This book is based on the First International Pd-Convention 2004, Graz/Austria, a cooperation of:

institut für elektronische musik und akustik

pd-graz

bang Pure Data

wolke

First Edition 2006

Coordination: Fränk Zimmer
Translations: Aileen Derieg and Maureen Levis
Lectors: Aileen Derieg, Florian Hollerweger, IOhannes m zmölnig
Cover design: pd-graz
Photographs: Uwe Vollmann and Irmgard Jäger
Photo selection: Reni Hofmüller
Typesetting: michon, Hofheim
Printing: Fuldaer Verlagsanstalt

pd-graz is:
Lukas Gruber, Reni Hofmüller, Florian Hollerweger, Georg Holzmann,
Karin Koschell, Thomas Musil, Markus Noisternig, Renate Oblak,
Michael Pinter, Peter Plessas, Nicole Pruckermayr, Winfried Ritsch,
Romana Rust, Uwe Vollmann, Franz Xaver, Ales Zemene,
Fränk Zimmer, IOhannes m zmölnig.

http://pd-graz.mur.at

Sponsored by:

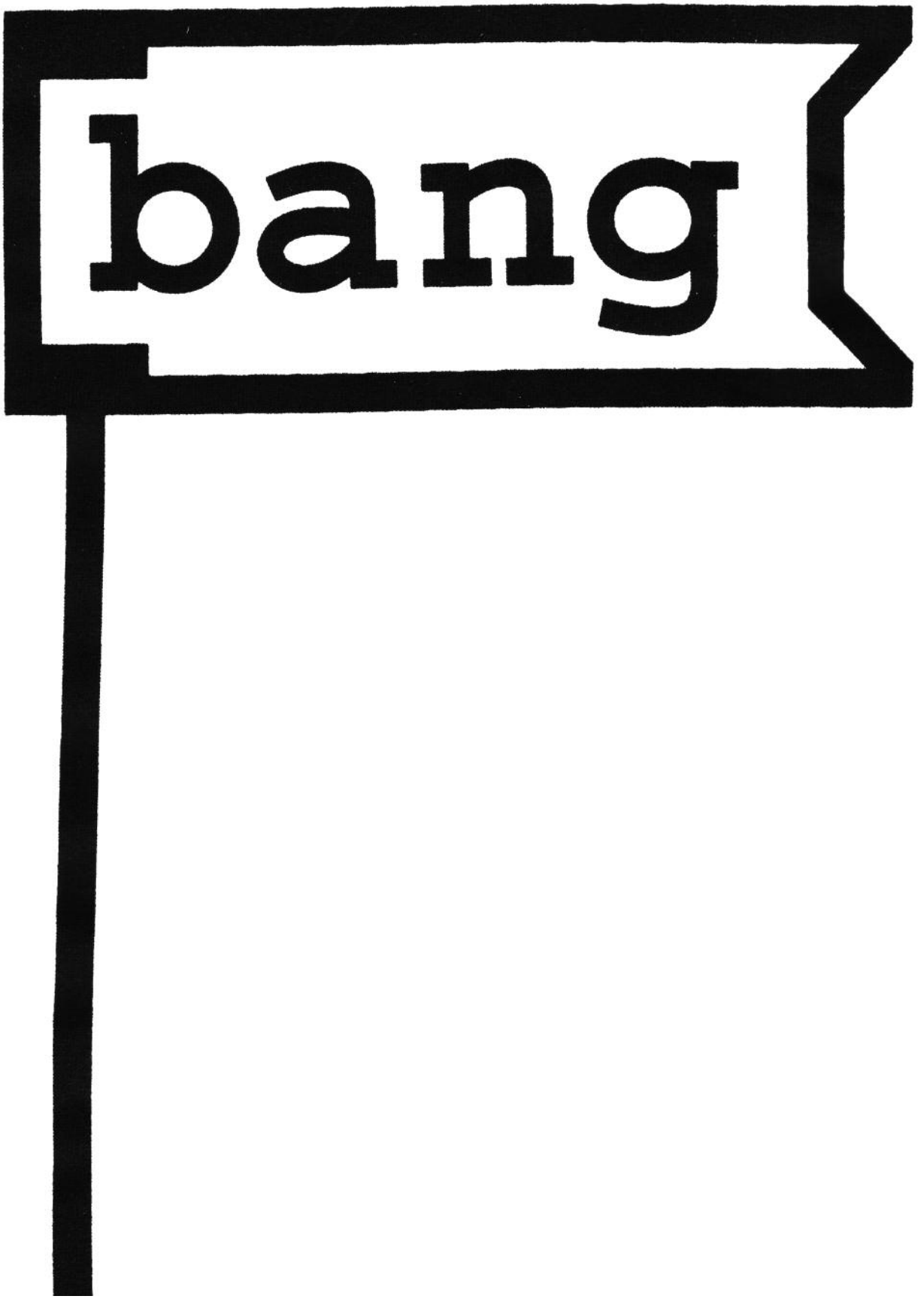
BUNDESKANZLERAMT KUNST

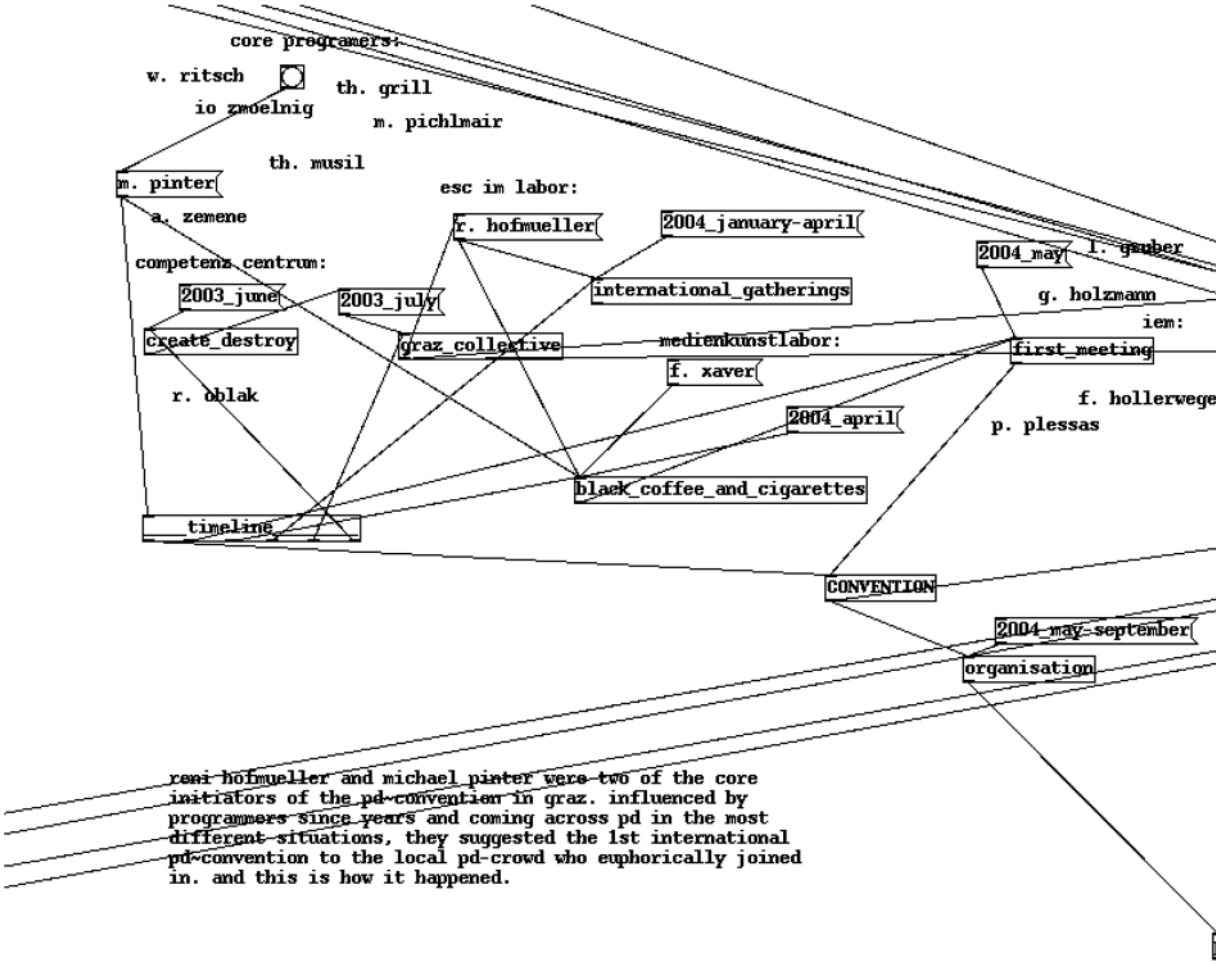
Das Land Steiermark
Kultur

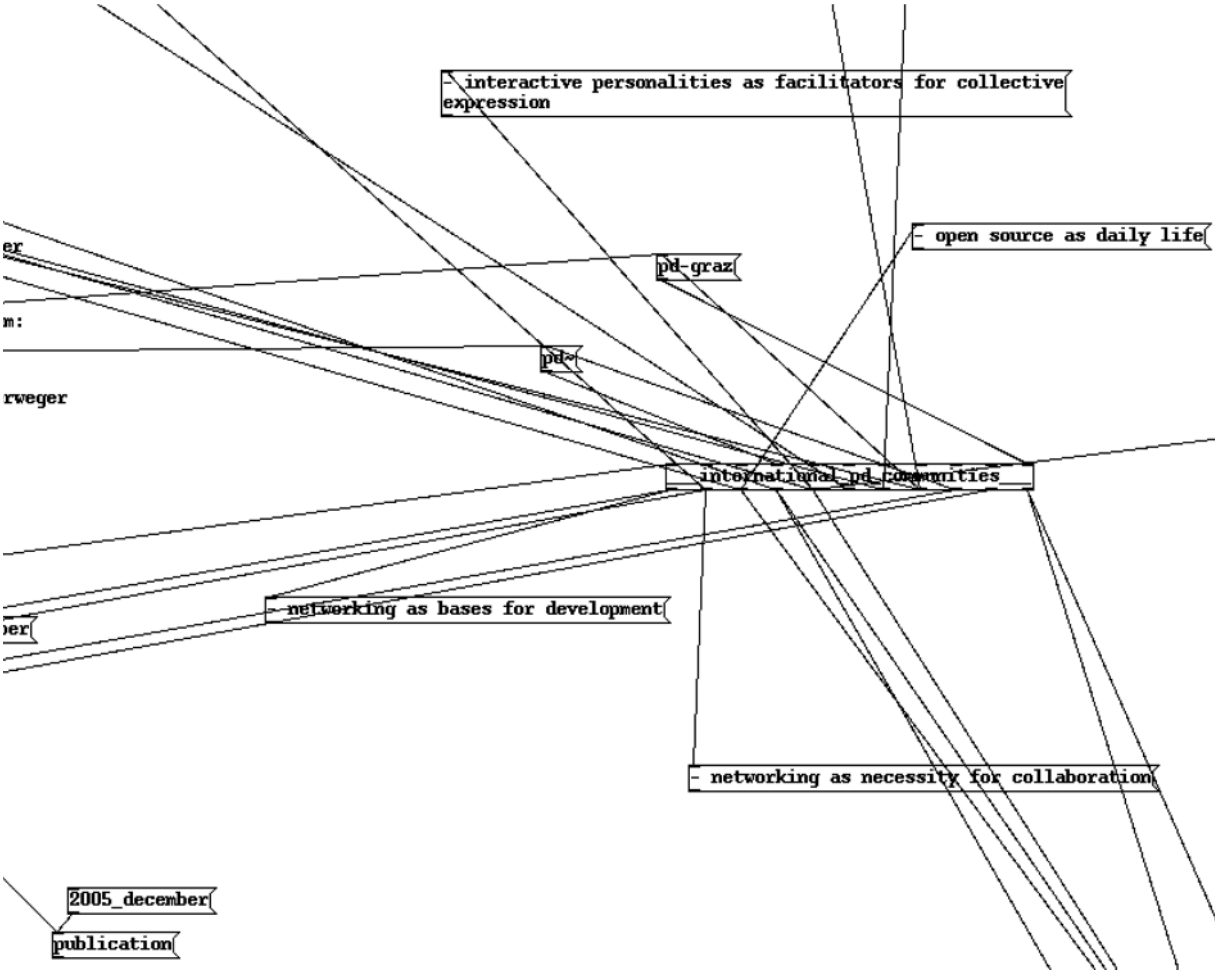
Stadt GRAZ Kultur

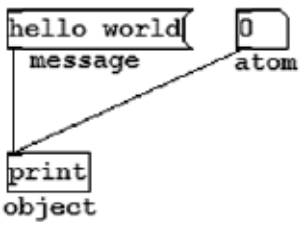
ISBN-10: 3-936000-37-9
ISBN-13: 978-3-936000-37-5

Content


bang


core programers:
w. ritsch
th. grill
io zmoelnig
m. pichlmair
th. musil
m. pinter
esc im labor:
a. zemene
r. hofmueller
2004_january-april
competens centrum:
2004_may
l. gruber
2003_june
2003_july
international_gatherings
g. holzmann
iem:
create_destroy
graz_collective
medienkunstlabor:
first_meeting
f. xaver
r. oblak
f. hollerwege
2004_april
p. plessas
black_coffee_and_cigarettes
timeline
CONVENTION
2004_may-september
organisation
reni hofmueller and michael pinter were two of the core initiators of the pd-convention in graz. influenced by programmers since years and coming across pd in the most different situations, they suggested the 1st international pd-convention to the local pd-crowd who euphorically joined in. and this is how it happened.


