Pure Dataflow - Diving into Pd

By marloes

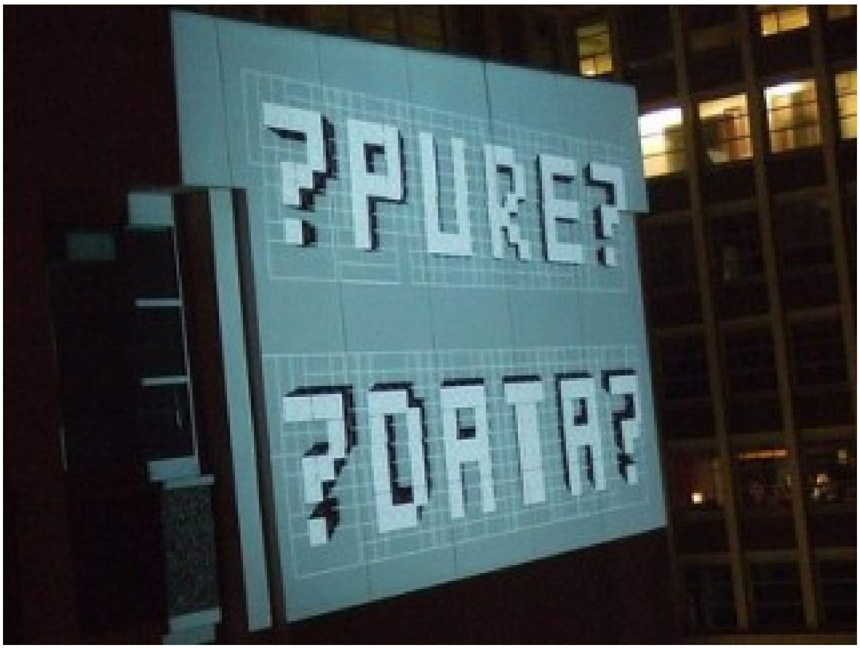
Published: 09/18/2007 - 13:43

Frank Barknecht , September 2007

This article introduces the possibilities of the software Pure Data (Pd), explains a bit why it's so popular among artists and shows what Pd can be used for. The goal is to help artists decide if Pd is a tool for their own work.

Intro

Pure Data, or PD for short, is a software written by mathematician and musician Miller S. Puckette [1] . It has become one of the most popular tools for artists working with digital media. Originally conceived in the late 90s as an environment to create sounds and to compose music, it was soon extended by modules to work with video and graphics. Pd is freely available for no cost, and it is Free Software in that the source code can be obtained, modified and distributed without restrictions as well. Pd runs on many operating systems including the big three: Linux, OS-X and MS-Windows.



Lightshow with Pd/GEM and physical modelling

Over the last decade, the user base of Pd has constantly grown and many of these users have also turned into developers who work on the software itself and make their own extensions. The sheer number of available extra packages may make Pd a bit intimidating to beginners, so this article will present Pd itself and give an overview about the various extensions available. The goal is to enable interested artists to make their own choice about whether Pd is a useful tool for their work.

History repeating

What can I do with Pd? That seems to be such an easy question. But actually it is one of these innocent and smart questions, that children ask and parents cannot answer. At its core, Pd is a full-blown programming language and in theory you can write every possible computer program using Pd. So one answer to the question "What can I do with Pd?" could be: You can use Pd to make a computer do everything that a computer is able to do.

However from everyday experience with a computer it is known that a computer often doesn't like to do at all what a human wants it to do.   Why is this printer suddenly refusing to print? I know it can print, it has printed before! Assuming the hardware is working, many issues humans have with computers are based on communication problems. How to make the computer understand what humans mean is a fundamental problem of the digital era and programming languages try to give one solution to it.

Most programming languages today are based on writing text. Text is wonderful: You can read this article and hopefully understand roughly what I'm talking about. (Although I'm not even talking at the moment!) If you don't understand what I write, you can write your own text with your questions and mail it to me. Text is quite easy for a computer to understand as well: It can be divided into single words or characters, that follow each other. Most of the time text is a one-dimensional medium: It connects words left to right (or right to left or top to bottom depending on local conventions).

