Chapter 2. Meeting 2, Foundations: Sounds, Signals, Samples, and Encodings

2.1. Announcements

• Reading and Listening Discussion Leader assignments will be posted today
• Note that I will not comment directly on posted notes (but do check that they are there)

2.2. Reading: Wang, A History of Programming and Music

• What non-software programming interfaces for music does Wang describe?
• What are some of the fundamental concepts shared by many music programming languages?
• What is the trajectory of programming languages proposed?

2.3. Reading: Puckette, Max at 17

• Is there one Max?
• What was the background development of Max?
• What is max good at? What is it not good at?
• What roles do style and aesthetic play in computer music software design?

2.4. Starting Pd, The Pd Window

• The Pd Window is the destination of all error messages and message sent with [print]
• The “compute audio” toggle
  • Controls all signal processing generation
• Can also be toggled in Media menu, with key strokes, and also with text commands (to be shown later)
2.5. Basic Components

- Patches: windows or collections of windows
- Object boxes: process or create data or signals
- Message boxes: store data
- Data
  - Can be “event” data or “signal” data
  - Passed via “patch cables” between boxes
- Comments: notes to yourself
- Interface objects: number boxes, slides, signal boxes, etc.

2.6. The Patcher Window

- A window represents a patch
- Windows can communicate between each other
- A patch can be embedded in another patch using [pd name]
- A patch can be stored as a file and loaded into another patch, called an abstraction

2.7. The Patcher Window: Edit and Run Modes

- Patch windows have two modes: edit and run
- Changing modes: Menu: Edit > Edit mode (command E)
- Edit mode: configure objects, create patches, move things around, selected objects are blue
- Run mode: objects do not move, user interface components (knobs, sliders) function
- Example: Put a Vslider; when blue, in edit mode, cannot use slider; in run mode, black, can use slider

2.8. Object Boxes

- An object is generally a computational subroutine
- An object has a class: a paradigm, an archetype
• We can make many instances of the same object, each with its own behaviour and settings
• Example: [random 20], [random 4]

2.9. Object Boxes: Creation
• Use the Put menu: Menu: Put > Object (command 1)
• An empty dotted-line box emerges: this is not an object
• An object has to have at least one creation argument to specify its type
• Additional arguments can be used to configure the object
• Example: [+], [random], [line], [select], [print], [osc~]

2.10. Objects: Types
• There are event (control rate) objects and signal objects
• Event objects process data: [line], [select]
• Signal (tilde) objects process signals: [line~], [osc~]
• There may be two versions of a type of object, one for events, one for signals: [+], [+~]

2.11. Object Inlets
• Inlets provide data or signals into objects (not both)
• White (hollow) inlets are for data, dark (filled) inlets are for signals
• Example: [+], [+~]
• For many event objects, leftmost inlet is hot: output is provided only when values are provided in this inlet
• Example: [+], [pack]

2.12. Object Outlets
• White (hollow) outlets are for data, dark (filled) inlets are for signals
• Example: [+], [+~]
• Outlets almost always provide output from right to left

• Example: [unpack f f f]

2.13. Object Interconnections

• Connections between objects can transmit either signals or event data

• Signal and event data connections are different, and cannot be interconnected

• To create a connection: in Edit mode, mouse over outlet until cursor is a circle; click and hold; mouse over desired inlet until cursor is a circle; release click.

• Example: [* 4] to [+ 3], [*~ 4] to [+~ 3]

2.14. Data

• Data can be bangs, numbers, symbols, lists, or signals

• Bangs (b): a trigger, an event, a “do something now”

• Numbers (f): all numbers are floating point values

• Symbols (s): character strings (not in quotes)

• Lists (l): a space separated collection of numbers or symbols

• Signals (v): floating-point number stream at the sampling rate (when “compute audio” is on)

2.15. Data Storage

• Data can be seen (in objects, interfaces, etc) and unseen (in objects, through patch connections)

• Only data that is “seen” is saved with patch

2.16. Data Storage: Object Boxes

• Objects can have additional construction arguments

• These arguments configure how the object performs on initialization

• These arguments can sometimes be overridden by inlet values

• Example: [* 2]
2.17. Data Storage: Message Boxes

- Use the Put menu: Menu: Put > Message (command 2)
- One inlet, one outlet; note curved left side distinguishes message boxes from object boxes
- Store bangs, numbers, symbols, or lists
- Saved with patches
- Provide a user interface: can be clicked in Run mode to provide output
  - Example: (bang) to [random 10] to [print]
  - Example: (3) and (10) to [+ ] to [print]

2.18. Interface Objects: Number Boxes

- Can be used to provide numerical inputs to other objects
- Can be used to receive the numbers outputted from objects
- Can be varied as a GUI only in Run mode
- Important: holding down shift permits enter floating point values
- Min and max values can be set with object properties