- interactive personalities as facilitators for collective expression
- open source as daily life
pd-graz
pd~
international_pd_communities
- networking as bases for development
- networking as necessity for collaboration
2005_december
publication



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.

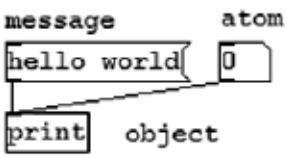
17 As a concrete example, this refers to the city of Königsberg and the question of whether there is a path through the city which crosses over each of the seven bridges over the Pregel just once to return to the starting point. Euler proved that no such path exists.

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



When you first open a Pd document like this one, your cursor will be an arrow. Select "edit mode" in the Edit menu and the cursor will change to the image of a hand. The patch is now in edit mode. You can move any object by dragging it.

Select "Edit mode" again in the Edit menu and you're back to the arrow cursor which acts on objects without moving them.

In Edit mode, if you click on a message, object, or comment, you can then retype the text. For objects this will create a new object and delete the old one. Pd will try to reconnect the newly created object in the same way as the old one.

When you're done changing the contents of the box, click outside the box to deselect it. This tells Pd to incorporate the new text.

You can create new objects by duplicating existing ones using the "duplicate" menu item. You can also "cut" and "paste" them. If you duplicate several connected objects the connections will be replicated too.

Edit mode also lets you make and break connections between objects. Put the "hand" cursor over a line connecting two objects: it turns into an X. Clicking will delete the connection. Hold the cursor over an outlet and it becomes a circle (a patch point). Drag to any box and release; you will be connected to the nearest inlet.

The "put" menu creates new text items of any of the four types. You can also put a "symbol" box, analogous to a number box but for showing and entering text strings.

updated for Pd version 0.33

Media Environments as Cultural Practices: Open Source Communities, Art and Computer Games

Reinhard Braun

"Media and media techniques are not simply given mediators or cultural techniques to be taken for granted, but nor are they brilliant or obscure inventions. They are systematically developed formations of persons, artifacts, operating instructions and spaces of possibility, which are positioned at very specific discourse points and produced in complex circumstances of cultural exchange. They indicate the extent to which media and technology are culturally coded, the degree to which every technical or technology-based development is tied to processes of its discursivization and culturalization."[1]

Developments in the area of the collective production of software can thus be described, beyond their possibly subversive or radical position within the framework of techno-economic discourses, primarily as a formation of the interlocking of technology, economy, and sociocultural fields. Under the presupposition that not only technological practices are involved, these developments can also be regarded as a form of culturalizing technology as a whole, a form of culturalization that possibly reveals conjunctions with completely different discourses that are not necessarily recognizable as articulations of interlinking practices [Stuart Hall]. These discourses include specific artistic positions on the one hand, and on the other a formation from the entertainment industry, namely computer games.

What the process of a – possibly – new type of production environment for technical developments involves is always also the establishment of new technologies; this process can also be described as the attempt by social actors to specifically "domesticate" technology, i.e. through appropriation and revision. It is neither coincidental nor unintentional that we touch here upon debates from cultural studies. Using numerous examples from mass media, cultural studies have shown that consumption, which in our case is the use and application of media for specific purposes, i.e., as Vilém Flusser would say, the production of improbable and thus, in the narrow sense, informative software applications, is hardly ever to be imagined as a passive operation of being subjected to power or ideology, but is instead, in almost every case, to be regarded as a form of appropriation, re-interpretation and recontextualization. In this sense, we are not in danger of speaking of a media usage that makes media look like tools that can be arbitrarily switched on or put aside. Television would not cease to be a mass medium, if all the viewers were to turn their TV sets off. "Domestication" is hence not intended to signify a trivialization with regard to apparatuses; instead it is much more a matter of the processes of integration in social and cultural operations. It is through these processes – which are simultaneously microsocial and global – of a functional adaptation of media-technical operations

---

1 Dirk Spreen, "Die Diskursstelle der Medien", in: Andreas Lösch et al. (Ed.), *Technologien als Diskurse. Konstruktion von Wissen, Medien und Körpern*, Heidelberg: Synchron Verlag 2001, pp. 21-40, p. 37.

that technology first becomes a form of cultural technique, in other words a social technique that includes a mechanism of identity at the same time. This circumstance has been characterized for television with the term "couch potato"; now it must be reformulated with regard to the way social subjects are permanently hooked up with the mobile telephone and/or MP3-player – yet in light of the increasing number of wireless-LAN hotspots, there is hardly anything left that could become a "potato". Nevertheless, it would be inadequate to describe this development solely in terms of optimization, miniaturization (hence as a new teleology of technology): instead, the "disappearance" of the apparatuses indicates specifically the situation that technology is to be understood more as a practice than as a formation of apparatuses. Thus it seems far more interesting to observe a phenomenon such as Open Source from this perspective, which queries it in its conjunction with other media practices, thus leaving the stage of discussions of code, licensing procedures or even something like software art.

In comparison, for instance, with the emphasis of Friedrich Kittler and his followers on the materiality of the discourses, i.e. the invocation of the subject inscribed in the circuitry by technology, so to speak, turning in this way to a quasi cross-platform questioning of the adaptation of a certain practice coupled with the technologies stresses specifically the role of these subjects as actors, emphasizing their capacity for taking action even in light of a world of apparatuses and media firmly sealed by micro- and nano technologies: indeed there are black boxes everywhere you look. Yet it must at least be presumed that the social has always preceded the technological, that there must have always been a place of discourse for the technological in the social already, before it was able to develop its materiality of the technological discourse. "It is not a matter of our adaptation to a new technological environment, but rather of the insight that this technology is our adaptation already."[2] However, this idea of technology as a permanent process of reworking the environment does not subordinate us to the electronic circuits, but instead views the conception of these circuits as a form of cultural practice that transgresses the technological. "The idea of mobilizing and reproducing the whole of society by means of media strategies, the notion of the social as a functional relational arrangement and the exploration of the relationship of individual bodies and mediation through the media already existed in the early 19th century. Thus these problematizations cannot be merely an effect of technical developments. The media become a problem, before the technical structures and apparative arrangements appear, in reference to which this problem is discussed today. There is a place of discourse in media that is interwoven with political and economic discourses, which is not the consequence of new technologies, but which precedes their emergence. […] Modernity's understanding of media is first of all a discourse."[3]

From this perspective it must be assumed that new media technologies must first be inserted in cultural discourse points that have always already been there, in order to be socially and culturally relevant. The appearance of new media is thus always accompanied by discourses and

2 Steven Shapiro, *Doom Patrols. Streifzüge durch die Postmoderne*, Bollmann: Mannheim 1997, p. 98.

3 Dirk Spreen, op. cit.

practices that assign it a cultural location in society. It is first through this insertion into discourse points, a process of implementation that makes it possible to become socially and culturally relevant, that media technologies emerge as cultural techniques that shape and remodel the idea and the form of communication, knowledge, experience, etc. Starting first from this cultural location that is imaginable in the interlocking of subject, society and the technological without having to ascribe a primacy to one of these "components", the technical modes of social relationships arise, from the telephone to television all the way to chatrooms or MMORPGs – and computer games/video games/online games especially exemplify these kinds of technical modes of social relationships, once one is prepared to move beyond violence debates and look more closely at the exact parameters and coordinates of these very specific media practices.

The historical example of the telephone may be used to exemplify a reconstruction of this process of the interlocking of culture and technology. The interplay between the way the body is mechanized and functionalized and the way the technology works, which this idea produces at the same time as it is realized by it, cannot be subsequently separated from one another. In order to successfully realize the idea of a technology related to the body, the body must first be regarded as something accessible to technology, something that can be divided up into components, so to speak, which can be subjected to technical appropriation and recording; the question that arises is whether technology subordinates or extends the body, as McLuhan suggested. We expand on this idea of extension by imagining the body as a collective body and technology as a form of mediation between and within this collective. Even the notion of a mechanized or electronified body by itself, a body that is accessible to the structural analysis of technological processes, is not capable of explaining the emergence of techniques, which aim to socially interconnect these bodies (such as the telephone), which are always more than mere physiological, mechanical, electrical phenomena: namely cultural subjects, social individuals, collective identities. This is the point where the telephone comes in as a cultural technique: not in the translation of sound waves into electronic impulses, but in the technological coupling of subjects as communicative, in the idea of a society whose complexity, speed vectors and production conditions make it necessary to establish a technology as a cultural technique, to save the societal, so to speak, to maintain the idea of a collective social body, which thus increasingly becomes a collective media body, as Christina von Braun called it, but one which can be set for permanence solely through this implementation of a technological operator.

And – to take a daring step into the present – a paradigm that stabilizes the bodies in their sovereignty with respect to the medium has become established in the same way in the area of multiplayer games. The challenge does not necessarily consist in playing against the AI of the game, but in playing against the other co-players qua interface, which is naturally to be understood as a technological interface: as the example of the telephone was intended to illustrate, new technologies emerge in interplay with a technologization of the body/subject, although this does not yet imply its subjugation. Cultural technologies emerge, however, at a different interface with society: not in the laboratory of the genetic engineers, to put it metaphorically, but in the practices of the plastic surgeons, not in Edison's laboratory but in the distribution offices of Activision and Electronic Arts.

"With their institutionalized technical and symbolic arrangements, writing, printing, telegraphy and television make certain forms of communication and perception obligatory and thus create unequivocal preconditions for politics and commerce, for mentality and subjectivity. Media are called 'cultural technologies' in this sense: they create and delimit the space of possibility of cultural forms."[4] Yet the places of discourse, which are at the beginning of the space of possibility, do not mean mediasupported communication, virtual reality or similar phantasms, in other words the frequently invoked new modes of interconnecting subject / body and environment that turns them into circuitry moments in the network of Internet3, but rather the utopias of control and steering, rooted in the idea of a potential technicity of the body. These are phantasms of cultural conditions of domination that culminated in modernism, yet reach far back to the beginnings of the Enlightenment – the idea of the subjugation, the appropriation, the recording and systematization of nature, and also the idea of a linear temporal development as the foundation for historicity. These are the coordinates that prepare the field of the cultural for the implementation of technologies, which undoubtedly always also represent technologies of power. Within the framework of this preparation it first becomes possible to more precisely specify terms like communication, collaboration, participation, but also mediality or technicity. Especially the concept of communication that can be found running from the telephone to the online chat like a red thread of technology-supported socio-cultural interaction then proves to be a form of maintaining the social under media conditions. For no state can be imagined, in which the subject could be described solely with a machine: there are always all the other subjects there with their machines as well, turning this situation into a collectivized practice in light of the machine, but also in light of the social as a whole. As the action of an isolated and thus medially individuated subject, this disposition would be simply meaningless.

If we talk about a technicity of culture, then we must also talk about about a culturalization of the technical. However, this culturalization does not only reveal itself in an exemplary way in the form of Open Source communities, i.e. in exhausting productivity within the framework of collaborative, not primarily economic technical development, in the area of highly technically supported developer communities. Instead it appears much earlier, as indicated above, in the area of artistic media practices and, at the other end of the scale of possible media practices, in the area of computer games.

In a project such as "The World in 24 Hours", organized by Robert Adrian X at Ars Electronica 1982, technology-supported communication, the "electronic space" as a space of action and interaction, is taken over to investigate new aspects and dimensions of the exchange relationships of increasingly complex societies using machines. Since media claim to mediate between individuals, groups and interests as an interface and to regulate them, to a certain extent they replace functions of a lost public sphere. Media-immanent "art" work consistently transforms itself through the stage of systemimmanence into an ultimately social immanence, if it does

[4] Markus Stauff, "Medientechnologien in Auflösung: Dispositive und diskursive Mechanismen von Fernsehen", in: *Technologien als Diskurse*, op. cit., p. 81-100, p. 83.

not merely occupy communication channels, but regards them as a new type of cultural space, in which differences, conflicts, contradictions and various models of representation are negotiated. The question of art is thus turned around into a question of cultural hierarchies and orders, possibly supplying, negotiating or rejecting proposals for cultural concepts. Another project in which Robert Adrian X was centrally involved was ARTEX (Artists' Electronic Exchange Program), which was implemented in 1980/81 for the commercial network "I.P. Sharp APL Network", existed until 1990, and was one of the first mail programs in the world regularly used by artists. ARTEX thus enabled the use of this new production form of distributed authorship, which was simultaneously a new form of communicatively oriented cooperation, as a permanent experimental "space" in the framework of artistic practices for the first time.

"Organizing worldwide communication projects with the help of airmail and telephone proved to be increasingly problematic, so in the summer of 1980 Bill Bartlett and I started putting pressure on IPSA to convince this company to develop a cheap and user-friendly email program for users that don't belong to any company or institution, but operate as individuals from their studios."[5] The result was ARTBOX, developed by Gottfried Bach, a simple and economical version of the IPSA "Mailbox". The ARTBOX underwent a series of changes, until it was defined in 1983 as ARTEX: the "Artists' Electronic Exchange Program" – a "user group" in the IPSA network. "FidoNet", developed by Tom Jennings, also reached its first peak around 1985. As Jennings wrote about it: "A computer bulletin board (BBS) is in fact a collection of social conventions encoded in software, each a microscopic 'internet' of dozens of hundreds of people, hundreds of downloadable files. In fact a lot of internet terminology ('download') in fact are BBS paradigms and words."[6] Apart from the fact that Jennings here refers to an important genealogy of the Internet that is rarely mentioned in the conventional representation of its military origins, the perspective of software as a form of coding social conventions seems especially noteworthy in the present context. Media techniques do not become established because they have become technically possible, but because cultural practices make them appear necessary.