But text doesn't have to be just left to right, it can be placed in two dimensions as well and still make sense. A spreadsheet is an example of text that is presented in a two-dimensional fashion.

To make a computer understand two-dimensional text, special rules for specifying the connections between words are necessary. In the tables of a spreadsheet, a rule could for example be, that words aligned in the same column have some sort of similar meaning.

Since the early days of the computer, scientists have looked for ways to make computers understand two-dimensionally arranged program text as well. A seminal document in this regard is Bert Sutherland's Ph.D. thesis `"The On-line Graphical Specification of Computer Procedures" [2] from 1966, where a lot of the concepts that are now common in graphical programming languages and dataflow environments like Pd are discussed at a very early stage.

Even though several computer scientists followed Sutherland's pioneering work, somehow graphical programming didn't really take off in the mainstream computer world. Besides the LABView [3] , a tool popular among industrial engineers, there is only one two-dimensional programming language that has found massive use - and that is the paradigm used in the "Max"-family of software which Pd is a member of.

Many of the visual programming languages employ a style of programming called "dataflow". The idea behind this is that changing the value of some variable in the system will make every object that somehow is connected to that variable recalculate its state. During that recalculation of course some other variables may change their values as well, forcing other objects to recalculate their state as well. In the end, changing one value at some place may start a whole torrent of calculations, and that's what gave the principle its name: the changes seem to flow through the system until everything is updated accordingly.

"Open Circuit", an installation transforming the dataflow principle into the physical world - made with Pd

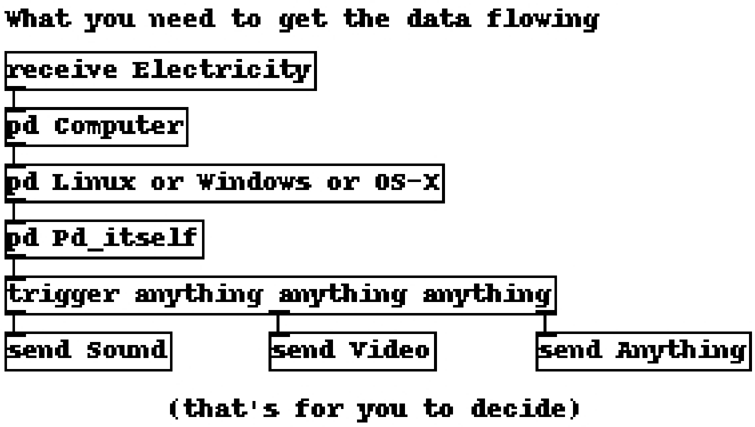
Again the spreadsheet is an example of this principle in action: Often a user is interested in the balance of a complete column of numbers in that spreadsheet, so the software provides a "SUM" function, which can watch many number fields at the same time and will update the sum every time a field changes its value. A modular software synthesizer like Pd constantly has to deliver new audio data to the soundcard, which has to react to new data immediately, for example as soon as a key on a connected keyboard is pressed or released.

While dataflow-inspired software often comes together with visual, two-dimensional programming, there also are pure text based programming languages that support dataflow ideas. Oz/Mozart [5] and Nova [6] are two examples.

Max and his family

Before Miller Puckette wrote Pd he was working at the IRCAM [4] institute in Paris and developed the first versions of Max, named after the "father of computer music" Max Mathews. Max became a very successful program and has many incarnations: Maybe the best known among these is Max/MSP, a commercial software produced by US-Company Cycling'74 [7] that could be yours for about 450 US-Dollar. With the Jitter extension by Cycling'74 Max/MSP also can handle video data and 3D-graphics.

The french IRCAM continues to offer its own version of Max, and IRCAM even made an offspring of it called jMax available as Open Source software some years ago. But jMax didn't enjoy widespread use, maybe because Pd had already filled the niche of a free version of Max. After IRCAM did some restructuring of its Open Source team, development of jMax practically came to a halt. Today jMax is of rather little importance, at least outside of IRCAM.


