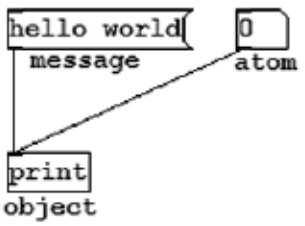

There are four types of text objects in Pd: message, atom, object, and comment.

Messages respond to mouse clicks by sending their contents to one or more destinations. The usual destination is the "outlet" at the lower left corner of the box.

Click the message box and watch the terminal window Pd was started in. You should see the "hello world" message appear.

Atoms respond to "Dragging" up and down with the mouse, by changing their contents and sending the result out their outlets. You can also type at an atom after clicking on it; hit "enter" to output the number or click anywhere else to cancel.

Objects, like "print" above, may have all sorts of functions depending on what's typed into them. The "print" object simply prints out every message it receives.

To get help on an object, right-click it. You should see a "help window" for the object.

updated for release 0.33

Does Pure Data Dream of Electric Violins?

Winfried Ritsch

Pd Introduction and Overview

Pd (Pure Data) is a graphic computer music language for real-time applications and was written by Miller S. Puckette. Pd can also be read and understood as *public domain*.

There has been a desire for *composition machines* ever since mathematics was used as the basis for composition (approx. 16th–17th century). Leibnitz and Marin Mersenne (1588–1646) used "combining" and "composing" as equivalent terms.

The *composition computer* was first developed in the time of *musique concrète* and serial music. One of its first applications was the use of random generators, which were controlled by paper tape readers.

Music Computer and Automatic Music

In order to create music by means of a computer, first a mathematical model representing the music must be found. The simplest representation is a list of discrete values (samples), the *audio signal*. This means that the entire information of the music is combined in one dimension and is plotted over time (in the time domain). Further representations are multidimensional vectors, such as the spectrogram, the sonogram in the frequency-time domain, or wavelet analysis in the wavelet domain. All these representations use an uninterrupted signal as a mathematical model.

However, as a person can certainly differentiate between different voices and events in music which traditionally correspond to the instruments and their notes, it is advantageous to split up the data according to this information. There are also parametric methods of music signal representation[9].

In order to gain even more control over the process of composition, it is useful to make a further division between control information and signal production. A closely defined interface between these parts is necessary there; the point of intersection is different from system to system. A division of "definition of signal generation" and "representation of music as control data" has resulted from this. One of the first forms of musical representation is notation, in which music is written down as events. These events are written down for the computer in a specially defined syntax, for example the "CSound score" or even in an existing programming language, for example "C" in "cmix".

Graphic notation is often used with control data[10]. In the representation of control data, two different basic models, the *sound continuum* and the *event model*, can be distinguished. With computer music languages not in real-time, where a distinction is made between the score (events) and sound synthesis (instrumentation), the time had to be noted.

Through the possibility of creating live music with the computer, the paradigm of computer music software changes from "composing" to "making music". The desire for live usage created a new generation of programs, real-time computer music programs, and thus created the new character of the computer musician. Real-time is the time that operations consume in the real world. Model time, however, means the run-time governed by the software itself. If the model time is synchronous to real-time, it is said that the system is capable of real-time. Early attempts of computer musicians were the use of analog computers with analog synthesizers; the analog sequencer above all enabled new complexities with the use of bucket brigade memory. Playing with points in time generated by machines was the latest, fascinating thing.

Pd was developed by Miller Puckette, based on existing computer music languages. Pd as a graphic programming language is historically based on the program Patcher[11] and thus on the use of MIDI[12] as data material. The event model was chosen with the MIDI control data[13]. An event possesses not only data information about its content but also the point in time of its occurrence as unnoted data information. Making music with computers that react to events and not only can handle but also save, alter, and repeat them in real-time led to the use of various sensors and input devices which represent the musician-machine interface. While the singer still produces the sound physically, the pianist presses keys and acts as the catalyst of sounds by pressing keys commands are given with sensors. Music making thereby changes – the creator of sound becomes an *operator,* and eventually a *commander*. The concept of interactive systems or interactive art suggests that the computer becomes an equal partner in creating music. That this is only a reaction because of the lack of intelligence of the machine and *reactive systems* are created is in large part ignored or intentionally juggled with these terms.