Yet is it not only artistic media practices that can be described as this form of culturalizing technology, the same could also be said for computer games (apart from the fact that they also share a common history with Open Source developments: the title "Doom", introduced in 1993, was released in 1999 under a GNU GP License; "Quake", also put on the market by id-Software in 1996, was the starting point for the "mod" scene, i.e. game modifications written by users and made available to the community for downloading – just to mention two of the most well known games).

---

5 Robert Adrian X, "Kunst und Telekommunikation 1979–1986: Die Pionierzeit", in *springerin* 1/1995, p. 10-11, p. 10.

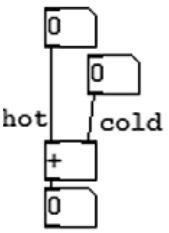
6 Paul Jennings, cf. Reinhard Braun, "One has to slide off into some other territory", in Kunsthalle Wien (Ed.), Robert Adrian X, Folio: Vienna 2001, p. 90-122, p. 112.

Computer games undoubtedly engage in the emergence of social competencies; they offer new forms of spaces of action and ideas of reality – yet they also promote a perception of reality that makes it appear largely formable and without consequences. Computer games have the potential to create new forms of communities and enable new forms of distributing knowledge. New types of self-images and ways of living are negotiated within gaming culture. At the same time, as subjects the players also submit to specific moral and institutional codes and norms – and these must be mastered in order to succeed. Computer games thus exemplify the contradictions inherent to the emergence of cultural meanings in the field of tension between conformity and dissidence under media conditions. Collective gaming environments exemplify post-territorial, translocal communities, whose identity concept oscillates between disciplining and uncontrollability – a collaboration that is both real and unreal at the same time among individuals, whose status appears to follow different virtual and real regulations.

Already starting with the form of television and the remote control, intensifying with the mouse and becoming almost radically manifest in the immersive presence of computer games, how we think of the media, of the world, of things and of the self is conditioned by the media to selectivity, contingency and alternatives. The concept of contingency is especially important here, as it indicates the unmistakable potential especially of collectively and collaboratively defined media environments, yet not only media environments, but also and especially the socially defined environments that are linked to media: with the intervention of everyone else in multi-user environments, with the contributions from everyone else, every story can always end differently. The successful computer games are not the ones based on striving for high scores, but those which, as open spaces of action, empower players to play the game differently each time, thus redefining it, in a sense, each time. Computer games increasingly conceive of players as culturally competent actors and afford them a high measure of possibilities for action within multiple plots.

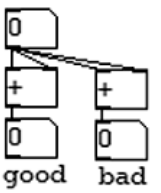
And finally, both production environments, as they are imagined for the creation of Open Source software or GP Licenses, and what has been outlined here as collaborative artistic media practices point in the direction of a relatively new and specific concept of production itself, less in the sense of an anti-economic or post-economic concept, but rather – possibly – in the sense of a production form that moves outside the realm of these two poles: then producing something no longer means making something new, whether in teamwork or virtually from nothing as a *creatio ex nihilo*, but instead understanding the production concept itself as the use of something that one has specifically *not* produced, but which we can still, within the framework of this use, in accessing and appropriating it, at once change, undermine, reinforce, reorganize or even completely reject. What is involved here are a new kind of spaces of possibility, in which no more sharp boundaries are drawn between production and consumption, and in which the distinctions between innovation and appropriation are blurred. This means that we can generally not respond to concepts such as production, development, consumption, entertainment, etc., without knowing in terms of which social community or society the question is to be answered. In each case we arrive at statements about the role or function that something plays or assumes within a communicative and thus always new collaborative practice of this society at a certain time.

However, these spaces of possibility first emerge in a complex interplay of cultural exchange conditions, in which ideas about bodies, apparatuses, technology, society, communication, the social, the public sphere, competition and much more are inscribed. Each successful realization of a possibly new disposition of the interconnection of these components (as in the case of cinema or computer games) follows a form of collaboration, which is not only to be understood as a media environment, but is to be described as a simultaneous and especially also as a collective cultural practice. This in turn means that the production environments and conditions of phenomena such as Open Source are possibly not all that new, but that they may hopefully continue the success story of collaborative production environments as a culturalization of the technical and technology.

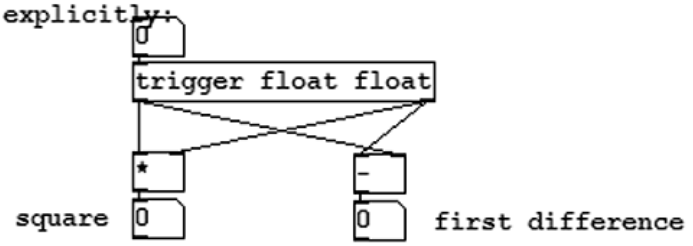


In Pd, most objects carry out their functions when they get messages in their leftmost inlets, and their other inlets are for storing values that can modify the next action. Here, the "+" object does its thing only when the left-hand input changes.

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.

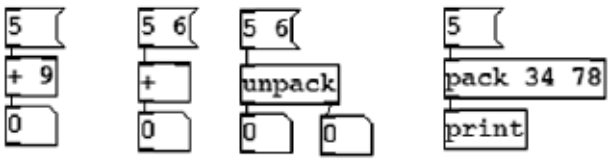
An ability to process MIDI signals is common among most currently available audio software programs, and Pd is no exception. By using a set of modules for processing MIDI data, one can create new MIDI programs that generate the audio output within Pd or receive and process MIDI commands from a variety 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

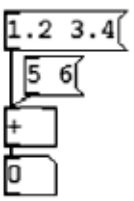
digital data. Creatively using its various features, multimedia artists or musicians, even those without special programming skills, can relatively easily build programs tailored for the realization of their unique ideas. Pd makes the act of artistic creation an exhilarating game, permeated with a sense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

Most Pd messages are just numbers or short lists of numbers:



If you send a list to an object with more than one inlet, the items in the list are spread out over the inlets, as seen in the 5+6 example above.

Unlike Max, in Pd all numbers are floating point. Numbers whose values happen to be integers are displayed without decimal points.



For more on messages, get help on any message box by right-clicking.

updated for Pd release 0.33

Pd as Open Source Community

Andrea Mayr

Introduction

Pd is a program developed by a community of electronic musicians in the style of an Open Source project.[1] The goal of this article is to make concrete the two central concepts of this general statement which in one form or another keep coming up in this publication. What are the aspects of an Open Source project which shape the development process of Pd and what does the community which supports this project actually look like?

Pd as an Open Source Project

Pd is Open Source Software distributed under the popular BSD license, which permits but does not require the publication and modification of the source code. It is thereby possible that Pd code becomes incorporated into proprietary software. Eric S. Raymond named three essential points as characteristic for Open Source projects: self-motivation, the openness of the input, and flat hierarchies.[2] While large, increasingly professionalized Open Source software projects, such as the Linux kernel, no longer conform to this pattern, these points are still characteristic for the dynamics of development of the Pd project.

Many Open Source Software (OSS) projects begin with a programmer encountering a problem for which there still isn't an adequate software solution. There are then two possibilities. In case the problem cannot be tackled within an existing program, a completely new project must be started. Miller Puckette did this when he wrote the original version of Pd. It is often the case that already sizeable code segments from other programs can be used. If other developers are found who find the same problem urgent and consider the proposed method of solution sensible, the cooperation can begin.

The most common case by far is that the problem is solved by changing or expanding an existing OSS program. In this case, the most efficient strategy is to write the new function so that it

---

1 This article is based on quantitative and qualitative surveys which the author conducted within the framework of a research project on virtual work teams in 2003. In the middle of 2005 some quantitative aspects were checked again and it was established that although the community had greatly grown, the core of developers had remained relatively stable. Thus the data from 2003 still permits a good insight into the structure of the Pd community.

2 Eric S. Raymond. The Cathedral & the Bazaar. O'Reilly 2001 <http://www.catb.org/~esr/writings/cathedral-bazaar/>

can be added to the existing program. Since the same problem is often tackled by several people at the same time, it makes sense for them to cooperate with one another and not to develop everything by themselves. This is the crux of what the Pd community does. It expands the area of application of the core of the program originally written by Puckette. The individual needs of the members who want to use the software for their own artistic projects are crucial here. Their engagement also stems partly from the processes of group dynamics. Pd is perceived very positively in reflection within the group. Connected over the Internet, the group consists of people trained in very specific areas who help each other. The emergence of small local groups who meet each other in face to face situations has also been announced in the international mailing list. Beyond the engagement in solving purely technical problems, a great potential for identification with the group thus arises, which as a motivational factor ultimately has an impact on (artistic)production wit Pd. In other words, the development of Pd is determined by the individual needs, motivations, and interactions of the community members.

The openness of the input of OSS projects is reflected in the fact that all developers and users have the possibility to influence further program development. This can range from submitting a simple bug report to taking over the responsibility for the maintenance of a segment of the program. Open input has two advantages. First of all, mistakes are found and repaired more quickly since all users are (potential) testers and co-developers. A forum is provided for those who want to play this part actively, one within which their expertise can be of maximum use. In addition, new ideas can be introduced from all quarters and a collaborative learning process within the community of developers is encouraged. Both aspects are very important for the vitality of Pd as a program and as a community. In order that multiple contributors can be incorporated into the project, it is necessary for the software's architecture to have a highly modular character. This permits many people to work in parallel to one another on the pieces which interest them the most without extremely increasing the coordination effort in the community. Modularity in Pd is very highly developed. In Pd, it is possible to enlarge the core of the program with single external program parts and to provide it with the relevant function.

The possibility to be able to link these externals modularly in the operation of the program, and thus apply an experimental approach to programming without having to change Pd as such, excludes many grounds for confrontation which another program architecture would involve. If someone wants to have a certain Pd function which is available in an external at his or her disposal, this person can start the program so that it is available. Others do not have to do this. A modularity of this kind enables the offering of a nearly unlimited variety of externals, a variety which in the meantime already exists and which long ago surpassed the original estimations. However, the maintenance and description of these externals is in this respect problematic because there is no plan for naming them or for identifying their function. The suitable documentation and organization of the externals constitutes an important task within the project team.

The integration of externals on the one hand allows the community of developers to remain open and on the other hand provides for the integrity of the platform through a very flat hierarchy. Miller Puckette indisputably holds the highest authority within the community. He wrote the core of the Pd program and was significantly involved in the organization of the community. In the meantime, other programmers have also contributed to this core. The porting of the program to another operation system is an example of an addition which had to be made to the original Pd program itself. Miller Puckette was happy to incorporate these kinds of expansions. Despite Miller Puckette's authoritarian manner concerning the expansion of the Pd program, he enjoys the highest standing within the community. In hindsight, decisions in which he rejected suggestions are assessed as positive. The process of change and of the distribution of the core is clearly hierarchically incumbent on Miller Puckette. He comments regularly on the mailing lists relevant to development and collaborates with other community members to improve the program. Individual team members position themselves within the community depending on their specific capabilities, their achievements, and their communication skills.

The writing and the distribution of external program functions – which can no longer be differentiated from the inherent parts of the core of the program when the program is run – are available to everyone. This project architecture makes possible and permits a wild mushrooming of external program fragments while Pd as the core remains untouched. A similar structure is replicated in the larger externals, which usually have a "maintainer" who maintains the input.

Pd as Community

The next section attempts to answer the questions who the people now taking part in the development of Pd are, which factors can be appreciated for fostering motivation, and which further socio-demographic conclusions can be drawn from the individual surveys.

Pders as a Team

The Pd community consists of a large group of people adept at technology who communicate with each other predominantly via internet based media. They do this in order to exchange information regarding the use and further development of the Pd program and its expansions.

There is a small group of programmers who form the core of the community with a smooth transition to a comparatively large circle of users with consistently very high technical qualifications in the area of digital and/or analog signal processing. Most have already used the program over a long period of time, employ it experimentally, and often implement very specific applications. These users are in large part also those who point out specific ways of looking at

a problem in their applications and thus have a hand in an important part of the process of stabilizing the program.

What can be expressed well in numbers are the results of a survey of a mailing list that is a central means of communication pertaining to the further development of the program: Pd-list, the official mailing list for the Pd-Community. Pd-list was originally the only list which supported the community. Further distributors followed by the names of Pd-dev, Pd-announce, and Pd-off-topic. Three further communication channels were set up for topic centered work between February 2003, the time of the first survey, and the time of the second in August 2005: Pd-cvs, Pd-web, and gem-dev.

The first list was started in January 1998 at the Institute of Electronic Music and Acoustics (IEM) at the University of Music and Dramatic Arts at Graz. With support from Miller Puckette and those who were helping him with the development of the program at this time, access to the production of Pd and its applications was offered to the students at the institute at the same time.

The number of subscribers to the respective lists in February 2003

List name	Subscribers	Inactive Status
Pd-list: (main)	584	(90)
Pd-dev: (developers)	410	(70)
Pd-ot: (off-topic)	185	(17)
Pd-announce	440	(54)
Total minus overlap due to subscription to several lists	716	~ 16%
Email addresses of active subscribers (approx.):		600

The number 600 in June 2003 can thus be seen as an indicator[3] of the part of the community which accesses the mailing lists as a means of information and communication. According to research from this study, this applies to the majority of the members of the community. Up to August 2005, the number of subscribers to the community oriented Pd-list increased to 1273, whereas about a third is inactive, i.e. is not receiving any emails at the moment. In contrast, the programmers' list, Pd-dev, remained relatively stable. The analysis of the email addresses of the subscribers to all lists yields a quantity of 1536 unique addresses, approximately 30% of which are inactive. The total number of addresses of active subscribers is 1078 in August 2005. In relation to the 600 in February 2003, this is an increase of approximately 80%.

