Chapter 2. Meeting 2, Foundations: Musical Parameters, Mappings, and Tools

2.1. Announcements

- If you have not downloaded and installed Python and PD-Extended, please do so now
- Download: most recent athenaCL

  http://code.google.com/p/athenacl

2.2. Overview

- Events
- Parameters
- Containers
- Instruments
- Generative software tools
- athenaCL and Python
- Digital Audio Workstations

2.3. Musical Events

- The event is the fundamental unit of music
- An event can be single sample lasting 0.0000227 seconds
- An event can be a note
- An event can be a continuous sound encompassing a complete work
- The minimum definition of an event is a start and end time

2.4. Events and Parameters

- An event can be described as with one or more parameters
- Parameters may be duration, pitch, amplitude, or any other collection of specifiers
- Parameters may be coordinated or independent
- Human musical production often coordinates parameters
- Independent musical paramerers can make interest musical structures
- The parameterization of musical events has been critical to the development of modern music

2.5. Event Lists

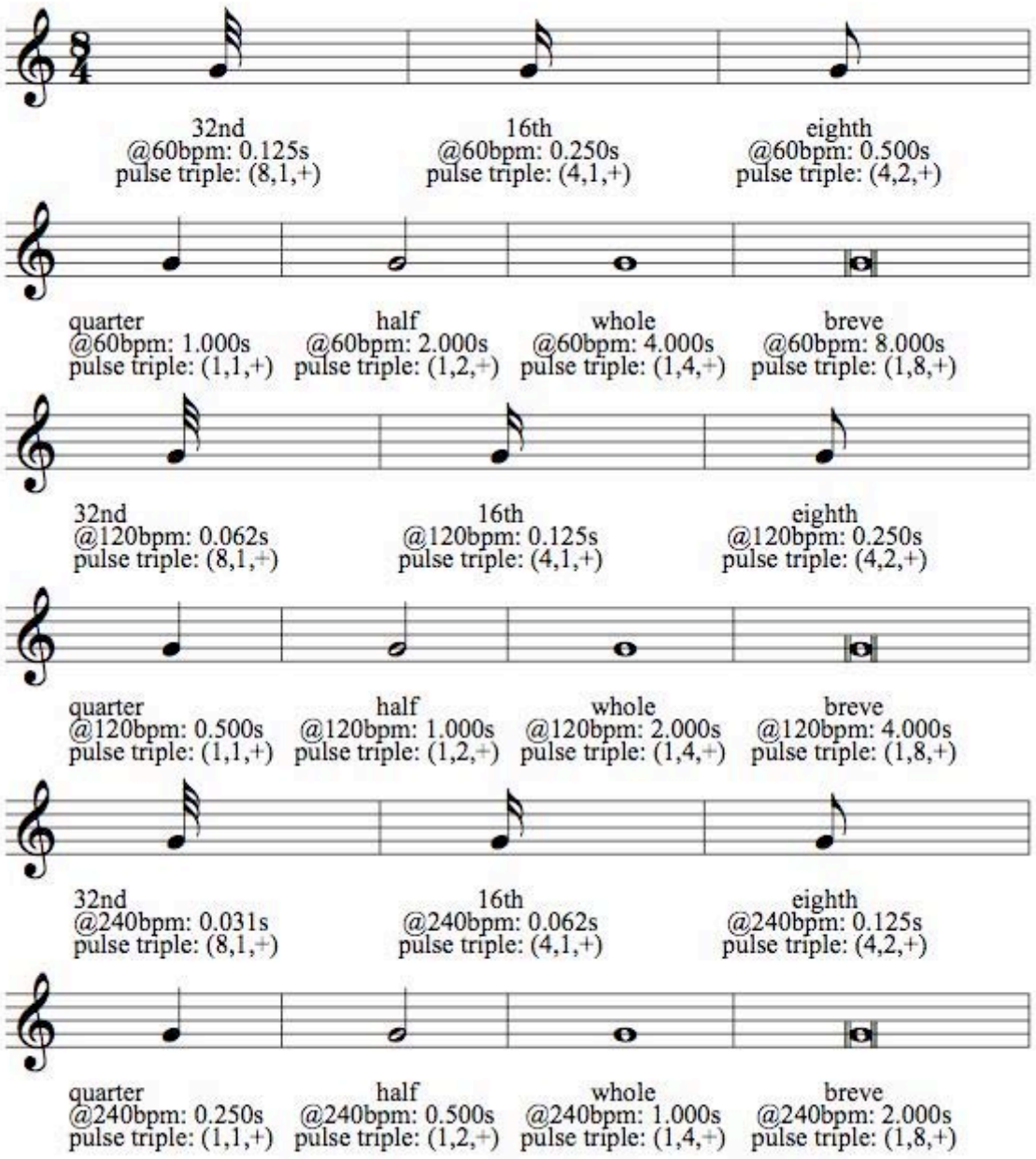
- Events, defined by an array of parameters, can be collected in a list
- Musical data is stored in various arrangements of event lists