What you need to get the data flowing
receive Electricity
pd Computer
pd Linux or Windows or OS-X
pd Pd_itself
trigger anything anything anything
send Sound
send Video
send Anything
(that's for you to decide)

Pd is free software and can be used in an almost completely free environment

That leaves Pd as the most popular Max-like software that's available with source code as Free Software. While Max/MSP only runs on the commercial operating systems MS-Windows and Apple-OS, Pd additionally works on Linux and several other systems. (As Pd is developed mainly on Linux, things often even run a bit smoother there, though the other systems are very usable as well.) In the end Pd can make you completely independent from having to buy commercial, closed source software at all. And if you can get a free old PC somewhere the only thing you need to pay for to work with Pd is electricity.

Diving into Pure Data

Pd shares the basic workflow with the other programs in the Max family: Starting with a blank page, the user populates this empty canvas with little boxes and connects these boxes with patch cords, through which the boxes can exchange messages and other data. The end result is called a patch in the Max/Pd-lingo.

The object boxes take their functionality from the names they've been given. Pd doesn't use any nifty icons inside these boxes: Although it is a visual programming language, Pd at its heart is a software to work with (spatially arranged) text. Or as the developer of the Pd extension GridFlow [8] , Mathieu Bouchard, once put it: A word says more than a thousand pictures. And Pd knows this.

Pd comes with about 120 built-in object names (also called "classes"). The available objects can be extended either by installing some of the extra packages, that we will see soon, or by writing custom objects yourself. Binary objects for Pd are generally called "externals" while extensions written in the graphical Pd language itself are known as "abstractions".

Miller Vanilla: What the Pd core language can give you

The objects in the core version of Pd as distributed by Miller S. Puckette[1] himself on his website mainly deal with working on messages and numbers and with creating, analyzing and modifying sound. For a beginning user it is crucial to make oneself familiar with the core objects and how they interact. Although their number is relatively small, it is already possible to build powerful applications by using them in a smart and creative way. An advantage of restricting yourself to using core objects is that it's easy to share patches with others without forcing them to install special extensions.

Learning Pd is like a lot like learning a natural language: First one has to built a certain word pool and get familiar with the basic vocabulary of Pd. Then out of these "words" a user has to create "sentences", that is, some small patches and idioms, which instruct the software to fulfill certain tasks. As already mentioned, core Pd excels in making sound and in composing: With a bit of exercise it is possible to record sound, modify it in real-time, synthesize new sounds, apply the timbre of one sound to another and so on. The message and mathematical objects in Pd can be used to write algorithmic compositions that then are played back for example by external synthesizers or other software instruments - or of course in Pd itself.

The core Pd unfortunately doesn't handle video or 3D graphics itself (yet), so video artists will soon need to install one of the graphics extensions. The time invested in learning the core objects is not wasted however, as the graphics objects are connected with each other in the same fashion as the core objects.

Libraries: Important binary extensions

Some extensions for Pd are so useful, that practically every Pd user already has them installed. This includes a number of objects developed at the Institute Of Electronic Music And Acoustics [9] (IEM) in Graz, Austria, collected in libraries like zexy, iemlib, iemmatrix and others. These objects extend Pd by making certain message operations easier, they allow fast matrix calculations or provide excellent audio filters.

Cyclone is another important library, especially for users coming from Max/MSP: It clones many Max objects that are missing in the core Pd and even includes a useful, albeit a bit limited, importer for Max patches. The author of this text has used Cyclone extensively to port the approx 150 objects in Karlheinz Essl's Realtime Composition Library [10] for Max to Pd without having to run Max/MSP even once.

Externals are normally shipped in source code form from the Pd repository website at pure-data.sourceforge.net [12] , so they would need to be compiled before you could use them. While compiling is no big deal on Linux, users of other operating systems are more used to packages, that include binary objects, that they can run directly. The pd-extended package makes this very easy by bundling most of the available extensions with a version of Pd itself, that can be installed quickly.

Abstraction libraries

Besides binary extensions Pd can also load objects written with the Pd language itself. Many of these so called abstractions are collected into libraries as well and can be installed by just copying them into a directory in Pd's search path. Pd-extended includes most abstraction libraries, too, so if you install this, you're ready to roll.

