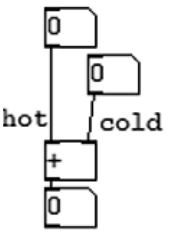
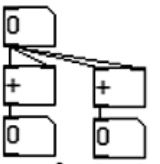
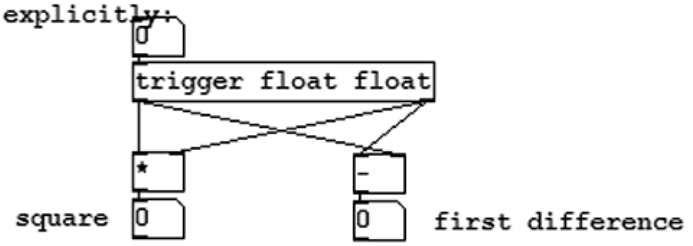


In Pd, most objects carry out their functions when they get messages in their leftmost inlets, and their other inlets are for storing values that can modify the next action. Here, the "+" object does its thing only when the left-hand input changes.

Here's the downside: drag this--->



In Pd you must sometimes think about what order an object is going to get its messages in. If an outlet is connected to more than one inlet it's undefined which inlet will get the cookie first. I've rigged this example so that the left-hand side box gets its inputs in the good, right-to-left order, so that the hot inlet gets hit when all the data are good. The "bad adder" happens to receive its inputs in the wrong order and is perpetually doing its addition before all the data are in. There's an object that exists solely to allow you to control message order explicitly:



Trigger takes any number of "bang" and "float" arguments (among others) and copies its input to its outlets, in the requested forms, in right-to-left order. Hook it to two inputs without crossing the wires and you get the expected result. Cross the wires and you get a memory effect.

updated for Pd version 0.33

An Exciting Journey of Research and Experimentation

Andrey Savitsky

One of the first things that becomes apparent upon the initial encounter with the *Pure Data* (Pd) environment is its versatility and the unlimited possibilities of its utilization. It is not *just* an audio file editor, not *just* a video mixer, a synthesizer, or a graphic design assistant. Pd is all of these things simultaneously and much more. It is capable of handling virtually any type of digital data, from prime numbers, audio-video sets, and MIDI signals up to web events. The end "product", resulting from the processing and manipulation of this vast array of data, is even more variable and unpredictable. If a musician, for instance, wishes to use a computer's CPU or complex mathematical formulas as the data source for an audio synthesizer, Pd would be indispensable for these purposes. On the other hand, exactly the same data sources can also be used to create a *video* stream, since Pd allows for the control of the color characteristics of the video image, the sequence of video fragments, the speed of their playback, or parameters of 3D models. Thus, given the infinite variety of data sources available for manipulation and the limitless array of possible end results, working with Pd becomes an exciting game: a game where the rules are created and changed on the spot by the player; a game where the process of playing may be much more thrilling than the outcome, while the outcome may be unpredictable and quite surprising.

As a musician, I find these qualities of the Pd program very attractive. The past several years, since I started using Pd, have been an exciting journey of research and experimentation with methods of manipulating, interpreting, and transforming audio material. Here I will describe several important concepts of Pd that are most crucial in my personal work with this program.

One of the most important features of Pd is its handling of the audio signal input/output modules as autonomous objects, controlling the direction of the audio streams. Most of the other currently available audio software programs provide only a limited number of possible ways to utilize these basic modules, usually just for the recording (in) and playback (out) of the audio data. Pd, on the other hand, allows for infinite variety of manipulation of input and output modalities, creating a plethora of possible audio streams – parallel or series, discrete or continuous. Most remarkably, the module of audio input in Pd can receive and interpret a signal from the audio output.

This Pd feature opens a vast field of possibilities for sound experimentation, especially useful in real-time live performances. An initial external audio stream (input) can be combined with the audio data resulting from the Pd processing (output). Then these two (or more) sources can be used as a single object for even further manipulation. One can build various algorithms to be utilized as effect processors of the input signal, or one can record audio fragments into the memory buffer and then replay them parallel to the main audio stream. The audio stream

itself can be divided into discrete parallel branches, allowing control of each of them separately. These audio branches can be multiplied, divided, added, deleted, or repeated.

Simultaneous manipulation of the external audio stream and "internal" audio signal, created and mutated within Pd, results in an infinite cycle of sonic transformations. Such audio-loops may become a never-ending source of amusement and pleasure during the live performance. I frequently resort to this method of sound production and manipulation during my own live shows: sometimes one single tone signal played at the beginning of the performance, passed through several virtual samplers and effect processors within Pd, may give rise to an all night long exploration of every nook and cranny of the liquid-fractal aural space.

This possibility of simultaneously recording and replaying the sound generated by Pd without slowing down or halting its other working processes is, in my opinion, the key factor rendering this program so attractive for real-time sound design. Another major advantage is Pd's ability to represent a recorded audio sample as an array of numbers. This permits easy and precise access to any time-point within the sample, by localizing its numerical value. In addition, an audio recording represented by a number set may itself act as a controller of various events; for instance, it may become a sequencer or a set of MIDI-signals. This mode of operative storage of audio samples enables the realization of various refined yet simple ways of audio playback – the sample can be fragmented to pieces of any length, played in any direction, at any speed, or as a loop.

An ability to process MIDI signals is common among most currently available audio software programs, and Pd is no exception. By using a set of modules for processing MIDI data, one can create new MIDI programs that generate the audio output within Pd or receive and process MIDI commands from a variety of external sources (from MIDI sequencers to serial port data readers). Using the VST library, one can create a chain of VSTi synthesizers that are linked together and process the MIDI commands, or VST plug-ins for sound transformation. The number of modules one can employ is boundless. In fact, one of the main advantages of this program is the fact that a musician is not limited by the defined set of modules designed to solve a narrow circle of problems. On the contrary, a specific program module can be designed and "built" to reach virtually any sonic objective of interest. The only limitations for one's imagination would be the computer's performance.

As mentioned above, besides its most common application – audio signal processing –, Pd offers numerous options for working with other types of data. For instance, one may experiment with written texts, statistical data, or visual images, and create their sonic interpretations. It is important to note that any operations occurring within the Pure Data program can be automatized. If one desires to build a robot collaborator for a live performance or multimedia installation – not a problem.

In conclusion, the Pure Data program has a universal modular collection of instruments and offers limitless possibilities for solving nontrivial problems while working with any type of

digital data. Creatively using its various features, multimedia artists or musicians, even those without special programming skills, can relatively easily build programs tailored for the realization of their unique ideas. Pd makes the act of artistic creation an exhilarating game, permeated with a sense of adventure and discovery.