[3] It should be noted that there are factors which influence the upper and lower limits of this value. The possibility of the double subscription of a single person with different email accounts as well as the registering of email lists as single subscribers make this figure just an approximation. That a real person can be found behind each email address is naturally a supposition and must be accepted with reservation.

The following analyses were drawn up in the middle of 2003 and are based on the figures available at that time. Because they concern the core of the community, which proved to be relatively stable over this period of time, they can still be regarded as representative two years later. The survey on geographical variation is based on the analysis of the mailing list archives from 1998 to 2003.

Geographical Variation

As the central tool for communication, whose archive is visible on the World Wide Web, the mailing list provided good conditions to approach an estimate of the community in different categories. The number of several hundred community members diminishes greatly as one's gaze moves from the long list of the mostly passive registered subscribers to those who actively contribute to the group.

The archive goes back to the beginning of 1998 and right away in the first sporadic pages gives an impression of the dynamics within the Pd work group. Individual contributions to the archive are generally sorted by the date of their arrival; during the twelve months of the year, this chronological order is interrupted only by the summary of the main themes, or "threads".

The parameters for the sender of each message – the address of the sender, the date specifications, and the content – are included in the individual messages in the archive. Next to the time when a message was sent to the list of recipients, the date specification also reveals the time zone that the message was sent from. Along with the Top Level Domains (.at, .edu, etc) of the Internet addresses, this information helps sketch a geographical net that permits drawing conclusions about concentrations in particular areas in coordination with the frequency of particular country codes. It can be observed here that the geographical center of the community has shifted over the years to Europe, especially western and central Europe. The USA, above all the coastal regions, provides a further focal point. Australia and Japan are also represented, albeit relatively weakly. Thus Pd perfectly mirrors the global geography of electronic music culture.

Pders individually

Pd is not a tool which can be used intuitively. It requires above-average knowledge of information technology as well as an engagement with the operation mode and the application mode of the tool itself. These conditions create a picture of the community. It seems very homogeneous.

The attempt to understand Pders individually can naturally only tend to sketch a picture of the typical characteristics of people who belong to this group.
A questionnaire available at a password-protected, specially made web site was designed to approach this problem. The answers of 38 people were analyzed.

More than 60% of those who supported this survey had already been in the community for longer than two years. 13% had been there longer than the existence of the Pd-list mailing list, thus longer than four years. 23 of the 38 (60.5%) had been a part of other software development groups within the previous year.

Many artists, above all musicians, are a part of this group. Pd seems to provide a basis which picks out the fields of art and technology together as central themes. A differentiation in the interpretation of both of these fields concerning work with the program can be found here only in individual cases. The border between those who refer to themselves as artists with a good knowledge of programming and those who see themselves as programmers who use their code creatively is fluid.

The questionnaire asked for self-assessment on a five-stage scale between "user" and "developer". In the context of Pd, which is designed as a program with a target audience of musicians and artists, it can also be interpreted as a scale which can be understood between the use of Pd as a musical instrument and the programming of Pd. The evaluation revealed a great balance in the spectrum between user and developer with a slight surplus on the side of the users:

	user	< >	**middle**	< >	**developer**
%	18.4	21.1	26.3	21.1	13.2

The primary professional occupations in the group questioned does not seem to depart greatly from Pd relevant themes. In 29 out of 38 cases, it can be assumed that their professional work, which they could describe in an open field in the questionary, was related to the work they did within the community. In answer to the question of whether direct or indirect monetary income resulted from work with Pd, 36.8% of the respondents answered yes, 50% answered no, and 13.2% found the question inappropriate.

More than 65% declared their average age to be 20-29 years old. The rest are older. There was nobody under the age of 20.

Pders are predominantly male. The interviewed group was in fact 100% male.

What was emphasized strongly in both the survey and the participants' observations was the existence of a very distinctive group consciousness within the community. Get-togethers of the Pd community are similar on face-to-face meetings of local Linux User Groups. These kinds of opportunities exist in Vienna, Berlin, Ghent, New York, Cologne, and Barcelona. Instead of chatting comfortably in bars like the LUGs do, local Pd groups transfer/move/shift their meetings in clubs and use this to present their work to the public.

By this point it seems to crystallize that the dynamic of this program has a strong subcultural character. The factor of reputation, to which great importance is attributed relating to free

forms of collaboration on the net, also has an effect in the area of art as a sign of quality. This is similar in the local environment of small scenes. The increase in the value of an individual's reputation during a short face-to-face meeting naturally follows other rules than that of the development of a product online. Nevertheless, they enhance each other. Incidentally, a greater public presence through such regular meetings could offer the possibility to change the homogeneity of the group, for example in favor of changing the gender ratio. This seems to have begun to happen in the past two years.

Communication

Communication takes place online and offline. 22 of the 38 questioned declared that they also spend their free time with other group members. In regard to Pd-specific themes, chiefly the mailing list Pd-list is consulted as a linking element. The flow of email from this list increased from 167 messages in its first year in 1998 to over 3200 messages in the first half of 1998 and to approximately 4500 in the first half of 2005.

Other tools are in use in addition to this list. In the survey, the following frequency of other forms of contact was indicated (data in %).

	Never / Not applicable	Less than once a month	About once a month	About once a week	A few times a week	Daily
Mailing List	5.3	5.3	15.8	18.4	31.6	23.7
Personal Email	13.2	23.7	15.8	21.1	18.4	7.9
Personal Telephone Conversation	63.2	15.8	7.9	7.9	0.0	5.3
Face-to-Face Interaction	31.6	26.3	15.8	15.8	7.9	2.6
irc (Chat)	92.1	5.3	2.6	0.0	0.0	0.0

Data in %.

An enormously wide spectrum of internet resources is set up to exchange information and data. Along with Community Platform, Wiki, and CVS (Concurrent Versions System) for the current version of Pd, the pages of Miller Puckette and the IEM play a great role.

The incidence of use of these tools turned out as follows (data in %):

	Never / Not applicable	Less than once a month	About once a month	About once a week	A few times a week	Daily
Community Platform	21.1	21.1	28.9	23.7	5.3	0.0
Websites	5.3	31.6	39.5	15.8	7.9	0.0
Wiki	39.5	31.6	18.4	7.9	2.6	0.0
Databases	13.2	42.1	23.7	15.8	5.3	0.0
CVS	39.5	21.1	23.7	7.9	5.3	2.6

Data in %.

There is a webring for networking at the individual website based information user interfaces. In the past two years, this has more than doubled and now comprises 27 independent pages.

Motivation

Pders consistently work autonomously and are self-motivated. They go back to the most diverse resources in order to obtain the best possible support for their efforts in the further development of their work. They are rewarded to the extent they are esteemed not only within the group but outside it as well. The implementation of problems in commissioned work or in self-initiated artistic works through the use of Pd has an impact here. They gain esteem and a position within the group from a combination of their specific qualifications, social skills, and the amount of their effort.

An informal coordination of the field of activities distinguishes the project and a friendly mood characterizes the atmosphere of communication in the mailing list.

Satisfaction within the group is enormous.

Our group morale is high.

not applicable	strongly disagree	disagree	agree	strongly agree
13.2 (5)	0.0 (0)	2.6 (1)	63.2 (24)	21.1 (8)

I enjoy being a member of this community.

not applicable	strongly disagree	disagree	agree	strongly agree
2.6 (1)	0.0 (0)	0.0 (0)	52.6 (20)	44.7 (17)

I gain intrinsic reward from it.

not applicable	strongly disagree	disagree	agree	strongly agree
7.9 (3)	2.6 (1)	10.5 (4)	50.0 (19)	28.9 (11)

It's fun.

not applicable	strongly disagree	disagree	agree	strongly agree
2.6 (1)	2.6 (1)	0.0 (0)	44.7 (17)	50.0 (19)

It's fame.

not applicable	strongly disagree	disagree	agree	strongly agree
15.8 (6)	21.1 (8)	34.2 (13)	23.7 (9)	5.3 (2)

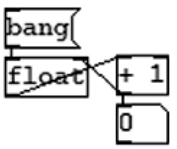
Data in % (Number of people)

Conclusion

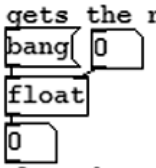
Pd is a successful OSS project. The functionality is continually being expanded and the community is growing without losing its original character. In contrast to the stability of the list of developers, the strong growth of the community-oriented Pd-list suggests that a differentiation between Pd users and Pd developers is slowly beginning to emerge whereby the Pd users possess above-average technological know-how and are not pure users. This differentiation between users and developers, which is not codified in the program itself but which arises from individual practice, is quite typical of advanced OSS projects.

The community itself seems to be handling the growth well. The mixture of individual motivation (which determines the speed of development and the direction) and organizational consolidation (which provides for a certain stability and continuity) permits very productive and innovative work. The very availability of this publication is a testimony to and a part of the process of development of the community.

Here's a simple counter. Click repeatedly on the "bang" message to see it:

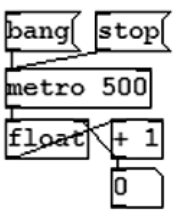


The "float" box is a storage element holding one floating-point number. The cold inlet (i.e., the one on the right) stores numbers. Sending the message "bang" to the hot inlet gets the number back out:



Float's outlet above is connected via + 1 to its cold inlet. The incremented value is stored for the next "bang" to spit out.

Here's a timed counter. Hit the "bang" to start it...



<-- new object: metronome. The "500" means every 500 milliseconds--i.e., twice a second.

updated for Pd version 0.34

I'm the Operator with the Pocket Calculator
Some Reflections on *Pure Data.*

Thomas Musil / Harald A. Wiltsche

1. In the *Tractatus logico-philosophicus*, Ludwig Wittgenstein wrote that the only purpose of philosophy is to contribute to the "logische Klärung der Gedanken" [logical clarification of thoughts] (Wittgenstein 1963, 41). There is much false but also some truth to this sentence. It is true that discussions are often clouded by premises which, when actually spoken, seem completely clear and almost banal, yet which cause great confusion if you forget to explain them. Logic is acquainted with this phenomenon under the term of enthymeme, and not often have highly astute arguments broken down because someone forgot to formalize and include sentences such as "All humans are mortal" in the set of premises. We will not conduct a logical analysis here but will bring to light some enthymemes that have significantly obscured past discussions of the programming language Pd instead, in order to clarify in at least some instances what we are actually talking about when we speak of art and Pd.

The thesis of the following essay can be explained quickly: Pd has grown out of a special scientific culture which has decisively shaped our western understanding of reality and the world. Yet Pd is a tool[1] for the production of artistic artifacts.[2] Out of this Janus-faced character grows a theoretical problematic recognizable in many different aspects, although we can only examine a few facets here.

2. Historically regarded, a close relationship between music[3] and science is in principle nothing new. As part of the classical *artes liberales*, music has always been situated in direct proximity to the mathematical disciplines and had already developed strict musical formalism in the early Middle Ages. A new level of connection between music and science arose from the electrification of music, which had already begun in the eighteenth century,[4] probably not coincidentally at a time when the triumphal march of modern science had already gotten completely

---

1 A note on the terminology: to have to understand Pd as a tool seems to us the only reasonable procedure for describing what Pd performs and can perform: Pd is something that makes it possible for us to generate something else (namely an artistic artifact). We would likewise characterize hammers, chisels, and brushes as tools. Voices which require one not to characterize Pd as a tool may represent an especially advanced concept of a tool which we do not share, or they are supporters of a digital mysticism which is very widespread in certain places, of which we are critical. Pd is still not an instrument per se. An instrument can be programmed in Pd.

2 We understand Pd here as a tool for the production of art knowing well that in principle, Pd can be used in the most diverse areas. However, such areas of use do not interest us at the moment.

3 It is in keeping with our theoretical interest to speak principally of music, even if most arguments can be extended to the whole system of art more or less without problems.

4 The first electronic instrument in the broadest sense was constructed around 1730. (Ruschkowski 1998)

under way. When we speak of the triumphal march of science in the eighteenth and nineteenth centuries, the fundamental changes which were actually occurring then are slightly hidden from our present gaze. At this time, science emerged from the salons of the European elite, where it had served up until this point for aristocratic amusement, and changed the *Lebenswelt* of the masses to an extent not yet known: the changes in the area of the communication technology has had a fundamental impact on the general perception of time and space. The incipient industrial revolution changed the social situation of the masses and scientific discoveries, such as that of infrared and ultraviolet light, confronted people with phenomena for which there is no natural sense organ. In a word, science spilled into people's daily lives and with it a certain way of thinking, in the center of which "method" seems to stand.