One of the biggest collection of its kind is developed by Pd users in Montreal under the name pdmtl [11] . It includes many synthesizer objects, sequencers and lots of useful small tools. It's a nice touch that the whole pdmtl collection comes with excellent documentation. Every abstraction is explained both with a Pd help patch and on the pdmtl wikisite.

RTC-lib by Karlheinz Essl, now also available for Pd, was already mentioned: This library is especially interesting for composers, as it includes many modules for doing serial or 12-tone music and lots of objects for working with randomness.

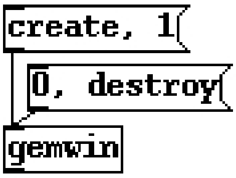
When developing Pd applications, one often has to transform a certain set of data into another realm, a process called "mapping". A library of abstraction with the same name makes these transformations easier. The "mapping" library is especially useful if you work with sensor data.

Abstractions do not only extend the functionality and vocabulary of Pd, they are also a good way to learn more about how Pd works, because it's possible to open an abstraction and look at the way it was built.

Visible Pd: Extensions for GFX and video

GEM

Several extensions to Pd allow working with video and graphics as well. The oldest among these is the "Graphics Environment for Multimedia" GEM [13] . It was first developed by Mark Danks, who later moved on to work in the game industry. IOhannes m zmoelnig (sic!) from Austria took over maintenance of GEM, but as with Pd, GEM is a group effort, where several developers are working as a team.



```
create, 1
  0, destroy
gemwin
```

The most basic GEM patch

GEM's main focus is 3D graphics: For this, GEM uses the OpenGL-library, an industry standard for dealing with 3D graphics (and of course 2D as well). GEM also handles video input with its "pix"-objects: It can read data from a camera or a file, transform and analyse the movies in various ways for example to do motion tracking and display them as animated textures on the screen.

Packets: PDP/PiDiP and pf

"Pure Data Packet" PDP [14] aims at image and video processing and is an ideal Pd extension for VJs. Some objects that were built on top of PDP are collected in the PiDiP [15] library, so to use PiDiP, you first need to install PDP. You can use PDP/PiDiP for real time video processing, blending, mixing or feedback constructs, do motion detection and even transform an image or movie directly into sound.

Tom Schouten, the author of PDP, is currently working on a related project that is going to supersede PDP in the end: PacketForth [16] or "pf". pf is a programming language itself that can be run as a standalone program, but also works inside of a special Pd object. When used as the latter, pf can be used to "clone" the objects of PDP, so that Pd patches written for PDP will continue to work with pf as well.

PDP in action: french artist Benjamin Cadon performing at make art festival 2007

GridFlow

A third popular library for video processing (and more) is GridFlow [8] . Mathieu Bouchard originally wrote it for IRCAM's jMax as well as Pd, but with the descend of jMax he concentrated on Pd and today more or less has dropped jMax support.

Like PDP/pf, GridFlow internally uses a full blown programming language, Ruby. This makes it possible to test and use GridFlow independently from Pd to some extent. GridFlow's objects are rather low level and to develop Pd applications with it, a mathematical background plus knowledge of image processing algorithms are useful if not necessary.

Talk to me: Connecting Pd to other software

Quite often it's necessary to connect Pd to other software, that may not even run on the same machine. To make different programs talk to each other, they have to agree on a certain language that everyone involved can understand. One such protocol comes with Pd itself, it's called FUDI and used in the [netsend] and [netreceive] objects in Pd. FUDI works by connecting two pieces of software with a network connection using the Internet Protocol (IP), and then lets them exchange simple lists of words or numbers that end with a semicolon. Because FUDI uses the Internet Protocol, which has powered the internet for many years, a lot of other programs can be connected to Pd using FUDI. Besides the ending semicolon FUDI is very free form. The players involved must somehow agree on the meaning of the messages sent on their own.