2.6. Fundamental Musical Parameters

- Duration and rhythm
- Frequency and pitch
- Amplitude and dynamics

2.7. Parameters: Duration and Rhythm

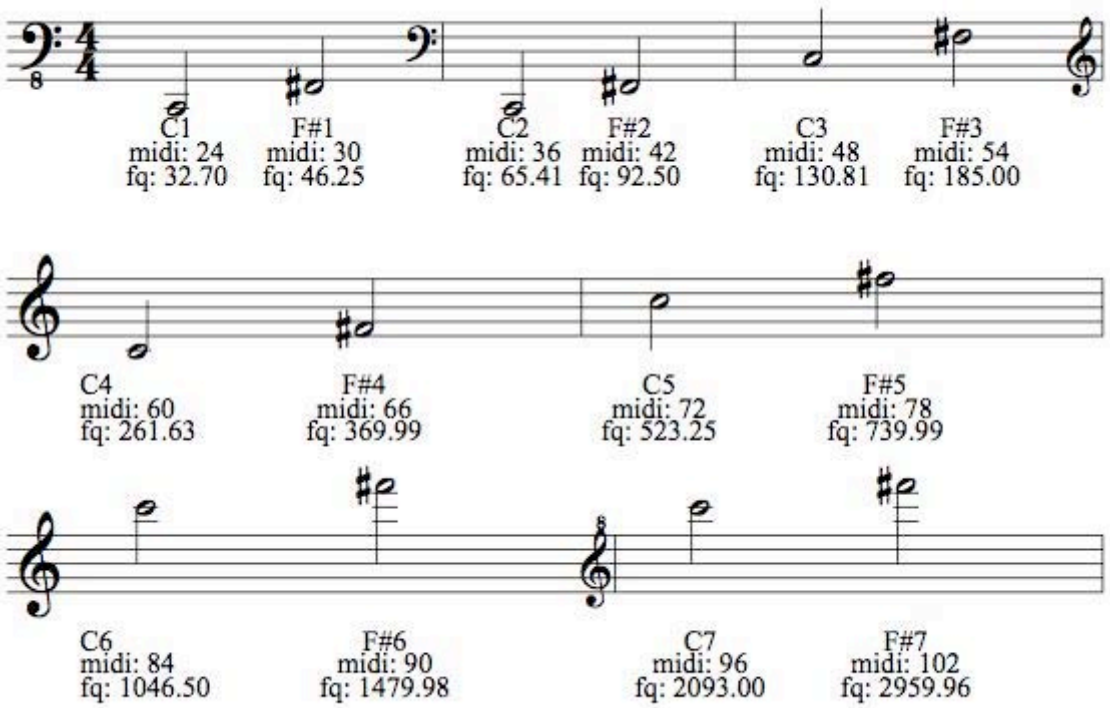
- Can be measured in absolute or relative values
- Absolute values: seconds, milliseconds
- Relative values
  - Notation: quarter, sixteenth, whole
  - Pulse triples: (divisor, multiplier, accent)
- Relative values proportional to a beat rate (tempo)
- Tempi are often thought of in beats per minute (BPM)
- A range of durations at different tempi [py/demo/parameterDuration.py]