This development should not be seen as independent from new thinking in art, especially media art, in which the path from "Mythos zum Prozessdenken" (myth to process thinking)[16] has already been implemented. This further step of abstraction has brought the perspective of music as process. It follows that music is written as the definition of process instructions and algorithmic music is represented in an almost pure form. Therefore, the notation in "graphic object oriented form" can be used and a composition represented as a data stream diagram. The use of this paradigm happened above all in automated sound installations and resulted in "endlessly" long compositions[14]. The view of the machine as a principle for modeling calls for the use of data stream models instead of program stream models as applied in other programming languages. Data stream models have graph theory as a basis[15].

However, it is new that in Pd an improvisation can happen with processes, which means that processes and algorithms can be discovered, changed, and rejected in real-time, which represents a machine which is altered during its use. I prefer to leave the answer to the question to what extent this leads to a new movement in art to the fine arts theorists.

The Pd graph

In graph theory, a graph is understood to be a multitude of points which run between the lines. The points are called nodes or vertices, the lines are called edges. They are marked with arrows in drawn graphs. The origins of graph theory go back to 1736 and to Leonhard Euler, who used it to solve the Königsberg bridge problem.[17]

In Pd data flow, a graph is a collection of drawn operations that are represented visually through boxes or other graphic elements, connected by lines (which should actually be arrows). There are operations without input that serve as sources for data without output and which can be considered as drains or ports to other systems. Operations that stand alone mainly serve as definitions. In Pd graphs, however, two independent levels of data flow are represented, one signal level (signal dataflow) and one event level (message system), which strictly speaking should be handled independently. The message system could thus be referred to as asynchronous data flow and a generalization of the synchronous data flow.

The scheduler-dispatcher mechanism is responsible for the allocation of data and thus the data stream. The order in which data is allocated does not always come directly from the graphic representations. Therefore the scheduler puts the sequence of operations in order and the dispatcher executes them. In Pd, a special scheduler is used which, adapted to the system during the run-time, intervenes in the graph and compiles it anew again and again. This was an essential step toward real-time programming with Pd and toward programming or composing becoming part of a performance.

As von Neumann architecture, the computer itself does not possess the characteristics of a data stream system, but it must from the outset be limited to offer its services as such. In con-

trast to programs in which the function calls are worked through from the program memory and are applied to data, this data stream paradigm makes a machine out of a computer, in contrast to programs which execute function calls taken from the program memory and aplies them to data, which lets data flow continuously and serves as a tool for building machines.

Pd thus enables a *reactive system* in real-time. It continuously observes the surrounding system with its interfaces and reacts to its information (actions) in real-time in contrast to a transformational system, which only reacts with the surrounding system at designated points of the program flow. Pd is a typical definition of robotic systems.

As data flow diagrams are a graphic notation, it seemed reasonable to realize this kind of programming language with graphic representations, so-called *graphic programming* or *visual programming*. This also resembles the patching of early analog synthesizers, in which signals are conducted through devices with cables.

Visual Programming

Visual programming or visual patching indicates programming by means of graphic illustrations and has its sources in the graphic notation of DSP algorithms in textbooks and analog devices such as synthesizers or analog computers. It is thus a natural way to represent and design algorithms for signal processors and signal streams. The first text-based computer music languages already use these graphics.

The first graphic programming language was probably written in 1966 by William Robert Sutherland. However, the most important programs of this kind were developed in the 1980s. Here is a definition from Meyers:

> Visual Programming (VP) refers to any system that allows the user to specify a program in two-(or more)-dimensional fashion. […] conventional textual languages are not considered two dimensional since the compilers or interpreters process them as long, one-dimensional streams.[21]

Since Pd works in a graph oriented manner, it is suitable for visual programming and also imitates patching from early analog synthesizers, where signals are conducted through devices by cables.

I can only speculate on the artistic implications of graphic programming; however, it is always mentioned as the basis of a new generation of software. The deciding factor is the possibility for quick prototyping and operations to the detriment of structured programs. Concepts of local variables and namespaces were only added later with tricks.

