INTRODUCTION TO
COMPUTER MUSIC

Roger B. Dannenberg
Professor of Computer Science Art, and Music
What’s In This Introduction?

- Why Computer Music?
- What is this course about?
  - Computer Music Technology
  - Making Music With Computers
- How is the course taught?

Why Computer Music?

- No limits to the range of sounds you can explore.
- Precision:
  - Microscopic changes to sounds
  - Exactly reproducible, incremental changes
- Computation
  - Decisions can be embedded at any level
  - Detailed and complete performances can be recomputed after high-level, abstract changes
- Blurring the lines between composer, performer, and even the audience
What Is This Course About?

• Computer Music Technology
  • Theory
    • Digital audio, Digital signal processing
    • Software design, languages
    • Data structures and Representation
  • Practice
    • Nyquist: a composition and sound synthesis language
    • Audacity: a digital audio editor

What Is This Course About? (2)

• Making Music With Computers
  • Theory: listening and discussion
  • Practice: composition assignments
Projects

• Schedule is on the web
• All projects are due 11:59pm on the shown due date
• You have 3 grace days for the entire semester (P1 through P6 only) - use them wisely.
• You can use at most 1 grace day for any one assignment
• Many additional problems to be worked out as you go through the on-line lectures.

Communication

• Web Site – www.music.cs.cmu.edu/icm
• Online Instruction – www.music.cs.cmu.edu/atutor
• Syllabus is on Web Site
• Project info too
• Come to class (!)
• Discussion/Newsgroup/Bboard for class:
  • https://piazza.com/class/i4mwwndya5f6qj
• Projects handed in to Autolab (See instructions in syllabus.)
Computing Hardware

- You must have access to a machine for homework and projects (there is no class studio, lab, or cluster):
  - Mac, Windows, or Linux is OK
  - No machine is too slow
  - 5MB/minute/channel of audio
  - Projects will be submitted via network to Autolab
    (if you have network problems, you can bring a CD-R or flash drive to campus and upload to Autolab)

Software

- Nyquist is a *self-contained* language for sound synthesis and composition.
- Audacity is an audio editor.
- Both are open source and free.
- You may use other languages, editors, and tools
  - … *in addition to* but not usually *instead of* Nyquist,
  - Project descriptions will generally ask for certain things to be done in Nyquist
Welcome to the Course

- Now is a good time to download Nyquist
  - http://www.cs.cmu.edu/~music/nyquist/
- and Audacity
  - http://audacity.sourceforge.net/download/
- Install them and get ready to make some music!

HOW DO COMPUTERS MAKE SOUND?

Some fundamentals
How Do Computers Make Sound?

• What is sound?
• What does analog mean?
• Digital audio representation
• Analog-to-Digital conversion
• Digital-to-Analog conversion
• Synthesis example

What Is Sound?

• Sound is a variation in pressure
• Pressure variations travel through air as waves
• Sound travels about 1000 feet/second
• Hz = Hertz = (cycles) per second
• We hear variations from about 20Hz to 20000Hz
• We hear amplitude variations over about 5 orders of magnitude from threshold to pain
What does analog mean?

- Pressure variations (sound) can be expressed as:
  - Mechanical displacement (microphone, speaker)
  - Voltage variations
  - Wiggles in vinyl record grooves
  - Degree of magnetization on tape
  - Optical density in film

- These representations are called analog

Digital Audio Representation

- Measure an analog signal periodically:
Digital Audio Representation

- Measure an analog signal periodically:

- Store the measurements as a sequence of numbers

Digital to Analog Conversion

- Use the sequence of numbers to control voltage
- Filter the voltage to produce a smooth signal
Synthesis Example

To compute each sample:

tlen = 1024 // table length
sr = 44100.0 // sample rate

phase += freq * tlen / sr
// phase wraps around table:
phase = fmod(phase, tlen)
samp = table[floor(phase)]
output = samp * ampl

NYQUIST, SAL, LISP

Getting started with Nyquist
Nyquist, SAL, Lisp

Nyquist IDE

Java Runtime System

SAL

Modified XLISP

Operating System

Linux, Mac OS X, or Win32

SAL, an imperative language

command-line Lisp interpreter

Nyquist: Top-Down

- NyquistIDE written in Java (requires Java runtime)
- interacts through sockets with SAL, written in XLISP
- XLISP is interpreted, written in C
- C is of course compiled to your native instruction set
- But there’s more:
  - XLISP is extended with signal processing primitives
  - Written as high-level specifications (see Nyquist Ref. Manual)
  - Translated by XLISP program (tran.lsp) into C
- And more ..
  - score data structures are interpreted by a built-in function (timed-seq) that calls on the XLISP eval function.
Read-Eval-(Print) Loop

- You enter commands into SAL
- SAL reads the command and compiles it to XLISP
- XLISP evaluates the compiled command
- This may or may not generate output