2.8. Parameters: Frequency and Pitch

- Pitch is a human interpretation of frequency
- Pitch asserts the octave as referential unit of equivalence
- An octave is 12 half steps, 8 diatonic steps (white notes on the piano), and a 2:1 frequency ratio
- Numerous other distances between pitches (intervals) have names: fifths, thirds, 13ths, quarter tones

- Pitch names can carry octave designation, where C4 is middle C
- MIDI pitch values place C4 at 60, use 1 as a half step, and range from 0 to 127
- athenaCL pitch space values place C4 at 0 and use 1 as a half step
- A range of fundamental pitches [py/demo/parameterPitch.py]

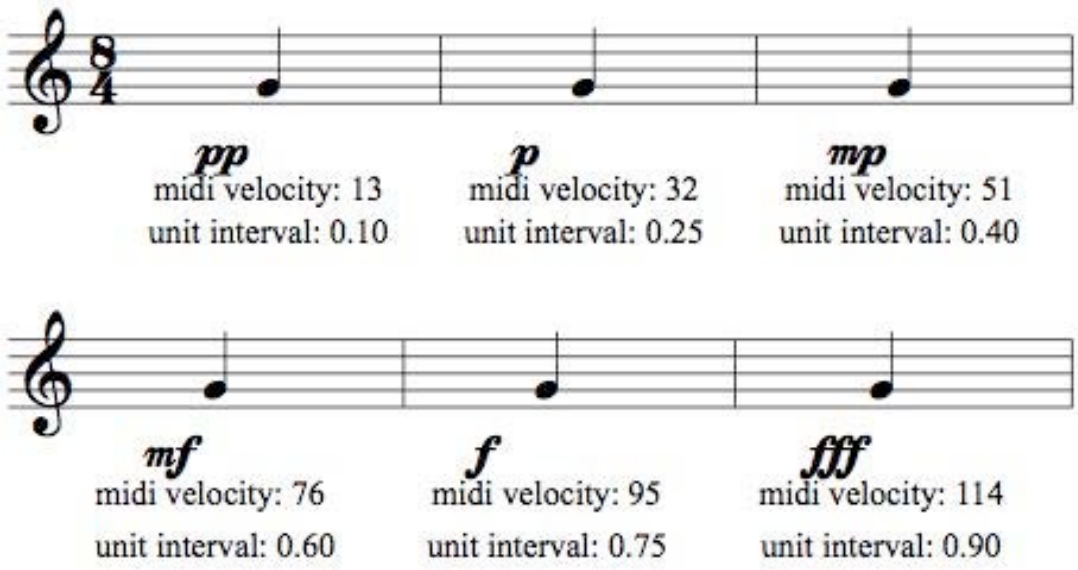


- The (ideal) audible frequency range: 20 Hz (MIDI 16, E0) to 20000 Hz (MIDI 135, D#10)
- Top three octaves (from 3-6k, 6-12k, 12-24k) contain spectral frequencies

2.9. Parameters: Amplitude and Dynamics

- Bits: discrete digital audio amplitude levels
- dB SPL: acoustic power
- dBv: voltage amplitude
- Unit interval spacings: between 0 and 1
- Notation: from *ppp* to *fff*

- MIDI velocity values from 0 to 127
- A range of amplitude levels [py/demo/parameterAmplitude.py]



2.10. Storing Event Lists in Containers

- Events can be streamed in real-time or stored in containers
- Western notation (scores, MusicXML)
- Musical Insturment Digital Interface (MIDI)
- Open sound control (OSC)
- Digital audio files

2.11. Containers: Western Notation

- Events are organized around notes that specify pitch and duration
- Parameter values are limited (mostly) to symbols
- Parameters are isolated for instruments by staves
- Parallel staves express simultaneous events
- Timbral specification relies mostly on instrument assignments

- MusicXML offers a standard for encoding notation
- Software permits opening, editing, and playing MusicXML files
  - Finale and Sibelius
  - Finale reader

    http://www.finalemusic.com/Reader

2.12. Containers: MIDI

- A binary representation of musical parameters
- Parameter values are often 7 bit, or 128 discrete values
- Parameters are isolated by numerical tags, called channels
- Timbral specification relies mostly on instrument assignments (programs)
- Software permits performing MIDI files
  - QuickTime and Windows Media Player
  - Virtual instruments