A further reason for quick prototyping is that it concerns a kind of interpreter. An interpreter (in the classical meaning of software technology) is a software program that, unlike assemblers or compilers, does not convert a program's source code into a file directly executable in the

system but rather reads in, analyzes, and executes the source code. As the von Neumann architecture is bypassed, there is no source code in the classical meaning of the term. If the call list of objects in the signal flow plan is compiled anew again and again when a graph is altered, only new paths of transmission of data are arranged in the message system. These criteria can hardly be applied, yet the behavior of the system corresponds to an interpreter because the input becomes effective immediately; it is a kind of scripting language but not perceived as such.

The High Art of Dynamic Patching and Scripting Pd

The dynamic constructing of patches arose from the need to construct Pd programs which adapt to the requirements of an application, for example the number of voices of a synthesizer, in order to duplicate parallel graphs. This also permits drawing up algorithmic graphs and for many is part of the high art of patching, but is regarded skeptically by many as the system is not really prepared for this since the referencing of connections and object instances does not recognize symbolic names. The vision of the machine that replicates itself is supported by this, which corresponds to the second meaning of the von Neumann probe. Von Neumann referred to these as *universal constructors*.

The integration of script languages was introduced only later as an expansion and comes from the need to use traditional interpreter languages. They serve as more complex data processing, as interfaces to system resources but also to construct Pd patches dynamically. Pd can thus "patch" itself. The weak point of signal stream programs should be overcome and a universal software environment created in which not only other computer music languages can be executed but also server functions for or the operation of external resources – the Internet, for example.

The difficulty here, among other things, is that scripts only seldom have constant execution times and thus jeopardize the real-time concept of Pd. To circumvent this, the interpreter languages – python, for example – are executed in a parallel process, which in turn counteracts the original approach to possess only one function graph (singlethread). These new demands on Pd show the historical boundaries and on the other hand the expandability and the adaptability of Pd.

Pd Networks as a New Concept for Bands

If several musicians play in a band, it is advantageous for them to communicate with each other by listening and reacting, thus by interaction. If several Pd musicians or their Pd programs as instruments play together, they can avail themselves in addition of the communication over the network. The [netsend] and [netreceive] objects were implemented very early in Pd. This permits not only the cluster formation of computers in order to distribute processor

load but also playful networking and thereby making music with others. Exchanging time information, such as synchronization signals, is a traditional method for this; however, to allow the others to play Pd as their instrument first became well manageable through these systems. It is possible just the same to send patches over the network and to let them run on other Pd instances, thus the exchange of algorithms in addition to data.

The implication of this means the possibility to exchange Pd patches over the network often over great distances in real-time and thus realizes the vision of the music jam on the Internet.

Appendix

Pd Implementation

In principle, Pd is Open Source and Free Software according to the Standard Improved BSD License. This means that Pd can be used everywhere and also be sold with a product.

The official code basis is administered by Miller S. Puckette, who, similarly to the Linux kernel, incorporates certain changes from the Pd community and improves the faulty code. In this way, he guarantees the stable functioning of Pd. Because greater and greater demands were made on Pd, an initiative came from the developer community which checked in the original source code of a release in source forge in a CVS system. Individual enhancements can be implemented from here out. This CVS version has several branches which follow different needs. Enlargements of Pd are for the most part developed separately as external libraries and follow their own numbering of versions. Only a few have turned out to be a stable development. However, these libraries have their own licenses and can thus be used differently.

In principle, each Pd patch is the same at several platforms. The libraries are executed differently depending on the operating system, meaning that some are independent of the operating system and some are only for use with certain operating systems.

Computer Music Languages

Only later could the computer be used for sound generation, for which special computer music languages were developed. One of the pioneers was Max Mathews, who constructed the first purely synthetically produced computer music in 1957 with an IBM 704 computer and the program "Music I". Max Mathews' readiness to pass on "Music IV" to the universities of Stanford and Princeton led to many new computer music languages (dialects), culminating in Barry Vercoe's CSound.

As a further development, at IRCAM[1], Miller Puckette developed the visual programming language MAX[3], which is interpreted in real-time by means of the DSP program FTS[5]. It is based on Mathews' idea of a flexible sound synthesis system able to be set up by the user.