Why method? When dealing with questions of concept formation and the construction of theory, you very soon come to the conclusion that theoretical activity always means reduction. The concept of method describes this precisely: "The term 'method', strictly speaking, 'following a way' (from the Greek *meta*, 'along', and *odos*, 'way'), refers to the specification of steps which must be taken, in a given order, to achieve a given end." (Caws 1967, 339) With Descartes, Galileo, and several others, a method was established in a strictly literal sense, a method whose innovation can be seen in the linking of an empirical approach with mathematics. Although thinking in its entirety was still very much shaped by magic and alchemical motives at this time, and researchers like Kepler were still adamantly convinced that horoscopes could be accepted as an inherent part of science,[5] on the side of method, it was clear that only that which had a place within the narrow constraints of mathematics could be a part of science. Thus it is also not astonishing that the significant distinction between primary and secondary qualities was formulated precisely at this time: extension, figure, motion, rest, solidity or impenetrability, and number are to be viewed as primary qualities, while the secondary sense qualities are color, smell, sounds, taste, etc. It is striking that the area of primary qualities is virtually defined as being able to be expressed in mathematics while this does not hold true for the secondary qualities. If the Cartesian concept of the duality of the world is added to the realm of the subjective and to that of objects in nature which have extension, as well as the notion that the book of nature is written in the language of mathematics, an image arises of a world which is able to be subjectified, expressed in mathematics and experimentally and scientifically analyzed through and through. A successful but nevertheless artificial method, namely that of empirical research and the reliance on mathematics, was stressed in such a strict sense that it mutated gradually from *a possible world view* to *the only possible world view*. Empiricism, early, middle and neopositivism, psychologism, biologism, AI research which has been promising results for decades, current trends in brain research which are on the point of scientifically arresting the soul and free will (cf. e.g. Monyer et. al. 2004), as well as countless contemporary opinions of individuals (e.g. Sellars 1963, 1-40) confirm in an impressive manner our cursory remarks on the history of science.

---

5 In addition, Kepler's *harmonices mundi* can serve as further evidence for the close connection of music and science.

The scientific way of thinking – with the strong support of some philosophical currents – has become a monopoly in questions of the observation of the world, while possible gaps are filled by esoteric obscurantism of the most varied provenance. Niklas Luhmann spoke of *symbolisch generalisierte Kommunikationsmedien* (symbolically generalized communication media) and thought of those cases in which scientists are interviewed on television about subjects which are not in their area of research, and they are believed because they participate in the medium of scientific truth and not because their arguments are so convincing. Only sluggishly has the philosophy of the twentieth and twenty-first centuries rediscovered what this perspective hides if it is made absolute.

3. What does this have to do with Pd? It must first be stated that Pd stems from precisely this mathematical and scientific line of thought and for good reason. Making instruments (and in the most general sense Pd fits into this category) has largely been dependent on science since the time of electrification at the latest. Science "lives" to an extent on omitting the subjective meaning, i.e. those structures which are attached to the secondary qualities in order to attain a mathematical and empirical reduction. Art, however, is perhaps one of the last peepholes into the aforementioned world of *Sinn* (meaning), into a world *before* scientific reduction.[6] If naively asked now what Pd actually is, a tool for the production of art, thus a tool for the pure formation of meaning or a scientific artifact, at the same time the obvious response ("Both!") is heard; with a trivial question, on has come to the crux of the problem. Work in Pd runs the risk of getting caught up in an infinite loop between the area of *the exclusion of meaning* and that of *the formation of meaning*. Work in Pd – expressed differently – very easily becomes a scientific experiment where the depths of what Pd can provide – and not what *meaning* (and thus *art*) can provide – are plumbed.

Let us pause for a moment and envision what we don't mean by this: technical innovations naturally play a role in the history of art, but they do not play a role in art. The following is meant: Nam June Paik could never have created his installations if the technical equipment he used hadn't existed. The technical equipment doesn't make a difference; the point of this art: Nam June Paik reflected a slice of the world, tinkered with the meaning inherent to this facet of the world, and used equipment that was on hand and seemed fitting to him. For art as such (for the essence of art),[7] the singling out of Nam June Paik's works is irrelevant, equally as irrelevant as nearly triangular objects that actually exist are for the geometrical idea of the triangle.[8] In this regard, it also does not change anything that Nam June Paik explicitly picked

---

6 Is it a coincidence that Kant begins his heavily invested project of creating a base for pure science with *transcendental aesthetics*?

7 To say it once again with all clarity: in no case do we claim to have the power to define what art is when actual works are concerned. Reflecting upon what art is according to its intention, the area of individual works is not at all restricted. However, let us admit that religion and science are in essence different, even though there are single scientific or religious manifestations where the border is difficult to draw.

8 This can be approximated by making oneself aware that saying the Pythagorean theorum twice does not double its truth.

out technology as a central theme, for in his art, this is only considered a slice of the meaning of the world among other things.

For Pd, this means that the question of whether a piece was composed with *MAX* or with Pd is not only difficult to decide; it is extrinsic to the essence of art. It can naturally be the case that Peter would rather listen to electronic music, while Berta favors the sound of the violin, just as Fritz realizes his artistic ideas on the computer, while Anna prefers the recorder; however, individual works and producers, like individual recipients, change nothing of the essence of art.

The impression could now arise that we are speaking of abstractions removed from the world, the essence of art, that we are blind to the actual development of works and represent a more or less hidden Platonism. We do not dispute that the technical, social, technological, and historical contexts of the production of art represent an important area of analysis; however, these factors act on another level. In Pd, however, we see the danger of losing sight of the essence of art precisely because it is a part of applied mathematics and logic similar to the early years of media art when the content of an installation sometimes seemed to lie in sending data packets from Linz to Tokyo, Sydney, New York, London and Paris and back again to Linz. In these cases, it is doubtlessly a matter of scientific experiments, and those who enjoyed this took pleasure in the excitement of scientific and technical achievements. This is not meant to be derogatory, since pushing the technical limits is of great importance; in the majority of cases, however, it has nothing to do with art: virtuosity (a great knowledge of programming) alone does not touch art as such, like urinals in galleries and four minutes and thirty-three seconds of silence show. Urinals and silence alone do not make art, also the essence of this seems to lie somewhere else: we think in the structure of meaning. That the method of sensory exclusion which lies behind Pd holds certain dangers for dealing with structures of meaning is obvious, even when it is a matter of a method that largely shapes our everyday life and our everyday perception.

The following is to be said of Pd: Pd is not an empty canvas per se, for behind Pd stands centuries of history of scientific ideas. Pd can become an empty canvas just as hammers, chisels, brushes, violins, and jackhammers can. Pd is not an art. Pd can be used for the production of art, just as hammers, chisels, brushes, violins, and jackhammers can.

Literature:

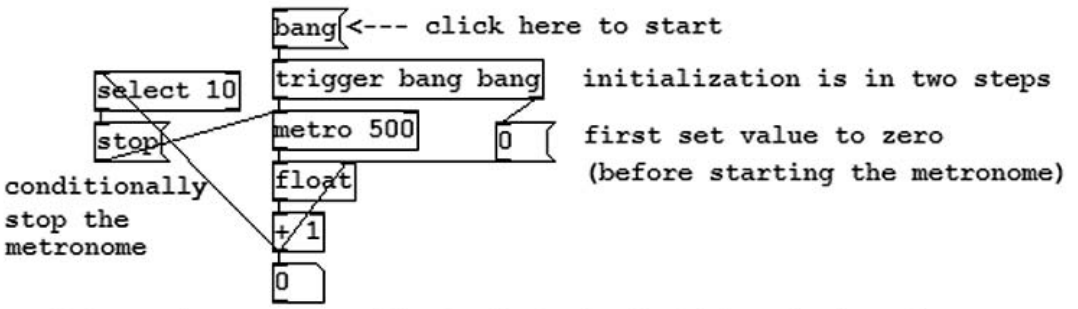
Caws, Peter. "Scientific Method" Encyclopedia of Philosophy, Vol. 7. Chicago 1967.

Monyer, Hannah/Rösler, Frank/Roth, Gerhard/Singer, Wolf/Elger, Christian/Friederici, Angela/Koch, Christof/Luhmann, Heiko/van der Malsburg, Christoph/Menzel, Randolf. *Das Manifest. Elf führende Neurowissenschaftler über Gegenwart und Zukunft in der Hirnforschung*. Gehirn & Geist 6 (2004): 30-37.

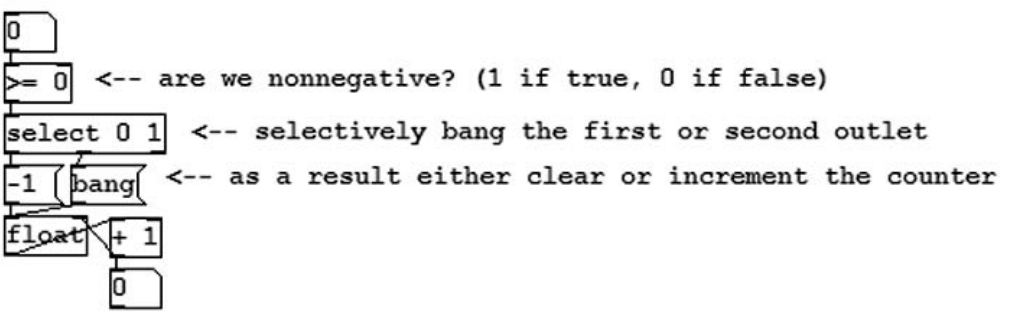
Ruschkowski, André: *Elektronische Klänge und musikalische Entdeckungen*, Stuttgart 1998.

Sellars, Wilfrid: *Science, Perception and Reality*. New York 1963.
Wittgenstein, Ludwig: *Tractatus logico-philosophicus. Logisch-philosophische Abhandlung*. Frankfurt a.M. 1963.

Here's a counter that counts from 1 to 10:



We're using one new object, "select, " which outputs a bang when it gets a matching value (10). This is useful for doing conditional computations, such as this one which counts while its input is 0 or positive but clears when negative:



updated for Pd version 0.34

Basic Physical Modeling Concepts

Cyrille Henry

Introduction

A computer processing unit (CPU) is a device able to perform mathematical operations with data stored as electrical voltage. It can only perform simple operations, but so many of them can be done in a second that it is possible to create a program (a defined combination of operations) to perform complex tasks. However, the processing unit of a computer is limited to processing virtual information that is electrically coded inside the computer. This information can be connected to the real world through an interface (a keyboard, a screen, etc.) but the main processing unit is not subject to physical rules like human being is. It is obvious that gravity has no effect on the way a computer performs its task; this also means that the computer has no idea what gravity is.

If you want your computer to move a fader like you do, you have to teach it how to do so. You must also teach your computer how gravity, the fader, and your muscles interact with each other in order to change the position of the fader.

First, we will compare and analyze the way a human moves a single fader (such as a mixer fader) with the simplest way a computer moves a virtual fader. Then we will explore certain physical modeling tools as a solution to teach a computer the basics of physical notions and will give a short overview of the possibilities of such tools. This article is not a complete mathematical or physical justification of particular physical modeling; rather it introduces basic ideas in order to allow anyone to use these tools.

Human / Computer Differences

Unlike computers, human beings live in a world with physical rules. Gravity, forces, and energy control their movements. The difference can be shown by studying the movement of a simple fader over time. A mixer fader can be used to change the amplitude of a sound, its frequency, or any other audio synthesis factor. We will focus only on the position of the fader and not on the effect of this movement on the synthesis.

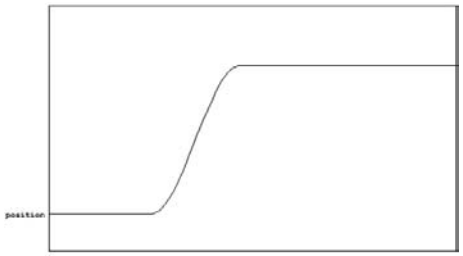


Figure 1. Human fader movement over time

Figure 1 is a typical example of a fader movement over time. The fader is moved by a human.

In this simple example, we can compute the velocity of the fader (how fast the position of the fader changes) and then its acceleration (how fast the velocity of the fader changes).

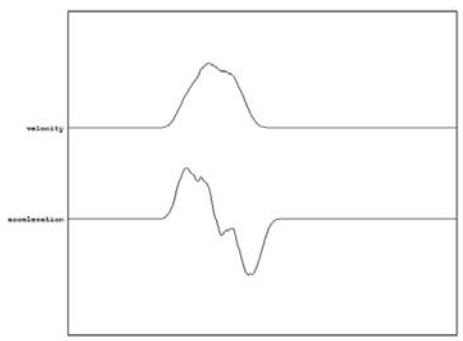


Figure 2. Human fader movement: the velocity and acceleration of the fader over time

Notice in Figure 2 that the velocity of the fader gradually increases from 0 to a maximum velocity, after which the velocity is almost stable for a short time before returning to zero. The acceleration is positive during the first part of the movement (while the velocity increases), almost null in the second part, then negative during the last part of the movement (when the velocity returns to zero). The movement is composed of these three periods.

Figure 3 represents the position, velocity, and acceleration of a virtual fader moved by a computer using pure-data.

The position of the fader changes instantaneously from one value to another. This very fast movement is highly unnatural. We will try to slow down this movement using a [line] object. This object slows to a specified target over a given time. It offers a linear variation of its output.

The movement in Figure 4 still looks different than the one made by a human.

