Computer Models of Sound with Pure Data

Winter 2019

Dr. Martin Jaroszewicz

Contents

Abstract

Pure Data (Pd) is an object based graphical environment for sound synthesis developed by Miller Puckette, professor at the University of California San Diego. In Pd you can create custom synthesizers, effects, musical patterns, and sonic and musical machines by connecting on-screen patch cords. Most importantly, PD is a great tool for sound research, analysis and re-synthesis [1].

Pure Data can also be used as the sound engine for mobile phone applications. The *libpd* library library is adaptable for use with any language that can support native code. Example projects for IDEs like XCode and Eclipse, and example code for languages like C, Java, Objective-C, and Python –including PyGame– and Processing.

Both Pd and libpd are free, open source, and can run on a wide variety of devices from phones to computers [2].

[1] Pd can be found at: `https://puredata.info/`
[2] libpd can be found at: `http://libpd.cc/`

Know you Mac - Keyboard Shortcuts Combinations

- Single click (select the item)
- Type a name for a file or folder (with item selected -¿ return)
- Tab or arrows (scroll through items)
- Double click (open an item)
- Ctrl-click (contextual menu)
- Cmd-N (new finder window)
- Shift-Cmd-N (new folder in a given window)
- Shift-Cmd-A (go to applications)
- Cmd-K (go to remote storage)
- Shift-Cmd-H (home)
- Cmd-del (move to trash)
- Cmd-O (open)
- Cmd-W (close)
- Cmd-F (search)
- Cmd-1, 2, 3 (change view to icons, list, columns)
- Cmd-Shift-4 (snapshot of screen)
- Cmd-Shift-Ctrl-4 (snapshot to clipboard)
- Ctrl-F2 (activate menu)
- Ctrl-F3 (activate Dock)

Pure Data

Navigation

- Cmd + N (New patch)
- Cmd + E (Toggle Edit mode)
- Cmd + / (DSP On)
- Cmd + . (DSP Off)
- Cmd + 1 (Put object)
- Cmd + 2 (Put message)
- Cmd + 3 (Put number)
- Ctrl + click (Object Properties/Help)

To connect objects, click and drag outlets to inlets. Inlets and Outlets (inputs/outputs) are the dark segments around the perimeter of the object's box.

Control Rate Objects

Basic Objects

print Prints to console for debugging.
trigger | t Splits and sends messages in order from right to left.
float | f Stores a number on the right that can be trigger by sending a *bang* to its left inlet.
list Stores a list.
symbol Stores a symbol.
random Generates random numbers.

Time Objects

metro A metronome that sends a *bangs* at regular intervals.
delay Delays a bang.
pipe Delays a number.

Routing

select Sends bangs if input matches the argument list.
route Split lists. Similar to *select*. Matches first element, let pass the rest of the message.
moses A "switch" that splits numbers when a threshold is past.
spigot A switch that can control stream of messages.
swap It swaps the values on its inlets and send them to its outlets. Can take a constant as an argument. Useful for calculating inverse values e.g (1 -x).
change Only outputs values when they change. It filters repetition.

Messages

send | s Sends a message wirelessly.
; foo Broadcast a message to any destination.

Lists

pack It assembles several items into a lists.
unpack Disassemble list into its components.
load bang Sends a bang when patch is loaded.
lists append list Appends a second list to the first
list prepend Prepends the second list to the first.
list split Splits a list into two.
trim Trims the "lists" selector.

MIDI

note in Reads incoming midi notes and outputs note number, velocity and channel values.
note out Sends note number, velocity and channel to an external device.
ctlin ctlout Reads and outputs respectively continuous controllers (CC) values such as faders and knobs from a MIDI controller.
mtof Converts modi note numbers to frequency in hertz.
ftom Converts frequency values in Hertz to midi note numbers.

Arithmetic

+
-
/
*
div Integer divide
mod Modulo. The smallest reminder of division.

Functions

cos
sin
tan
atan
atan2 Arctangent of the quotient of two numbers
exp Exponential e^x.
log Natural log. Base e.
abs Absolute value.
sqrt Square root.
pow Exponentiate the left inlet to the power of the right inlet.

Comparative Objects

> Greater than, outputs a bang if true.
<
>=
<=

==
!=

Audio Rate Objects

Many objects such as the ones that perform arithmetic operations have their counterpart at audio rate speeds. The difference between them is the symbol " ". For example, to add to signals we use +~.

adc~ Analog to digital converter.
dac~ Digital to analog converter.
wrap~ Constrains a signal between 0.0 and 1.0.
osc~. A cosine waveform oscillator.
phasor~ Asymmetrical periodic ramp wave.
samplerate~ Returns the sample rate.
noise~ White noise generator.
tabosc4~ Implements *wavetable* synthesis using an array.
line~ Signal rate control data vline~ Reads multi segments lists such as an ADSR.