The Open Sound Control OSC [18] specification is simular to FUDI, but it's a bit stricter in regard to how messages need to look. Generally OSC messages have to start with a specifier that looks a bit like an URL or a filesystem path, for example "/trumpet/volume". After this specifier (the "OSC target") one or more values can follow, so that complete messages may look like: "/trumpet/volume 100" or "/player/tom go left". OSC messages are normally transported over IP-network connections as well, though the official specification also allows different kinds of connection. In recent years, OSC has become a standard that is supported by many programs, among these a lot of commercial, closed-source applications like NI Reaktor or Ableton Live. Pd doesn't include OSC as one of the core objects, instead special externals are used, that already are included if you install the Pd-extended package.

The oldest among the common protocols to make (music) software talk to each other is MIDI. Originally conceived to simplify and standardize the physical connections between hardware instruments, it's also useful to connect programs. Pd includes a complete selection of objects for receiving and sending MIDI data both from hardware and software MIDI instruments.

Shaking hands: Connecting Pd to Hardware

If you want to use external hardware with Pd, again MIDI may be the easiest way, provided you have a piece of hardware that has a MIDI plug, like MIDI keyboards or MIDI slider boxes. Some specialized objects were written to connect so called Human Interface Devices to Pd, that is for example game controllers like joysticks, dance mats or the Wiimote.

Pd also is a good choice if you want to build your own instruments or generally want to interface with custom-made electronic devices. Many of these, like the popular Arduino [17] controller board, connect to the computer through a serial connection. The [comport] object is the low-level building block to activate such hardware in Pd. For Arduino there even exists a special object [arduino] that greatly simplifies working with this board.

Advanced Development

If you're already a programmer, chances are high that you can continue to use your favourite programming language and create your own extensions by writing Pd objects in that language. Pd itself is written in C, so externals written in C or C++ are very common. To get you started, a good tutorial is available. Many other languages are wrapped in special Pd objects: Ruby can be used through GridFlow as already mentioned, Python is possible with the py/pyext objects by Thomas Grill. Other languages like Lua, Scheme, Haskell, Forth or Java can be used as well.

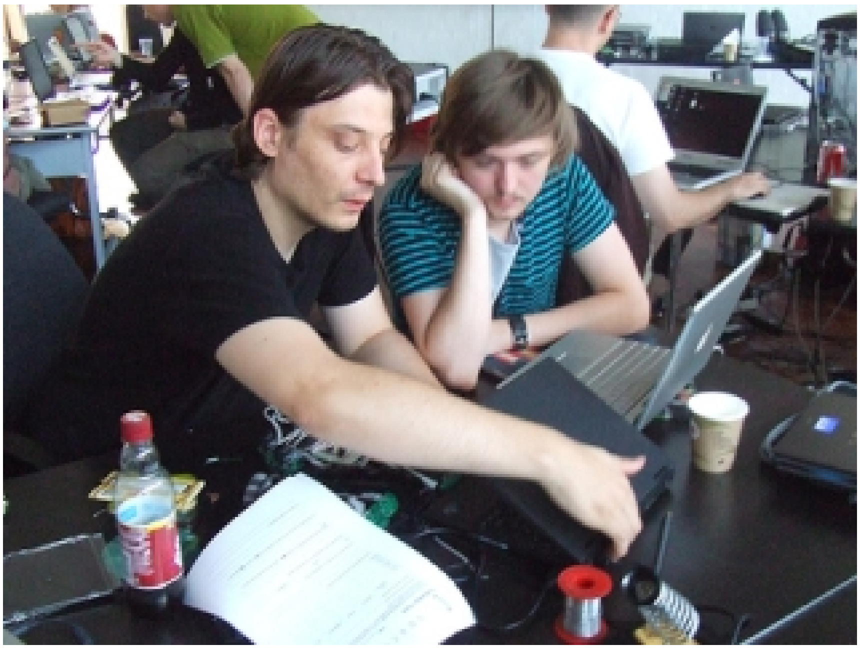
Pd siblings

Besides the version of Pd that Miller Puckette has continued to publish for more than a decade now, several related projects or spin-offs are in active development. They use the - freely available - source code of Pd as a base and built on it. DesireData [19] is one such project where a lot of improvements in the area of usability have been made. For example a lot of functionality can be accessed without having to use the mouse all the time, DesireData is heavily internationalized and generally looks much nicer than the original Pd. On the other hand some functionality is still missing at the time of writing, but DesireData definitly is a project to keep an eye on.