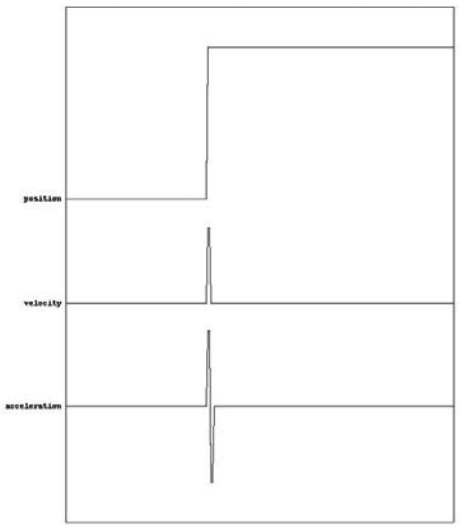


Figure 3. Switch from one value to another with a computer

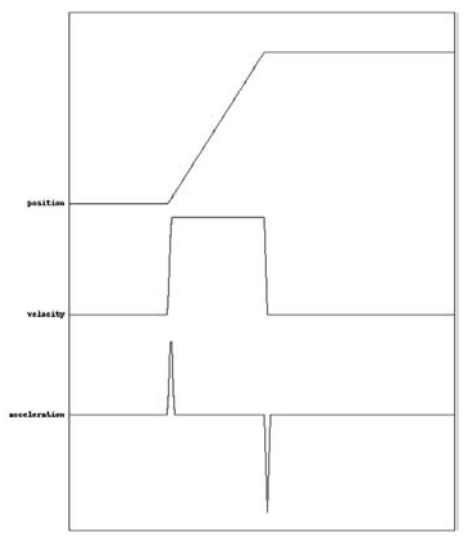


Figure 4. Computer fader movement: the velocity and acceleration of a fader over time

In the example in Figure 4, the fader has neither an acceleration nor deceleration period. The velocity changes almost instantaneously. This movement looks very "robotic" because the fader acceleration or deceleration is concentrated at the beginning and at the end of the movement. In a real movement, this acceleration and deceleration continues throughout almost the entire movement. To compensate for this very short acceleration and deceleration time, the acceleration is very great. The computer does not have any problems moving the fader in this unusual way because it only moves a virtual fader. Unlike human movement, no energy is involved in this virtual motion.

Simple pmpd Description

Aim of pmpd

pmpd provides objects for the simulation of physical behaviors such as basic energetic and movement related objects. These are mass and link, allowing particle-based physical modeling. They simulate the behaviors of a single mass or link (a link is, for example, a spring between two masses) and react like ideal objects in a physical world. Just as a CPU performs complex tasks with simple instructions, assembling this small "physical" element can generate complex behaviors due to interactions. pmpd allows the user to create a network of masses and links, to simulate the behaviors of this system, and to interact with it in real time while sending forces or changing a physical parameter.

Mass

Masses are objects that react like a virtual mass. They are defined only by their position in space and their weight. The only thing a mass can do is to move according to Newtonian dynamics. This means that applying a force to this mass will change its velocity. The [mass] object has one inlet to receive forces and one outlet for its position (another outlet provides information not essential for the simulation). The pmpd mass is punctual, which means that it has no volume and cannot rotate. Although this does not exist in the physical world, it can still be a good approximation. We will see later how to simulate massive objects.

The physical notation of Newton's law is F = ma where:

F = the sum of all forces applied to a specific mass
m = the weight of the mass
a = the acceleration of the mass

This physical law means that the acceleration of any mass is proportional to the force applied to it, but it also depends on the weight of the mass. This equation rules all non-relativistic physics and is the only equation for the movement of mass.

Link

A [link] object creates a connection between two masses. It allows the masses to interact with each other, i.e. to exchange energy.

A link needs the position of two masses in order to compute the two different forces to be applied to the two masses. Since the forces depend on both the position and the velocity of each mass, the [link] object has two inlets and two outlets.

A link can be elastic if forces depend only on the distance between the two masses. The force of an elastic link is proportional to the elongation of the link. The elastic link does not introduce any energy loss but can convert kinetic energy (energy related to movement) and potential energy (energy related to a deformation).

The viscosity link is related to energy loss. The forces, proportional to the damping, are opposite to the movement of the mass.

pmpd links are all visco-elastic, the sum of an elastic and a viscous link.

The equation of a link is F = KL + DdL where:

F = force generated by a link
K = rigidity factor
L = elongation of the link
D = damping factor
dL = elongation variation of the link

This link is symmetric, so the forces sent to the two masses are opposing.

Metronome

Due to computer specifications, it is not possible to compute the position of a mass at all times. The position of a mass is therefore computed at a specific time, and a time reference is needed to perform a simulation. pmpd uses an "external" scheduler (a metronome). This metronome corresponds to the time discretization of the physical equations. This is another approximation of real behavior. To be accurate, this time discretization should be the smallest possible. The speed of the metronome depends on the velocity of the movement you want to simulate. The metronome should be faster than the highest frequency of the movement: the faster the metronome, the faster the simulation. Of course, the faster it is, the more positions need to be computed for each mass, resulting in more computing time.

Units

Just like Gem, the Pure Data OpenGL extension, pmpd does not use specific units; you can choose your own. If you must use units, you should try to use consistent ones, i.e. if you chose inches for the distance unit, then the unit of rigidity should be the unit of force divided by inches. Since everything is relative, a mass weighting "10" will react the same way to a force of "1" as a mass of "200" to a force of "20".

One, two, or three dimensional network

A pmpd physical model is composed of virtual objects (masses and links) that do not have any specific representations. In order to visualize the physical model, a mass can be represented by a circle or sphere and a line by a link. This is just a representation of the physical model, but another representation can be chosen for any system.

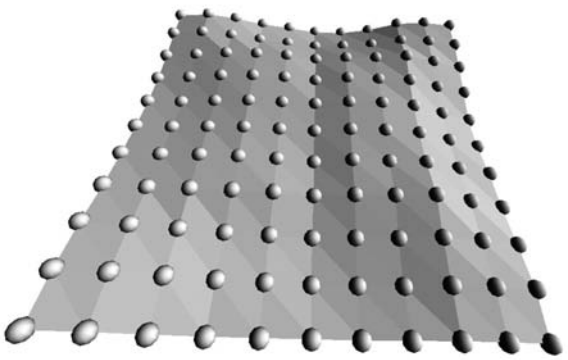


Figure 5. Visualization of a 1D membrane

Figure 5 is a representation of a physical model of an elastic membrane, like in a drum. This is a 1D system: the masses can move in only one dimension, but the representation of the system is in 3D space.
In order to model a massive "elastic" object, you can discretize it into small particles. Each particle can be modeled as an elementary pmpd mass connected to the rest of the objects by visco-elastic links.
pmpd does not allow the simulation of rigid bodies. Anyway, all objects can be seen as elastic bodies, with high rigidity.

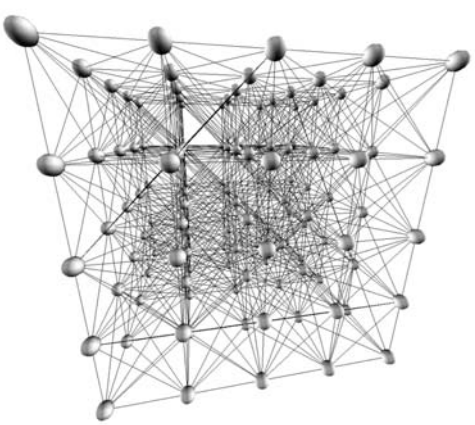


Figure 6. Example of 3D rigid body discretization

Figure 6 is a representation of a cube in 3D space. Many masses are needed in this example to model massive objects, but they also model the deformation of these objects .

Interactor

An interactor object acts as a link object but provides a patching facility. In effect, a single object can create an interaction with an entire class of masses.

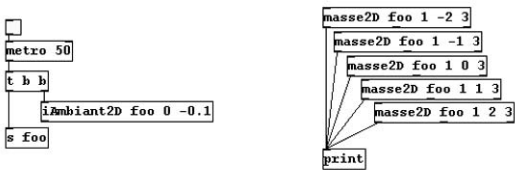


Figure 7. Sending forces to a class of masses

In Figure 7, all masses are subject to a constant ambient force. This force can be viewed as the force of gravity applied to each mass. Different interactor objects provide interactions with a point, a line and other simple primitives.

Test Objects

These objects test the position (as well as the distance, speed from a point, a line, orientation, etc.) of a mass. Thereby, Pd has access to much information regarding the state of the system. This allows interactions with the rest of the patch. Another test object gives information about the link (deformation, speed of deformation, orientation, etc.).

Limitation

The most commonly encountered problem when using this kind of physical modeling is instability. Instability comes from the approximations made with the discretization of the equations. To reduce the risk of instability, the model should be slowed down (increasing the metronome speed can be necessary to keep the desired speed of the simulation). This is usually not a problem due to the relatively low frequency of the simulation. Unlike physical modeling based audio synthesis, the structure usually doesn't need to be computed at audio rate but only at a few hundred hertz. This kind of simulation can thus be calculated for real time applications by a low-cost computer or laptop.

It is possible to choose non-physical values for pmpd parameters. For example, you can set

damping to a negative value, which means the creation of energy. This is not physical and can lead to instability or saturation of the model; nevertheless, it can be useful for artistic reasons.

Use of autonomous structures for real time control over audio synthesis

Controllers are physical devices used to send information to a computer. A controller can be a joystick, a keyboard, a midi fader box or any other human interface device which sends information such as the position of a fader, a key press, or any kind of data coming from sensors. Mapping between the control parameters and the audio synthesis algorithm parameters is an important part of the "instrument".

One of the applications of physical modeling simulation is to create a mapping between a musician and the audio synthesis. A dynamic structure modified, moved, and distorted by the user can then be used to control audio synthesis. With the help of sensors, a user can create a virtual structure linked to his or her movements, to the real. The user can then play with a virtual yet "physical" instrument, allowing a natural comportment to digital audio synthesis.

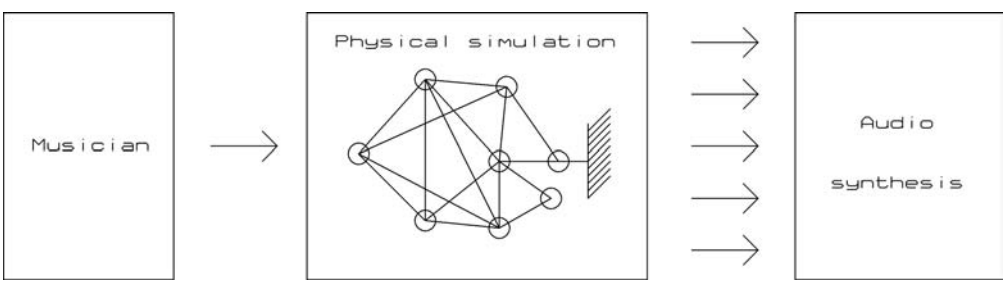


Figure 8. Using physical modeling between the musician and audio synthesis

Figure 8 shows a physical model of a structure used for the mapping of user action and audio synthesis. Such mapping has interesting specifications. For example, a few input parameters can generate many different data flows (a musician can play with only a few control parameters on the whole structure and then generate lots of data to control any audio synthesis). Moreover, the control parameters are intuitive because they correspond to physical values. Playing with such a system can be very intuitive. Certain control parameters can change the way the structure evolves within a time period, allowing the control of the synthesis evolution over time. Another important specification is that all data coming out of the physical model are not independent. The relation between them can be adjusted in regard to the topology of the structure.

Using data generation from a physical model shows the relative importance of audio synthesis.

For example, using simple additive synthesis as in Figure 9 can produce very different sounds depending on the shape and the physical parameters of the structure.

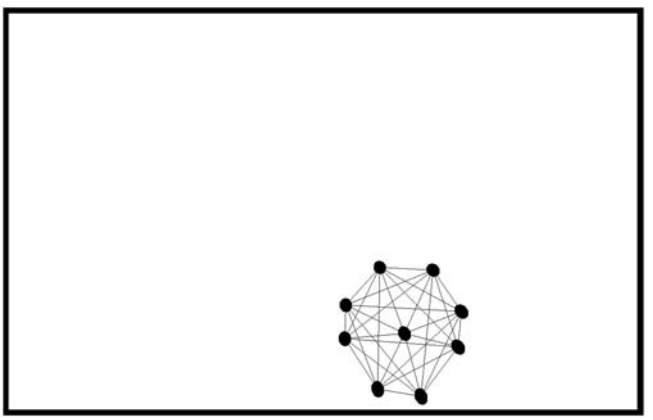


Figure 9. Bouncing ball used for additive synthesis

In Figure 9, forces applied to each mass control the amplitude of the sinusoid which performs additive synthesis. The sound produced by the structure can be controlled while moving the structure, making it bounce, etc. The evolution of the sound over time (the musical structure) can be run with a few parameters that describe the global behavior of the structure, such as rigidity or dampening of link. The physical model offers a user-friendly yet rich and complex interaction with the musical instrument.

A simple physical model can generate a non-linear interaction: the system will react in different ways to the same input solicitation depending on the state (position/velocity) before this solicitation.

Figure 10. Non-linear system

Figure 10 represents time evolution of X, Y and Z forces of a single mass linked to a fixed point. The same force is sent to the masses in the X direction three times. The displacement depends on the position of the mass when the force is sent. In this example, a sound is produced with an additive synthesis: the amplitude of three sinusoids depends on the force over the link in X, Y, or Z. This instrument offers a rich playing technique due to its behaviors.

Linking this patch to a force feedback joystick permits the user to feel the force it sends to this virtual mass, allowing for more precise control over the synthesis, like with a real (physical) instrument.

Another example of using a physical object to control audio synthesis is scan synthesis, where the shape of a structure is used as an audio waveform.