2.19. Interface Objects: Bang

- Can click to send a bang
- When receiving a bang, darkens
- Sending a bang can be replaced by a message box with “bang” specified

2.20. Selecting, Moving, and Copying Objects

- Objects can only be moved in edit mode
- Can click and drag to create a selection area
- Objects (and interconnections) can be duplicated and copied
- Copying and pasting overlays existing objects: always duplicate
2.21. Object Help, Documentation, and Tutorials

• Control click on an object and select “help” to view a help patch
• Demo patches (when available) provide examples and explanation
• The PD Glossary [http://www.flexatone.net/docs/pdg](http://www.flexatone.net/docs/pdg)

2.22. Object Properties

• Control click on a bang interface object and select “properties” to specify visual appearance
• Colors and other attributes can be configured

2.23. Comments

• Comments are notes left to readers of the program
• Comments cannot be used as data in a patch
• Comments are critical and are essential in good code
• Use the Put menu: Menu: Put > Comment (command 5)

2.24. Saving Patches and PD files

• Always save files with a .pd extension at the end
• PD files are text files that specify the interconnections between objects

2.25. Abstractions and Martingale

• Abstractions are PD patches that can be used in other PD patches
• Abstractions may have any number of inlets or outlets
• To load an abstraction, it must be placed in a directory that PD knows about
• Download Martingale manually: [http://code.google.com/p/martingale/](http://code.google.com/p/martingale/)
• Add the “martingale/pd/lib” directory to Preferences > Path; this permits loading abstractions from the martingale library
2.26. Noise

- Noise at the audio rate is random amplitudes, scaled between -1 and 1
- White noise produces equal energy across entire spectrum
- Source of rich signals and randomness
- \([\text{noise}\sim]\) object provides random audio rate values between -1 and 1
- Example: martingale/demo/signalWaveforms.pd

2.27. Mouse State Noise

- 1. Connecting Noise to output; scaling amplitude with \([\ast\sim]\), turning DSP on and off with message boxes

- 2. Smoothly controlling amplitude with \([\text{mgUiMouseState}]\), \([\text{sig}\sim]\), and \([\text{lop}\sim 20]\); conversion of event data to audio rate data
• 3. Performing subsonic amplitude modulation (tremolo) with [cycle~] and [mgRectify~]
• 4. Scaling unit interval values with [mgScaleMinMax~]
2.28. Sines

- Sine waves provide a perfect circular motion over time
- Produces single, perfect frequency with no overtones
- Example: martingale/demo/signalWaveforms.pd
- Audible range from 20 to 20,000 Hertz
  - Example: martingale/demo/earLimits.pd
- Frequency is logarithmically related to pitch; equal pitch values are octaves, or a 2:1 frequency ratio
  - Example: martingale/demo/earLogFrequency.pd
- MIDI pitch values are an integer to half-step mapping; can convert from MIDI to frequency with [mtof] and [ftom]
2.29. Mouse State Sines

- A mouse theremin: y axis controls amplitude, x axis control pitch (scaled between MIDI values 40 and 64 and converted with [mtof~])

![Diagram of a mouse theremin setup]

2.30. Harmonic Waveforms and Wavetables

- Anything other than a sine tone has a rich (or richer) spectrum
- Many naturally resonating bodies produce secondary vibrations at whole-number multiples of the base frequency
- Common waveforms represent common arrangements of overtones produced by summing harmonic overtones: triangle, square, and sawtooth
- Example: martingale/demo/sumOfSines.pd
- Example: martingale/demo/signalWaveforms.pd
- A wavetable is an array that stores wave patterns (or other data) and reads them back at variable rates
• Arrays store (generally large) lists of values indexed from zero
• Each array in Pd must have a unique name; names can be provided as arguments

2.31. Mouse State Harmonic Drones
• Mixture of detuned saw and square waves; y axis controls amplitude, x axis controls tremolo

2.32. Listening: Schaeffer
2.33. Listening: Cage and Oswald

- Listening: John Cage, *Williams Mix*, 1952

- John Oswald, “Dab,” *Plunderphonics 69/96*, 1989

2.34. Stored Digital Audio

- Audio file data can be loaded into arrays and treated as a wavetable

- Audio files may have different bit depths and sampling rates; when loaded in Pd amplitudes range from -1 to 1

2.35. Mouse State Audio File Looper

- Variable rate audio file looper: y axis controls amplitude, x axis control rate of playback from -2 to 10
2.36. Pd Tutorial 1

1. The following examples demonstrate operations with Pd. Recreate the following patch components in a Pd file and answer the provided questions as comments in the Pd file.
2. In a Pd file, re-create one of the demonstrated Mouse State instruments shown above. Extend the instrument in some way: alter fixed parameters, apply alternate mappings of mouse values, combine different sound sources.
21M.380 Music and Technology: Live Electronics Performance Practices
Spring 2011

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.