Some Examples

- `play pluck(c4)`
- `play pluck(c4) ~ 3`
- `load "pianosyn"`
- `play piano-note(5, fs1, 100)`
- `play osc(c4)`
- `play osc(c4) * osc(d4)`
- `play noise() * env(0.05, 0.1, 0.5, 1, 0.5, 0.4)`
Some SAL Commands

- `print expression` - evaluate and expression and print the result
- `exec expression` - evaluate expression but do not print the result
- `play expression` - evaluate and expression and play the result, which must be a SOUND
- `set var = expression` - set a variable

CONSTANTS, VARIABLES, FUNCTIONS

More of the Nyquist (SAL) language
Constant and Variable Expressions

- Constants evaluate to themselves, e.g. 12 or "string"
- Symbols denote variables and evaluate to the variable’s value (static scoping), e.g. x or volume or g4 or tempo
- Symbols can contain *, -, +, and many other characters you might not expect. Lisp conventions:
  - *global-variable*
  - local-variable
- Not case sensitive!

Applying Functions

- Don't forget to use set, exec, print, etc...
- Infix operators mostly as you would expect
  - a + b
  - 10 * (y + 3.14159)
- Built-in and user-defined functions
  - autonorm-off()
  - lfo(5.9)
  - string-left-trim(input, " ")
  - s-read("vn.wav", time-offset: 1.5, dur: 0.6)
USING SAL, DEFINING FUNCTIONS

More on Nyquist and SAL

Using the SAL Interpreter

• We’ll do a lot of work on-line, but you need learn how to use Nyquist and SAL on your local machine.
• Nyquist installation and startup:
  http://www.cs.cmu.edu/~music/nyquist/
• Now, we’ll cover:
  • Evaluating a SAL command
  • Finding the result of a SAL command evaluation
  • Creating a SAL program file
  • Loading (executing) the SAL program file
  • Saving your work
Evaluating a SAL Command

Nyquist window

Type the SAL command here and type the return key

Finding the Result of a SAL Command

input window

Nyquist window

Command and result appear here.

output window
Creating a SAL Program File

Click on New File button...

... to create SAL Program File window

Click on Save File button...

... enter name with .sal extension ...

... and click Save button.
Loading (Executing) the SAL File

- Edit the file...
  - select File ...
  - select Load...
  - results appear in Output Window

Saving Your Work

- The Load menu item
  - Saves your file
  - Instructs Nyquist to load your file
  - Nyquist then evaluates each command in the file
  - thus, file saving is automatic!
- There is also a File:Save menu item
- … and a File:Save As… menu item
Defining Functions in SAL

```
define function my-function(p, q)
    begin
        print "the value of p is", p
        display "furthermore", q
        return p + 12
    end

Call it:

SAL> print my-function(c4, "middle-C")
the value of p is 60
furthermore : Q = middle-C
72
```

Concept checklist:
- define function
- What does begin-end do?
- What does print do?
- What does display do?
- Why is 72 printed?

Keyword Parameters

- Keyword parameters are optional parameters that are matched by keyword rather than by position
```
define function kwdemo(p, scale: 1, vibrato: nil)
    begin
        with s = pluck(p) * scale
        if vibrato then
            set s = s * (1 + lfo(6) * 0.1)
        return s
    end

play kwdemo(c4); uses default keyword parameter
values
play kwdemo(ef4, vibrato: #t); turn on vibrato
play kwdemo(fs5, vibrato: #t, scale: 0.4); order
doesn't matter
```
Begin and With

```
begin
    with local-variable = 4,
        another-local ; default init to nil
    command1
    command2 ; any number of commands here
end
```

- Use `begin-end` any place you can use a statement
  - function body
  - multiple actions after `then`
- `with` introduces local variables
  - Initialization is optional
  - Default initial value is `nil` (means both “false” and “empty list”)

CONTROL CONSTRUCTS

Conditionals, Loops, and More Fun
With Nyquist
**If-Then-Else**

```plaintext
if pitch > C4 then
    return flute(pitch)
else
    return tuba(pitch)
```

- **then** and **else** are followed by single command
- Use **begin-end** to contain multiple commands
- Avoid `if c1 then if c2 then s1 else s2`
  - Which **if** does the **else** belong to?
  - Use **begin-end** to disambiguate

**Loop Command**

- Basic syntax is just `loop commands end`
- Use **with** to declare local variables
- `for i from 0 below 10 - i = 0, 1, ... 9`
- `for elem in my-list` - iterate over list elements
- `for v = init then update` - flexible update
- `while expression` - arbitrary stop condition
- `until expression` - arbitrary stop condition
- `repeat n` - iterate n times
- `finally return local-variable` - executed once at end

- Many options! See Ref. Manual. Next up: SAL examples...
Example

function pluck-chord(pitch, interval, n)
begin
    with s = pluck(pitch)
    loop
        for i from 1 below n
            set s += pluck(pitch + interval * i)
        end
    end
    return s
end

play pluck-chord(c3, 5, 2)
play pluck-chord(d3, 7, 4) ~ 3
play pluck-chord(c2, 10, 7) ~ 8