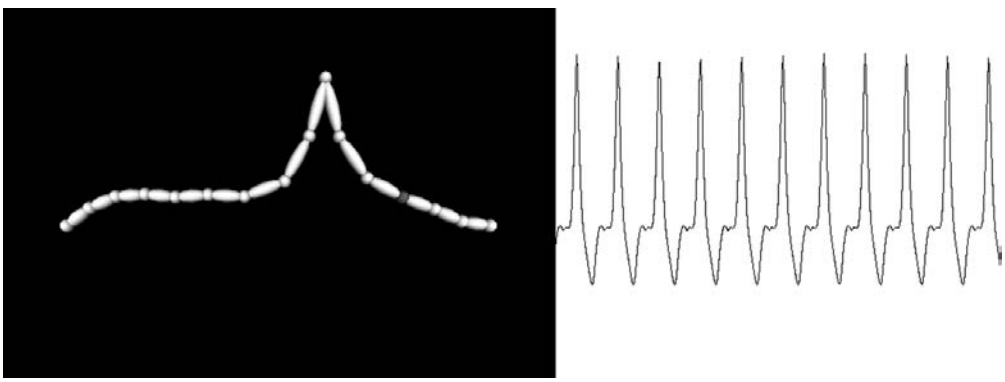


Figure 11. Example of scan synthesis

In the example in Figure 11, the user can interact with an elastic string. This string moves too slowly (only a few hertz) to generate audio sound, yet its shape is recorded in an array that can be looped at audio rate.

Examples

pmpd has been used to model many different kinds of physical phenomena such as sand, fluid mechanics, and elastic objects. It is also possible to create the behavior of non-physical objects, or objects that cannot be physically built. In fact, structures can be made depending on the kind of behavior you wish. For example, Figure 12 is a snapshot of the "Threads" performance by Ben Bogart where each "word" is a physical modeling structure.

pmpd is not the only set of physical modeling objects for Pure Data. The msd object suite allows the creation of structures inside a single object. msd objects are more optimized than pmpd, but less intuitive.

Figure 12. "Threads" by Ben Bogart

Figure 13. Simulation of the movement of a tree and its leaves

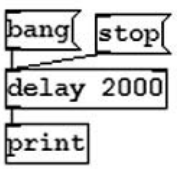
Figure 13 is an example of a simulation of a tree and its leaves. This simulation was made with a Lindenmayer system and the [msd3D] object.

Conclusion

Altogether, only a few simple equations are needed for a complex physical simulation. However, computers deal with digital values at a discrete time, unlike the physical rules that deal with analogue values in continuous time. An approximation must then be found to allow for realistic simulations.

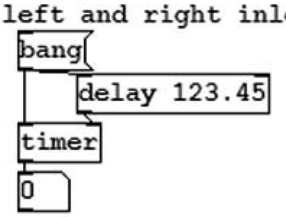
pmpd offers objects for basic physical modeling simulations, and its integration with pure data allows for its use in a wide range of different applications. Many applications still have to be explored, for example haptic devices, which use the sense of touch to better control electronic instruments, and physical models in more than three dimensions. It is hardly possible to represent such a space, but it is possible after all to make "physical" simulations of masses and links.

Besides the metronome, there are three objects for dealing with time:

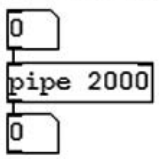


The delay objects sechedules an event for a future time expressed in milliseconds. Unlike in Max, time values need not be integers. If a delay has been scheduled and you "bang" it again, it is rescheduled (the previously scheduled output is cancelled.) The right inlet can be used to set the time value without scheduling any output.

The timer, shown below, measures the time elapsed between its left and right inlets:



Note that all time calculations are idealized; they do not show the effects of computation time or OS latency. This way you can write deterministic algorithms dealing with time passage.



The pipe object allocates memory dynamically in order to schedule any number of delayed events. The events may hold any collection of data (as usual, for more details you can consult the help window.)

updated for Pd version 0.34

Pure words – (Ab)using Pd for Text-Based Score Generation

Thomas Grill

Writing a chapter for a book involves taking suitable words, forming sentences, and considering various formulations so that the desired content gradually comes into existence. This is related to writing a musical score, where following some pattern, one note follows another. This can be tiresome.

marking_time thinking discursive reflection discursive reflection decide action intuitive idea realize generate structure score music performance represent notion lively time marking_time movement circulating movement circulating movement time marking_time movement circulating movement process movement time onset break chain following associated network weight trigger generate structure pattern structure network connected following pattern parameters score music performance lively human intuitive idea realize action intuitive lively human voice discursive thinking process movement time marking_time

Hence, it might be advantageous to just sketch all relevant rules and relationships and let a machine decide about the details, like when improvising musicians follow the general structure of a score but decide upon the sound microstructure instantaneously. Machines won't get tired of this work; they will just strictly follow the given rules. The result might be boring, since it's probably too well ordered, but on the other hand, one can always take it as pure material, rearrange parts, introduce faults and thus make it more lively and human.

movement process performance represent notion lively time onset break gradually break gradually time onset break gradually break gradually break gradually time marking_time movement time boring lively time onset break chain following associated following pattern structure patch chain following pattern structure pattern parameters score music performance represent notion lively human tasteful music performance represent notion parameters score music performance lively human intuitive lively time boring lively human voice discursive notion lively human intuitive lively time marking_time movement

Not too long ago I was asked to write a score for clarinet, viola, cello, and narrator upon the subject of the anti-capitalist fairy-tale "Das kalte Herz" ("The cold heart") by Wilhelm Hauff. Written in the first half of the 19th century at the climax of romanticism, the language of the narration wasn't really what I considered suitable for contemporary music and on-stage recital. Nonetheless, the ideas and notions used in the tale were interesting and timeless. Therefore, I decided to understand the composition as a reflection of the essence of the tale rather than of the story itself, like when thinking of reasoning as a process as opposed to an actual physical movement.

notion parameters score music performance represent notion lively time onset break chain following pattern boring lively time onset break chain following associated network weight trigger generate structure network weight trigger action intuitive lively time onset break chain following pattern structure network weight trigger action intuitive idea music performance represent notion parameters score music performance represent notion

The idea was that the musicians and the narrator would stand still in a musical sense, just marking time, circulating the same few ideas over and over again but on different trajectories, therefore being independent in principle but meeting many times in various constellations in the course of half an hour of performance. To underline the idea of physical immobility with highly discursive activity, I would let the musicians play in an atmosphere of traffic and people walking, played from tape, so that the instruments and voice would often not even be audible, masked by the noise of vehicles passing by. The audience should notice within the first few minutes that no action is taking place and therefore calm down and take part in the reflective process.

score music performance lively human intuitive lively human intuitive idea realize generate structure score music performance lively time marking_time movement process movement time boring lively time onset break chain following pattern parameters score music performance represent notion lively human voice discursive thinking process performance represent notion parameters score

When considering how to realize the structure of the piece without being too tasteful in a musical sense, or simply too human, I thought of the patch structure of Pd, which with its objects is able to represent various ideas chosen from the tale, each one connected to a specific musical expression, and where the interconnections of objects represent semantic relationships between the several ideas. The patch would then form a semantic network, where each object in the patch is able to trigger in turn another associated one and gradually make up the material for a score.

time marking_time movement time marking_time movement process movement process performance lively human intuitive lively time boring lively human voice discursive reflection discursive decide action intuitive lively time marking_time movement process movement time boring lively human voice discursive thinking discursive reflection discursive realize action intuitive lively human intuitive lively time onset break chain following associated following associated network connected following associated following pattern parameters score music performance represent notion lively time onset break onset break chain following associated following associated network weight trigger action intuitive lively time

This representation of interdependencies is also known as a Markov chain[1], which is commonly displayed as a table of probabilities between associated elements. The advantage of the patch representation is that the interconnections can be set up and tuned much more intuitively than by filling numbers into a table. On the other hand, it has the drawback that associations can hardly be generated automatically, which is also a widespread technique. For the given application, however, it's just fine.

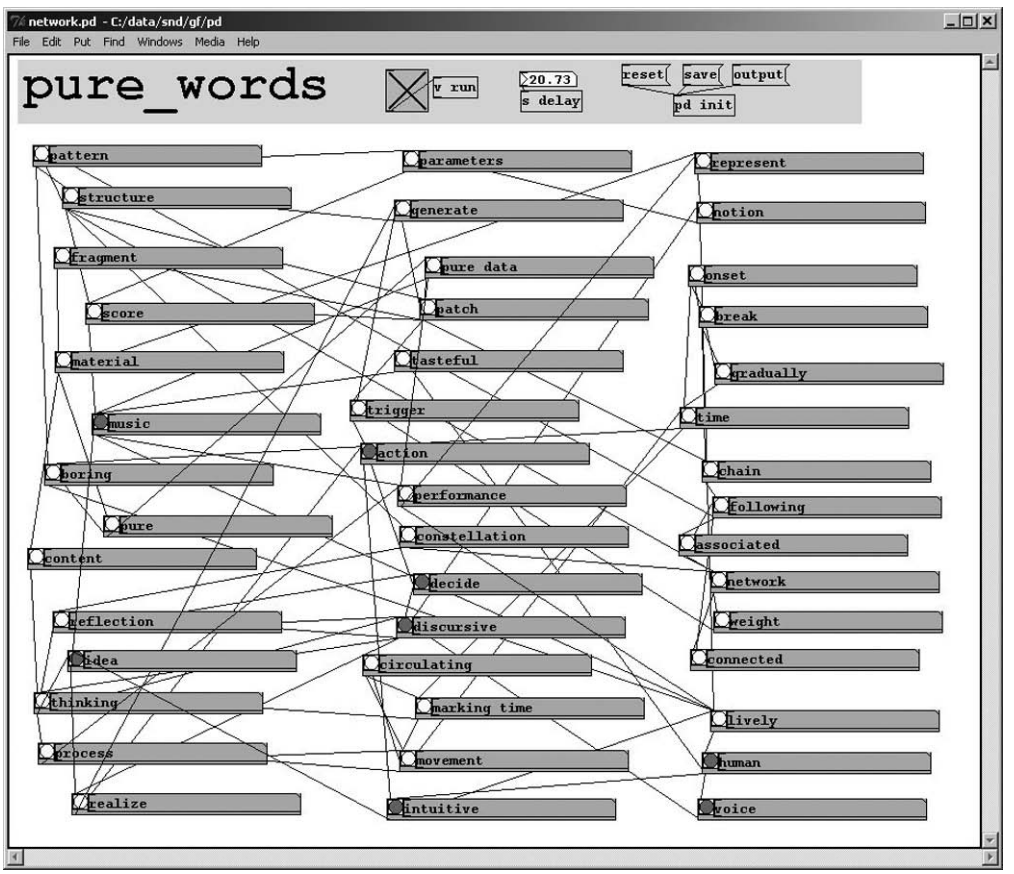
discursive notion lively time boring lively time boring lively time onset break chain following associated network weight trigger action intuitive idea realize generate structure pattern boring lively human tasteful music performance lively time marking_time movement time boring lively human tasteful music performance lively time marking_time movement process movement circulating movement time boring lively human voice discursive

There are two main difficulties when using the patch structure of Pd for this kind of network.

1 <http://en.wikipedia.org/wiki/Markov_chain>

First, it should be possible to connect several downstream elements to one object, but only one of them should be triggered to pass the relay on to the next elements. Second, it should be possible to weight the connections so that one association can be stronger than another one. Both issues can be mastered with some tricky patching. Although it's not strictly necessary, I decided to use py/pyext[2] for reading in the word database, for constructing the score as a list notion by notion, and for saving the score to a file. The advantages of using py/pyext instead of common Pd objects are that it's very convenient and makes for quick coding[3].

reflection decide action intuitive lively time onset break chain following associated following pattern structure score music performance lively human voice discursive realize action intuitive idea realize action intuitive lively human intuitive lively time boring lively human intuitive idea music performance lively time onset break chain following associated following associated network connected following pattern structure patch generate structure pattern boring lively time onset break chain following associated network connected following pattern structure constellation reflection



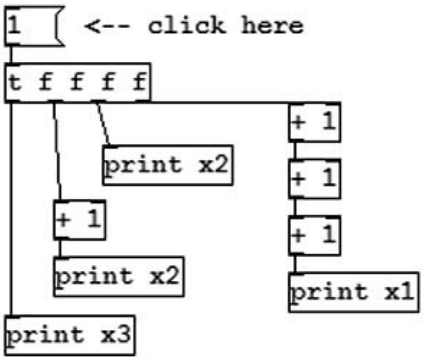
2 <http://grrrr.org/ext/py>

3 The respective patches can be downloaded at <http://grrrr.org/ext/patches/purewords>.

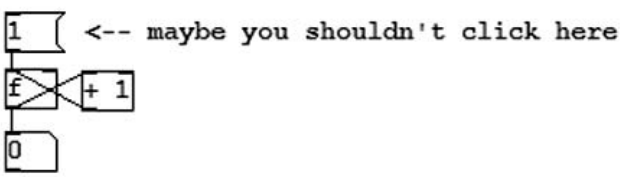
Finally, the process of automated thinking – a path of associations defined by a complex network of selected ideas and their interdependencies – will generate a bunch of text. I associated six more or less orthogonal musical parameters – such as energy, purity, duration, variety, brightness, opposition – with each of the ideas. The fact that these parameters of one idea in the score will in many cases be not too different from the preceding ones creates a kind of smooth envelope over time. On the other hand, rapid changes in one or more parameters seldom can be used to mark the time structure in the score. Manual filtering and reviewing the resulting gestures are indispensable and finally render the structure to music – with three instrumental voices this gives a quite complex pattern of autonomous or concerted movements, of onsets and breaks, accompanied by associative text fragments recited by the narrator[4].

[4] "sich über sich" was performed on June 29, 2005 at Grabenfesttage, Vienna with Gerald Preinfalk (clarinets), Petra Ackermann (viola), Andreas Lindenbaum (cello) and Johannes Poigenfürst (voice).

