PD overview

Matthias Rath (XVI ciclo)

Dipartimento di Informatica
Università degli Studi di Verona
rath@sci.univr.it

- General
  - Open Source, originally written by Miller Puckette (author of Max/msp, developed at IRCAM, today distributed commercially)
  - PD is a modular System. It includes a (simple) graphical design interface, but can be used without.
  - Plugin-like architecture: Modules can be written and compiled separately and loaded at runtime. This is the only difference between internals and externals. PD uses its own interface definition, but a Ladspa module enables the integration of Ladspa plugins.
  - Combination of modules in patches; these are stored in "clear" text format, normally built/edited graphically (through GUI).
  - Organization in subpatches is provided, to hide away details and structure the design process.
  - PD communicates to the outside world via special modules like "adc∼", "dac∼", "midiin", "midiout", "print" ... and through GUI elements like "atoms" and sliders ....
- Patching and module communication
  - PD modules take and send audio signals and messages.
    Convention (that should be retained when programming new externals): Modules that process audio signals are marked by a "∼" at the end of their name.
  - Each object class handles a fixed collection of message types. Arriving messages of different type are reported as errors on standard output.

- If not defined specifically otherwise (through a “list” method), objects respond to “list” messages by distributing the list arguments to their inlets.
- There are two types of subpatches: “one–off–subpatches” are stored within the mother patch.
  “abstractions” are stored in a separate file and can be instantiated in foreign patches.
- Only modules and their connections are stored in patches, no states of modules. (Exception: Subpatches.)

- GUI principles
  - Object boxes hold **messages to the pd engine**. They cause PD to instantiate modules.
    Further elements: message boxes and “GUI operators” (atoms, graphs, sliders . . .)
  - Two operation modes: **Edit** and **Run mode**
    Run mode: Input to number-/symbol-boxes and sliders
    Edit mode: Editing boxes, connections and layout; standard editing commands exist.
  - Audio–/message processing is independent from GUI mode.
  - “put” in edit menu instantiates modules, eventually after typing text into the opening box; PD automatically changes to edit mode.
  - Only boxes, their connections and creation arguments are saved within a patch (see above); extra objects for storing further infos exist.
  - “one-of-subpatches” are created or instantiated by typing “pd subpatchname” in an object box.
    “abstractions” are created or instantiated, when only a name (that is not yet reserved for a module) is supplied.
  - Access to properties and help by right click on boxes, in both modes.
- Audio
  - Audio signals are 32bit floats processed at sampling rate; of course, output/input resolution depends on hardware and driver. Input signals are interpreted as values between -1 and 1 (no matter what format: aiff, wav), output is clipped to that range.

  - audio values are processed in buffers of a fixed number of samples (default: 64). Buffersize is set globally by command flag and patch-wide via “block~”; this is also the window size (determining resolution) for “fft~”.
  - Signal and control processing are interleaved, audio ticks are “atomic”: No messages are processed during a dsp cycle (of buffersize).
    $\Longrightarrow$ Precision for external controls or timer objects of e.g.
    $$\frac{64}{44100Hz} \simeq 1.5ms \tag{1}$$
  - No reordering for realtime considerations; determinism is preserved.
  - When audio computation is turned on (i.e. when “pd” receives a “dsp 1” message), or when the audio network is changed while audio computation is running, PD sorts all audio-modules in a linear order.
  - Feedback loops are detected as errors; they can be build with non-local connections (“send~”/“receive~”, “throw~”/“catch~”, “delread~”/“delwrite~”) but feedback signals are only processed with one buffersize delay.
  - “switch~” can turn audio on and off for subpatches.
  - “sig~” and “snapshot~” are converters between audio and control messages.
  - Audio rate of input/output is neither adjusted nor checked; so you must set the one you use on program start (default 44100 Hz).
  - The scheduler tries to keep ahead of realtime (controlled by the “audiobuffer” flag). Computational overload results in dropouts; disk streaming still works correctly.
    Audio engine and gui compete for computational power: avoid drawing during audio processing!

- Messages
  - Messages consist of a selector and any number (incl. 0) of arguments. Arguments can be symbols or numbers.
  - All numbers are interpreted as floats. (PD, as opposed to MAX/msp, does not know an “integer” type.) Messages that transport single values for numerical computations normally have the selector “float”.

- Message boxes send messages (surprisingly), when "something" arrives or when they are clicked in the GUI. The associated module is also called "message".
- Message boxes/modules can keep multiple messages, separated by commas and sent in sequence but at the same logical time. Semicolons may also separate messages; in that case the first symbol gives the "destination", i.e. semicolons "clear the destination": The first destination is the outlet (precisely all the modules connected to it), others are named ("receive"s, "arrays", pd windows or "pd").
- With few exceptions, only the leftmost message inlet of a module/box is "hot": Messages arriving at this inlet cause an output. "cold" inlets normally are "sticky", which means that their values are stored until changed (exception: "line" and "line∼").
- The "trigger" module is used to achieve active behavior of right inlets. This convention is useful: Active behavior can be easily simulated, the converse would cause bigger effort.
- Messages are passed in "depth first" order: If message A is received before B, then all following messages resulting from A are also handled before B. This assures that messages are processed instantaneously (logically) in welldefined order (which depends on order of cabling). To avoid dependency on the order of cabling use "trigger".

  As a consequence infinite loops (without delay) lead to stack overflow.
- Messages that appear/are generated at the same logical time, are also processed at the same logical time in the order of their reception.

  Message flow is **instantaneous**. No reordering whatsoever is done for realtime considerations! Determinism is preserved at "any cost".
- Message arguments starting with "$*number*" are "environment variables":

  In case of a message box/module the environment are the arguments of incoming (possibly "list") messages.

- Note that the text in an object box is sent to PD as a message: Consequently **creation arguments** of abstractions and one–off–subpatches can thus be referred to by "$1", ...

    Special: “\$0” is a unique identifier that is automatically set for each instance.

  - Unspecified environment variables are undefined.
  - GUI convention: Single numbers in a message box (which can not mark a selector, only symbols), are automatically interpreted as float messages. I.e., selector “float” is added automatically.
    Analogously, selector “list” is automatically added, when several arguments starting with a number are typed into a message box.

- Terms
  - “message” also names a module with its own GUI element, the “message box”.
  - “symbol” should maybe be seen as synonymous to “(character)string”: Message arguments can be numbers or **symbols**, the selector is always a symbol; BUT “symbol” (is also a symbol that) names a selector!!!