Filters

lop~ Low Pass Filter.
hip~ High Pass Filter.
bp~ Band Pass Filter.
vcf~ Voltage Control Filter. Fast signal rate control of the cutoff frequency and resonance.
biquad~ Static Biquad Filter.

Delay

delwrite~ Creates a buffer for delaying a signal.
delread~ Gets delayed signals from a buffer.
vd~ Variable delay object. Tape echo and doppler-shift type effects.

Order of operations

In Pd, the order of operations or data flow is the following:

- Hot and cold inlets.
  The leftmost inlet is always a "hot" inlet that will trigger the object creating an output when receiving data. All other inlets are "cold" meaning that data sent to them will not trigger an output. For example a "bang" sent to a f left inlet will trigger the number that was stored by sending a value to its right inlet.
- Order when connecting boxes.
  The order of events is determined by the order in which connections were made. This is unknown by a user who did not create the patch. To avoid confusion, it is best to use trigger.
- Depth first message passing.
  Pd will schedule events in the following order:
  1. Right to Left
  2. Depth First

  In other words, the compiler looks ahead and tries to go as deep as it branches.

Subpatches

It is possible to create subpatches by creating an object with the word "pd" followed by an arbitrary name. A new window will pop. Subpatches can be accessed by clicking the object. For example: `pd mySubpatch` will create a subpatch named "mySubpatch". Use the following objects for communicating between subpatches and the main patch,

`inlet myInlet` Creates an input inside a subpatch.
`outlet myOutlet` Creates an output inside a subpatch.

It is possible to create subpatches inside subpatches.

Abstractions

Subpatches are very useful to prevent overcrowding the main patch. In some cases you may want to reuse the same "subpatch" repeatedly. For this purpose, it is better to create an abstraction that can be called at any time. An abstraction lives in a different file under the same path or folder where the main patch is. Abstraction can be reloaded many times by many patches, can have a separate internal namespace and can take creation arguments that can result in different behaviors. They behave like *functions* in text based languages.

Scope

To add unique identifiers to objects to prevent the compiler to call an object that is out of *scope*, especially a duplicate array living in a subpatch, it is useful to rename arrays adding the prefix "$0-" to its name. For example an array named "array1" should be renamed "$0-array1". This property will make the object local to the abstraction because the compiler will replace "$0-" with a unique identifier during runtime.

Dollar sign in messages

A dollar sign "$" in messages will be replaced by data sent to the object. For example, a message `Name is: $1` will be replaced by whichever message is sent to the object. Sending the message "Bob" will output "Name is Bob"

Dollar sign in objects

Using dollar signs in abstraction allows for instantiating the "subpatch" with parameters. Variables should be named with consecutive numbers e.g. $1, $2, $3... For example, if an abstraction `myAbsraction` that contains the objects `f $1` and `f $2` is instantiated with `myAbsraction 50 730`. $1 and $2 will be replaced by the numbers 50 and 730 respectively.

Audio for Virtual Reality

Given that virtual reality engines are within an object oriented framework, in virtual reality applications –including games– everything is an object. For example, Unity and Unreal provide C and C++ Software Development Kits (SDK) to expand the functionality of their core engines. These objects have methods: they move around, they can be destroyed and interacted with. When they collide with other objects, they usually produce a sound ! e.g: Terrain based footsteps varying on type of surface.

Audio Sources

There are two types of audio in virtual reality: Samples and Procedural

- **Samples** are pre-recorded sounds. Depending on the microphones and techniques, sounds can be used for ambiance, collisions, sound spatialization. Note that the audio system of the 3D engine will take care of spatial audio. There are problems when using samples. Examples are aligning loops with movements, sample length, pitch shifting.
- **Procedural audio** is real time audio generation. Parametric or procedural audio does not require the use of samples and can be coupled with machine learning algorithms, being more flexible at runtime. Perhaps the most interesting advantage is the automation of sound generation in real time.

Pedestrian Crossing Scene

Let's create a scene for a virtual reality application !

Tools

- Sound files (recordings of traffic and pedestrian crossing signal button)
- Sound editing software (Audacity) [3]
- Spectrum analyzer (Sonic Visualizer) [4]
- Pure Data [5]

Concepts

- Sound
- Frequency / Sine Wave
- Partials
- Signal Flow (DSP I/O: dac) e.g Delay input. adc =¿ delay =¿ * 0.2 =¿ dac
- Control Rate (Bang)
- Oscillator
- Delay
- Operators on audio signals (+,*)

[3] Audacity
[4] Sonic Visualizer
[5] Pure Data

Workflow

1. In Audacity, edit the sound file so it only contains the sounds to be analyzed. This will make the file smaller for faster analysis. Export spectrograms for documenting your process.
2. Export as a .wav file. This format is better for cross platform editing. Never use .mp3 which appplies a compression algorithm and will discard important spectral data.
3. Perform spectral analysis using Sonic Analyzer. Take notes of the peak frequencies and their composition (harmonics).
4. In Pd, create an additive synthesis model using oscillators and gui elements.