2.13. Containers: OSC

- A hierarchical representation of musical parameters
- Parameter values can be numbers or strings
- Parameters are organized hierarchically with URL-like syntax
- Timbral specification relies mostly on receiving device
- Sending and receiving OSC data
  - Hardware controllers
  - Software controllers

2.14. Containers: Digital Audio

- A micro mono-parameter representation
- Store amplitude values within a dynamic range taken at a sampling rate

- Signals can be mixed or stored in isolated channels
- Digital audio is a timbral specification
- Software permits playing and editing digital audio
  - QuickTime and Windows Media Player
  - Audacity

    http://audacity.sourceforge.net
  - Digital Audio Workstations

2.15. Synthesizers, Samplers, and Virtual Instruments

- Acoustic instruments translate parameters into acoustic sound
- Electronic instruments synthesize tones with oscillators or stored samples
- Digital electronic instruments are built by combining basic software components
- Virtual instruments are software synthesizers or samplers that respond to MIDI or OSC parameters

2.16. Digital Synthesizers

- Built from combing fundamental signal generators and processors (unit generators or Ugens)
- Can be designed to accept any number of initial event parameters
- Can be designed to accept dynamic parameters over the course of an event

2.17. Digital Synthesis Languages: Csound

- Developed in part from the first synthesis language Music 1 in 1957 (Roads 1980)
- Extended and ported by Barry Vercoe at MIT (1986)
- A huge library of processors and instrument models (Boulanger 2000)
- A low-level language for defining instruments
- A flat list of data for event lists

2.18. Digital Synthesis Languages: PureData

- Over 20 years of development in synthesis, sampling, and a visual programming envrionment
- Numerous related alternatives: Max/MSP, jMax, Open Sound World (OSW)
- Developed by Miller Puckette, creator of the first Max (Puckette 1985, 1988, 1997, 2002)

2.19. Digital Synthesis Languages: SuperCollider

- An extension of Csound archetypes into a modern language and network archetype
- First released in 1996 by James McCartney (McCartney 1996; McCartney 1998)
- A complete object-oriented language: create objects, manipulate, and reuse code
- A server-based architecture: SynthDefs live on a server and send and receive messages and signals
- Designed for real-time performance and experimentation

2.20. Virtual Instruments

- Software plug-ins that can receive MIDI or OSC messages
- Distributed as VST, AU, or other plug-in formats
- Can employ any internal software and synthesis model

2.21. Algorithmic Composition and Generative Music Systems

- May be built within a synthesis language
- May be stand-alone systems
- Numerous systems support multiple output formats from a single interface
  - athenaCL

    http://code.google.com/p/athenacl
  - AC Toolbox: Lisp based Macintosh application/environment

    http://www.koncon.nl/downloads/ACToolbox/
  - Open Music: Lisp based visual programming language

http://recherche.ircam.fr/equipes/repmus/OpenMusic/

- Common Music: Lisp based programming language

  http://commonmusic.sourceforge.net

2.22. A Brief History of athenaCL

- Started as a way of automating the production of Csound scores in 2001
- Originally attempted to integrate a variety of post-tonal music theory tools
- Gradually became a more general tool for composition
- A way to test and deploy modular approaches to generating music parameters and structures
- Support for output in MIDI, SuperCollider, and other formats incrementally added
- Version 2 strips away post-tonal music theory tools, focuses on compositional tasks
- Present alpha releases may have bugs: please report any problems to me immediately

2.23. Installing and Running athenaCL

- Download the most-recent version
  - A distribution from Google Code

    http://code.google.com/p/athenacl
  - Via SVN command-line argument:

    ```
    svn checkout http://athenacl.googlecode.com/svn/trunk/ athenacl-read-only
    ```

- Install in Python's site packages
  - Windows: run athenaCL.exe installer
  - Others: extract athenaCL.tar.gz

    With terminal, cd to athenaCL directory

    Enter: python setup.py install

    If permissions error, try: sudo python setup.py install
- Start Python

    - Windows: run python.exe or IDLE.py
    - Others: open terminal, enter: python
- Start athenaCL
    - From within Python, enter: from athenaCL import athenacl
    - Others: open terminal, enter: python

