INTRODUCTION TO COMPUTER MUSIC
PROGRAMMING TECHNIQUES

Mastering Nyquist

Roger B. Dannenberg
Professor of Computer Science, Art, and Music

Programming Techniques

- Recursive sound sequences
- Matching durations
- Smooth transitions
- Composing control functions
- Global vs Local control functions
- Stretchable behaviors
- Reading Sound Files
- Using Open Sound Control

Recursive Sound Sequences

- SEQ delays evaluation of each behavior (it's *lazy*)
- Infinite sounds can be expressed recursively:

```
define function drum-stroke()
  return noise() * pwev(1, 0.05, 0.1)
define function drum-roll()
  return seq(drum-stroke(), drum-roll())
define function limited-drum-roll()
  return const(1, 2) * drum-roll() ; duration=2
play limited-drum-roll()
```

- Note that multiplying limited sound by an infinite sound gives us a finite computation and result.

MATCHING DURATIONS

Getting 2 sounds to have the same length

Matching Durations

- Most common error in Nyquist: Combining sounds and controls with different durations.
- Example of common error:

```
play
  pwl(0.5, 1, 10, 1, 13) * ; 13-seconds duration
  osc(c4) ; nominally 1-second duration
; result sound stops at 1 second(!)
```

- Remember that Nyquist sounds are immutable. Nyquist will not adjust behaviors to get the "right" durations – how would it know?

Specifying Durations

- Make everything have nominal length of 1 and use STRETCH:

```
(pwl(0.1, 1, 0.8, 1, 1) * osc(c4)) ~ 13
```

- Provide duration parameters everywhere:

```
pwl(0.5, 1, 10, 1, 13) * osc(c4, 13)
```

- If you provide duration parameters everywhere, you will often end up passing duration as a parameter – that's not always a bad thing.

CONTROL FUNCTIONS

Synthesizing control is like synthesizing sound

Smooth Transitions

- Apply envelopes to almost everything.
- See Code 6 (code_6.htm) for example:
  - Without envelopes
  - With gradually increasing vibrato
  - With amplitude envelope
  - With richer wave table
  - With time-varying filter

Composing Control Functions

- Try combinations of:
  - LFO – low frequency sinusoid
  - PWL – arbitrary contours and shapes
  - NOISE – random jitter
- See Code 6 (code_6.htm) example using NOISE

GLOBAL VS LOCAL CONTROL

Hierarchical control

Global vs Local Control Functions

- Control functions from PWL, LFO, etc., can be passed as parameters and returned from functions.
- They are of type SOUND, just like audio.
- See example in code_6.htm of control function spanning many “notes”

STRETCHABLE BEHAVIORS

Toward behavioral abstraction

Making "Stretchable" Behaviors

- Nyquist has default stretch behaviors for all primitives,
- But this may not be what you want
- Often, you want certain things to stretch, and others (e.g. rise times) to remain fixed.

Stretch Example 1

- You want the *number* of events to increase with stretch:

```
define function n-things()
 begin
    with dur = get-duration(1),
    n = round(dur / *thing-duration*)
    return seqrep(i, n, thing() ~~ 1)
 end
```


Stretch Example 2

- You want an envelope to have a *fixed* rise time. MY-ENVELOPE has a fixed rise and fall time, but stretches with the stretch factor:

```
define function my-envelope()
 begin
   with dur = get-duration(1)
   return pwl(*rise-time*, 1,
              dur - *fall-time*, 1, dur) ~~ 1
  end
```


GRANULAR SYNTHESIS

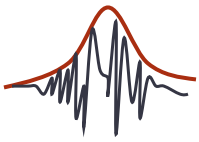
A versatile synthesis technique

Summary

- Duration mismatch is a common bug in Nyquist programs:
  - Normalize durations to 1 and use stretch (~)
  - Explicit durations everywhere
- Smooth transitions – not just fade-in/fade-out
- Do not neglect control functions or copy over-simplified examples – your goal is expressiveness
- Global control spanning many sounds (notes) add expressiveness on a different time scale

Granular Synthesis

- Combine many “grains” of sound
- Grain is typically taken from a sound file
- Apply smooth envelope to avoid clicks



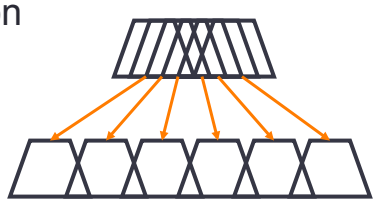
- Grains can overlap

Control

- Too many grains to specify each one
- Stochastic/Statistical control is common
- Dimensions:
  - Where to get grain: smooth progression or random
  - Resample grain? Fixed ratio or random in range.
  - When to play grains? Regular or random.

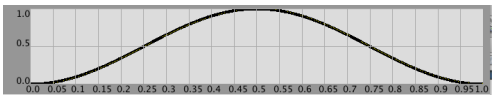
Things to do with Granular Synthesis

- Texture generation: contains spectrum but loses articulation, rhythm, identity
- Vocal mumblings: grains can chop up speech to make speech-like nonsense
- Time stretching
- Or compression

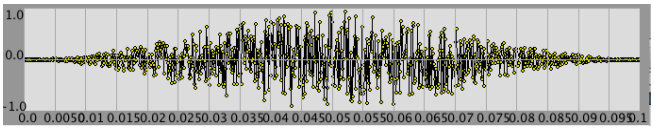


Implementation: Contruct a Grain

```
function cos-pulse()
        return 0.5 * (1 + hzosc(1 / get-duration(1),
                                *sine-table*, 270.0))
```



```
s-read("filename.wav", time-offset: seconds, dur: d) *
  (cos-pulse() ~ d)
```



GRAINS IN SCORES

Generating grains as sound events in scores

Implementation: Using Scores

- You can make a score with Score-gen, e.g.

```
{{0 0.05 {grain offset: 2.1}}
 {0.02 0.06 {grain offset: 3.0}}
 ...}
```

- And define a function:

```
function grain(offset: 0)
  begin with dur = get-duration(1)
    return s-read("filename.wav",
                  time-offset: offset, dur: dur) *
           cos-pulse()
```


Implementation: Using Score-Gen

- For the previous example, we need to specify time (or inter-onset-time), duration, and offset.
- We could extend this to pass in other parameters to modify grains, e.g. pitch shift:

```
score-gen(score-len: 2000,
          ioi: 0.05 + rrandom() * 0.01,
          dur: next(dur-pat),
          offset: next(offset-pat))
```

GRAINS WITH SEQREP

Generating grains using the seqrep construct

Implementation Using Seqrep

```
seqrep(i, 2000,
 set-logical-stop(
   grain(offset: next(offset-pat)) ~
     next(dur-pat),
   0.05 + rrandom() * 0.01))
```


Examples

- See granular.sal

Extensions

- Continuous control of parameters like pitch and rate of travel through file: s-ref(*sound, time*)
- Use amplitude in file to vary rate of travel to time-expand attacks

