoder
  - Spectral Interpolation
  - Wavelet transform
- Instrument libraries

Conclusion

- Thank you for your attention!
- Please share your comments on this tutorial
- ... and on Nyquist.