The digital signal processing (DSP) took place on an individual processor card hosted by a NEXT computer, the ISPW[4]. Afterwards, this system was ported to SGIs, where the main processor was used for the DSP. As a commercial derivative without signal processing and specialized in MIDI control, the enhancement for MacIntosh computers was sold to the company Opcode, which distributed Opcode MAX[6] further.

After Miller Puckette left IRCAM and was appointed to the University of San Diego, he decided to rewrite MAX/fts with Pd as Open Source/Free Software.

David Zicarelli later added the DSP engine MSP[7] to Opcode MAX. At IRCAM, Miller Puckette's MAX/FTS was transferred into jmax and enhanced, a MAX with Java user interface which was later published as Open Source. Open Sound World (OSW) represents a further Open Source development of visual programming languages for music[19].

Miller Puckette at the first Pd convention in Graz

References and Notes

1 Institut de Recherche et Coordination Acoustique/Musique (Institute for Music/Acoustic Research and Coordination) in Paris

2 Mr. Mathews directed the Acoustical and Behavioral Research Center at Bell Laboratories from 1962 to 1985 and developed the Music I Language. He was Scientific Advisor at the Institut de Recherche et Coordination Acoustique/Musique (IRCAM)

3 See <http://mitpress.mit.edu/e-books/csound/fpage/pub/csbook/contents/foreword.html>, Music 11. In 1980, student Miller Puckette connected a light-sensing diode to one end of the PDP-11 and an array-processing accelerator to the other, enabling one-dimensional conducting of a real-time performance.

4 IRCAM Signal Processing Workstation

5 *faster than sound*, a DSP rendering, originally the operating system of the ISPW card, later a daemon, which is executed parallel to the graphic user interface and is controlled over TCP/IP.

6 It was sold to the company Opcode, taken over by the company cycling74.

7 Max Sound Processing (or Miller Smith Puckette ;-)

8 <http://osw.sourceforge.net/>, OSW was begun as research project by Amar Chaudhary, Adrian Freed and Matthew Wright at the Ceneter for New Music and Audio Technologies (CNMAT) at the University of California at Berkeley. It is currently maintained as an open source software project lead by Amar Chaudhary (first public Release 2001).

9 See p.3, "Representation of Musical Signals", edited by Giovanni De Poli, Aldo Piccialli, Curtis Roads, MIT Press, 1991

10 e.g.: in analog, among other things, used by Stockhausen in his first electronic works.

11 Macintosh program by Miller Puckette. See "The Patcher", Proceedings, ICMC 1988 (Cologne).

12 musical instrument device interface

13 This can be transferred to a continuum model by means of Metros.

14 The works of Alvin Lucier can be regarded as an early example of pure process thinking as composition, especially the piece "I am sitting in a room", in which he makes the room vibrate with a room microphone and the repeated recording of feedback with one spoken sentence as source material.

15 Computer scientists created a basis for this. See "Software Synthesis from Dataflow Graphs", Shuvra S. Bhattacharyya, Praveen K. Murthy, Edward A. Lee, Kluwer Academic Publishers, Norwell Massachusetts, 1996.

16 Christa Lichtenstern, *Vom Mythos zum Prozessdenken. Ovid-Rezeption – Surrealistische Ästhetik – Verwandlungsthematik der Nachkriegskunst*, Weinheim (Acta humaniora) 1992.

17 As a concrete example, this refers to the city of Königsberg and the question of whether there is a path through the city which crosses over each of the seven bridges over the Pregel just once to return to the starting point. Euler proved that no such path exists.

18 "The on-line graphical specification of computer procedures." William Robert Sutherland, 1966. See <http://theses.mit.edu/Dienst/UI/2.0/Describe/0018.mit.theses%2f1966-24?abstract=>

19 See <http://osw.sourceforge.net/oswfaq.php#1.2>

20 Eberhardt Knobloch in *Maß, Zahl und Gewicht*, Herzog August Bibliothek, 1989 pp.249-264

21 "Taxonomies of Visual Programming and Program Visualization", B. A. Myers, 1990, journal jvlc 1, pp. 97-123, volume 1