Vibrez [20] by Thomas Grill and others is a spin-off of Pd with many improvements regarding stability and usability. It's available for Windows and OS-X only. Vibrez is based on an older branch of Pd that introduced a reworked audio subsystem and allowed better performance with lower latencies especially on Windows. The work on this branch also kind of found its way into Nova [6] , a Pd-like system developed by Tim Blechmann, who also works on Vibrez. Nova itself doesn't have a graphical patcher system, instead it could be used as the underlying engine for such a system. In fact, at least one such patcher system already is in development.

The Pd community

Since Pd was introduced to the world in the middle of the 90s, it has found hundreds of users all over the world. Many of these communicate with each other using mailing lists, websites or the #dataflow [22] IRC channel. The puredata.info [21] website is a central hub to learn more about these activities and be informed about any news in the Pd scene. The IEM in Graz maintains the important mailing lists [23] : pd-list@iem.at is the list for general discussions on Pd and the first place to ask questions about Pd usage. Most questions are answered in less than a day. Developers of externals and the developers of Pd itself use the pd-dev@iem.at list to discuss and debate more technical issues. Both lists, especially pd-list, can be quite chatty and one should expect to get quite a lot of mails when subscribing. If you're only interested in important announcements, the pd-announce@iem.at list is a low-traffic list for exactly this.



The GOTO10 summer school

In the last years the number of workshops where Pd is taught is on the rise as well. Media centers and artist-run places quite often invite well-known Pd people to spread the knowledge. The programmer/artist collective GOTO10 [24] is famous for its Summer Schools (that sometimes happen in spring or autumn as well), that go on for one or two weeks covering Pd and related topics like physical computing or other synthesis and media software.

With freely available mail support, lots of tutorials online, affordable workshops everywhere, in the end there's no excuse to avoid learning at least a little bit of Pd. Promised: it will come in handy at some point.

Notes

[1] Miller S. Puckette: http://crca.ucsd.edu/~msp

[2]The On-line Graphical Specification of Computer Procedures: http://hdl.handle.net/1721.1/13474

[3] LABView: http://www.ni.com/labview

[4] IRCAM: http://www.ircam.fr

[5] Oz/Mozart: http://www.mozart-oz.org

[6] Nova: http://tim.klingt.org

[7] Cycling'74: http://cycling74.com

[8] GridFlow: http://gridflow.ca/

[9] IEM: http://www.iem.at

[10] Realtime Composition Library: http://www.essl.at/works/rtc.html

[11] pdmtl: http://wiki.dataflow.ws/PdMtlAbstractions

[12] pure-data.sourceforge.net: http://pure-data.sourceforge.net

[13] GEM: http://gem.iem.at

[14] PDP: http://zwizwa.goto10.org/index.php?page=PureDataPacket

[15] PiDiP: http://ydegoyon.free.fr/pidip.html

[16] PacketForth: http://zwizwa.goto10.org/index.php?page=PacketForth

[17] Arduino: http://www.arduino.cc

[18] OSC: http://opensoundcontrol.org/

[19] DesireData: http://artengine.ca/desiredata

[20] Vibrez: http://vibrez.net

[21] puredata.info: http://puredata.info#

[22] #dataflow: http://wiki.dataflow.ws/DataFlow

[23] mailing lists: http://www.puredata.info/community/lists

[24] GOTO10: http://goto10.org

Images

[1] Lightshow with Pd/GEM and physical modelling. Photo by Chun Lee

[2] "Open Circuit" is an installation transforming the dataflow principle into the

physical world - and it was made with Pd. Photo by Sebastien Vriet

[3] Pd is free software and can be used in an almost completely free environment. Photo by Frank Barknecht

[4] The most basic GEM patch. Photo by Frank Barknecht

[5] PDP in action: french artist Benjamin Cadon performing at make art festival 2007. Photo by Sebastien Vriet

[6] Pay attention, class! At the famous GOTO10 summer school. Photo by Chun Lee

" >

span-->

‹ Working with sound up Working with others ›