      Enter: from athenaCL import athenacl

2.24. Running athenaCL Without Installing

- Download athenaCL (as above)
- Launch the file athenaCL/athenacl.py with Python

2.25. athenaCL: System Overview

- Create and edit Textures (TextureInstances) and Paths (PathInstances)
- Paths are static pitch collections
- Textures are dynamic variable parameter event list generators
- TextureModules define various approaches to create Textures
- ParameterObjects are used to configure and generate parameters within Textures
- EventModes define orchestras of instruments and available output formats
- EventOutputs are output formats, some available with all EventModes, others available from only one
- EventLists can be created, rendered, and heard

2.26. Interactive athenaCL Commands

- athenaCL as an interactive command line program
- Commands can be provided with space delimited arguments, or the user can be prompted for all necessary arguments
- Acronyms are always accepted for arguments

- cmd: view all commands
- ?: get help for any command
- EMo: select EventMode midiPercussion
- EMi: list available instruments
- TIn: create a new TextureInstance (provide name and instrument number)
- ELn: create a new EventList
- ELh: hear (or open) a new EventList
- Commands with full arguments and sample output

```
pi{}ti{} :: emo mp
EventMode mode set to: midiPercussion.

pi{}ti{} :: tin a 50
TI a created.

pi{auto-highTom}ti{a} :: eln
command.py: temporary file: /Volumes/xdisc/_scratch/ath2010.02.04.09.45.48.xml
      EventList ath2010.02.04.09.45.48 complete:
/Volumes/xdisc/_scratch/ath2010.02.04.09.45.48.mid
/Volumes/xdisc/_scratch/ath2010.02.04.09.45.48.xml

pi{auto-highTom}ti{a} :: elh
EventList hear initiated: /volumes/xdisc/_scratch/ath2010.02.04.09.48.11.mid
```

- Setting the scratch directory to "/Volumes/xdisc/_scratch"

```
pi{}ti{} :: apdir x /Volumes/xdisc/_scratch
user fpScratchDir directory set to /volumes/xdisc/_scratch.
```

- Editing the Texture's temp with a WaveSine generator (space divisions matter)

```
pi{auto-highTom}ti{a} :: tie b ws,t,10,0,40,400
TI a: parameter bpm updated.

pi{auto-highTom}ti{a} :: eln; elh
```

2.27. Automating athenaCL Commands with Python

- athenaCL command can be scripted and controlled in Python script
- Permits reuse and extensions
- Must create an athenaCL Interpreter object and send string commands
- Creating a Python script file
  - Windows: use IDLE.py or another text editor

- Others: use emacs, vi, or other text editor
- Mac: use TextWranger (free)

  http://www.barebones.com/products/TextWrangler

- Automating the production of one Texture: create file 02a.py and run with python [02a.py]

```
from athenaCL.libATH import athenaObj

ath = athenaObj.Interpreter()

ath.cmd('emo mp')
ath.cmd('tin a 45')
ath.cmd('tie b ws,t,10,0,40,400')
ath.cmd('eln')
ath.cmd('elh')
```

- If Python cannot find the athenaCL directory (because you were not able to do an install) you must provide to python the file path to the directory containing athenaCL

```
import sys
sys.path.append("/path/to/dir/that/contains/athenacl")

from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()

ath.cmd('emo mp')
ath.cmd('tin a 45')
ath.cmd('tie b ws,t,10,0,40,400')
ath.cmd('eln')
ath.cmd('elh')
```

- Automating the production of three Textures [02b.py]

```
from athenaCL.libATH import athenaObj
import random

ath = athenaObj.Interpreter()

ath.cmd('emo mp')
for x in [45, 51, 75]:
    ath.cmd('tin t%s %s' % (x, x))
    ath.cmd('tie t %s,%s' % (random.choice(range(0,10)),
                             random.choice(range(20,30))))
    ath.cmd("tie b ws,t,10,0,40,400")
ath.cmd('eln')
ath.cmd('elh')
```

- If you have trouble running a Python script on Windows, visit:

  http://www.python.org/doc/faq/windows/

21M.380 Music and Technology: Algorithmic and Generative Music
Spring 2010