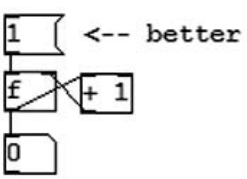
In Pd, message passing is depth first, so that in this patch:



... you get "x1" first, notwidthstanding the fact that "x2" and "x3" appear to be closer to the source. This means that you shouldn't do this:



... because the "depth" is infinite. The counters you've seen always have the message chain terminated somewhere in a cold inlet:



updated for Pd version 0.34

Why Do People Develop Free Software?

Susanne Schmidt

Every year around the time of German finals at school, I get some emails concerning a howto I wrote a few years ago. It is a short "anti-slacking howto for young nerds", written after my experiences working as a project manager, when I saw some young nerds and geeks quitting school due to the enticements of the "new economy" and its job offerings. The emails I get are (mostly) grateful and friendly. Every time I read one of them, I think: "Well, it was worthwhile writing it." If I ask myself why I published the howto instead of just emailing it to friends who could possibly benefit from some hints against slacking around, well – I don't know. I never even considered anything other than publishing it. After 12 years with Linux and Open Source software I have become very used to the idea of publishing for free and for everyone. Shareware? Freeware? Well, why bother with half of the cake, if I can have all of it – the complete source? The phenomenon of Open Source has crept into everyone's computer and Internet behavior: It is not just software and source code, published freely, it is also projects like Wikipedia and more and more scientists publishing their research results and theories, making them available to everyone on the net. The main issue here is familiarity with the idea of creating Open Source software and sharing it with others. Why do people do that?

It is obvious that people need and use Open Source – not just a small howto, but documentation, information, raw data, software, icons and pictures, algorithms, patterns, sounds and music, and they like it, value it and sometimes they are even truly grateful. This is very motivating. We sometimes call it "fame and glory through Open Source" – more in an ironic sense, but all the developers I have ever met are rather flattered if someone tells them their software really is used and valued. It is also sometimes amazing to see what people do and create with the developer's software – especially if your software facilitates a very immediate, direct experience like music applications, sound, graphics, pictures or movies do. You might say that one motivation could be to make users happy. Or, in dry, passionless words: "In the world of Open Source software development, actors have one more degree of freedom in the proactive shaping and modification of technologies, both in terms of design and use."[1]

Why do developers spend hours, weeks, months and even years programming a piece of Open Source software in their spare time? Well, it's fun. You may call it more sophisticated "intrinsic motives" and wrap an intimidating theory all around it, but there is the pure joy of craftsmanship. Just as I like – maybe love – writing, they have a passion for hacking and coding. It is satisfying to look at a final piece of code, checked in into the source code repository, and see peo-

[1] Brent K. Jesiek, "Democratizing Software: Open Source, the Hacker Ethic, and Beyond", First Monday, Volume 8, Number 10 (October 2003), <http://firstmonday.org/issues/issue8_10/jesiek/index.html>

ple using and adopting it. Even if people start to nitpick about your style, your code or your programming language of choice, you can still remain happy with the project. A student of graphical design took my howto one day and made a print version to be distributed freely as a students' brochure at the university. In this way, the effort that one person makes may lead to an interesting project or idea from another person – we all are happy, if someone actually uses our library, our patch or our piece of code and finds something useful to do with it. For my part, I was rather proud and flattered – who wouldn't be? The satisfaction of craftsmanship and puritanical, Protestant "usefulness" can be very forceful: making things work, because you can, making things work even better, because you give all your ability and perfectionism into it – that can be very rewarding. Having hacked a small tool or having written a nice piece of documentation feels the same way as looking onto a freshly painted wall in your apartment: one can see one's work and one can see if it is well done. Sometimes, writing Open Source software is truly a labor of love, and in a way it is the geek's way to express political engagement rather than walking a protest march through Washington – Berlin – Paris – Tokyo.

The second reason is the insistent, nagging feeling of "But someone has to!". Dissatisfaction with existing projects, sometimes the unaffordable price of commercial software for a small library one needs right now, badly written software crashing and leaking your memory, the non-existence of something that would be really useful or is needed for another project – these are just a handful of reasons why developers spend so much time on Open Source software. All this may lead to an effort of "ok, I will just do it now", and it can be rather simple: If you don't start with it, maybe no one ever will. The Italian economists Andrea Bonaccorsi and Cristina Rossi just put it in economic terms of "incentives" and filling a gap in the market.[2]

Sometimes it is the sheer lack of a tool you need that leads to a new project. Sometimes it is the vast of existing tools that another operating system offers, but yours does not, and sometimes it is the poor quality of an existing piece of software that leads to many weeks with short nights and bad food – and a new tool. Sometimes it may even be a feeling (or simply imagining) that "I can do better!" that results in a new project.

The amazing thing with Open Source software is that if you have the ability to program, in a way the world is yours. You may compare it to someone who is a great manager and has the skills to organize and manage huge exhibitions or establish a political movement. With the development of software, we have the ability to change circumstances and environments – at least from time to time. Sometimes, the idea of making software Open Source can be very powerful – just look at projects like the former StarOffice or Netscape, which changed into Mozilla which leads us to Firefox. See the revolving influence of any peer-to-peer software and the fights around DeCSS, and you get an idea of what is possible today with the right software project.

---

2 Andrea Bonacorsi and Cristina Rossi, "Altruistic individuals, selfish firms? The structure of motivation in Open Source software." First Monday, Volume 9, Number 1 (January 2004), <http://firstmonday.org/issues/issue9_1/bonaccorsi/index.html>

The tendency of some industries to consider their customers as too stupid to handle a tool of any complexity often hampers the capabilities a program could have, and suddenly you are limited solely by marketing decisions in what you can do with your application and what is no longer possible. Years ago, I found an interesting marketing study somewhere on the web about "Lego" and why there is so much "theme-oriented" Lego, like Star Wars-Lego, Lord of the Rings-Lego, Harry Potter-Lego and Biff and Bim-Lego, and not much basic Lego bricks anymore, as I remember from my childhood: Because "the customer is too overwhelmed by the flexible choices and isn't able to develop the creativity needed to build really cool things". In other words, your spaceship does not look as cool as that stylish "bird of prey" superfighter on the wrapper and never will. This is exactly the same for many applications, which are intended to be "more smart" than the user, "make things convenient" for the user, and don't make it too difficult for me! I'm not talking about a very well developed user interface to improve the handling of application X – by all means, using it should be as fast and easy as possible – I am talking about hiding capabilities. The price one pays is always vanishing capabilities and the disappearance of flexibility in the application: what you can do is suddenly limited by design and industry, not by your ideas and – let's say – physical laws. There is the wonderful German term of "Volksverdummung" (brainwashing of the people) which applies perfectly here. Or, even worse, software overzealously hides certain capabilities due to possible copyright frauds or legal problems somewhere in some country on this planet. Don't mess with business.

Open Source software may give you (back) that freedom and choice. That is another reason why people hack Open Source. One may argue that this freedom is a liberty for a few initiates, those developers who are capable of handling complex and demanding applications – but it is the principle that counts here. In principle, you can change the code. I still agree with Eric Raymond's optimistic view on Open Source, even if some people see a barrier for "ordinary people" because of the complexity of large Open Source projects: Nikolai Bezroukov wrote in "First Monday" that "[...] Open Source, in Raymond's view, is just source code, not all of the complex infrastructure and implicit knowledge that are used in large software projects."[3] Nevertheless, in principle, you can build any kind of application on your own to suit your personal needs. It is like arguing that certain philosophers can no longer be read, because they are too demanding and customers cannot be expected to learn a foreign language. "Dear citizen, due to the fact that Marx is too demanding to understand, and his ideas are not compatible with the existence of the Department of Trade and may lead to license and copyright fraud and can cause a revolution, and you cannot be expected to consider learning German, we offer you our new product PaterNal 2.0, now with a superior decisionless graphical interface!" The steep learning curve of software sometimes asks for much effort on the part of the user, but most of the time the effort leads to a deeper knowledge and understanding of the inner world of computer software – an ability much needed and very useful in a thoroughly networked

[3] Nikolai Bezroukov, "Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)", First Monday, Volume 4, Number 10 (October 1999), <http://firstmonday.org/issues/issue4_10/bezroukov/index.html>

and computerized society. As Open Source developers seek to make their project the best one in the sense of what the application can do, rather than in terms of being easy to use, many projects in turn give users the flexibility they demand. An Open Source developer is not required to meet the demands of a marketing department, and that can be very liberating.

Nevertheless, Open Source is not healing the planet and it is not happy faces everywhere. If one takes a look at Sourceforge or Freshmeat, it sometimes looks like a huge graveyard with tombstones of alpha-versions dedicated to unfinished and abandoned software projects. One may also ask if we really need 36 web servers (search string "httpd") instead of developing four or five for different purposes and scenarios. Freshmeat even lists over 1000 projects with the search string "editor" – well, if that isn't freedom of choice! Things have changed during the last 12 years and so has the type of personality of the developers programming Open Source software. Today, even 10-year-olds have the possibilities and technical resources to start developing a new operating system and can rely on the large amount of code examples available on the Internet. Open Source software today has some tendency to be focused more on cute, neat and funny things, rather than on solving boring or difficult problems or bothering with necessary clean-ups of some projects. This is the other side of the coin that everyone can write code they like – it is not that easy to convince a young developer to engage in projects that are more necessary than "cool". Many developers of the first generation of Linux tools, Internet applications or device drivers are disappointed, over-worked or simply have other things to do. Therefore, the world of Open Source is changing rapidly.

Open Source software development is also still a male-dominated business, I am sad to say – and this is not due to the fact that an Open Source project is not the ideal environment for curious girls to take their first lessons in computer programming. Even with all the possibilities and freedom and access for everyone, women still are a very small minority in the free software business. I can offer no explanation for that phenomenon – obviously even the impact of Open Source software and the public recognition it has reached in the last decade is still not enough to attract (many) women. Here and there we can see a few female developers, and I'm glad to see them, but we are still far away from taking half of the cake.

Yet even though the history of free software and the last decade have been very bright and promising, things are changing rapidly, and there are some clouds in the sunny sky of free software: software patents, for example, just to mention one problem that can throw projects into serious trouble with one legal stroke. One might remember the legal issues of distributing strong encryption - we now face legal issues forcing programmers to find a way to creatively bypass patented graphical algorithms or mechanisms to perform a search. It is not just the Department of Defense giving computer enthusiasts a hard time "in times of terror" – it is the movie and music industry and the World Trade Organization's demands that developers face today.

The cultural battle in the software business is fought along the lines of legal issues of Digital Rights Management, software patents, privacy and civil rights. Now and then, on an IRC-

channel, you can more and more often hear a developer talking quietly about not releasing code or hesitating to publish an idea, because he is afraid of legal issues. And this is not the bright future where young developers may eagerly start new Open Source software projects that the world still needs.

Further Readings:

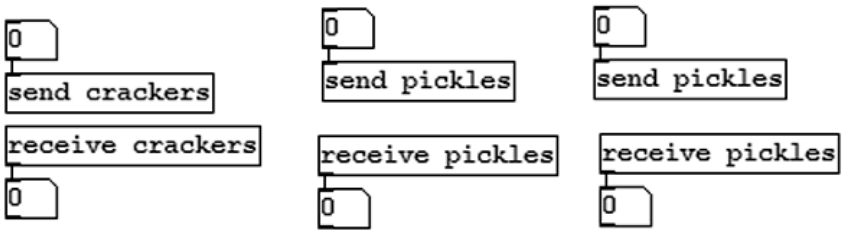
Eric Raymond. The Cathedral and the Bazaar. 09 Jan. 2006
<http://www.catb.org/~esr/writings/cathedral-bazaar/>.

Eric Raymond. Homesteading the Noosphere. 09 Jan. 2006
<http://www.catb.org/~esr/writings/cathedral-bazaar/>.

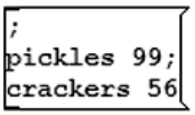
Bruce Sterling. A Contrarian View of Open Source. 09 Jan. 2006
<http://www.oreillynet.com/pub/a/network/2002/08/05/sterling.html>

Michelle Levesque. Fundamental Issues with open source software development. 09 Jan. 2006
<http://www.firstmonday.org/issues/issue9_4/levesque/>

The send and receive objects allow you to make non-local connections. These work globally--you can use them to make two different patches intercommunicate if you wish. Any message a "send" gets appears at the output of every receive of the same name. There can be any number of sends and receives sharing the same name:

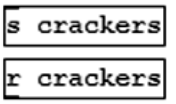


You can use the semicolon feature of message boxes to address receives, too. This is useful if you want to do a whole list of things:



The transaction takes place in zero time---i.e., if you tried to use "timer" to measure the time delay between the two, you would get zero.

send and receive can be abbreviated:



updated for Pd version 0.34

Building Your Own Instrument with Pd

Hans-Christoph Steiner

1. The Problems with Live Computer Music

With the power that even a cheap laptop can provide, the computer has gained widespread acceptance as musical tool. Composers have been creating music using computers for more than 40 years now, and even music created with analog instruments is generally recorded and mixed on computers. More and more musicians are using computer-based instruments for live performance, to the extent where you can see live computer music in just about any major city in the world. There is a wide range of music software specifically designed for live performance, such as GDAM[5], Ableton Live[1], SuperCollider[9], Max/MSP[11], and Pd. Although these tools can provide an engaging performance environment, the live performance leaves something to be desired and is often indistinguishable from someone reading their email. Such performance lacks physicality in the interaction and is quite limited in the range of possible gestures.

Audio synthesis has freed instrument designers from the constraints of the physical method of generating sound, thus any interface can be mapped to any synthesis algorithm. For example, a guitar's strings are both the interface and the sound generator, while a MIDI keyboard can control any MIDI synthesizer. This flexibility allows musical instrument designers to choose their interface without the constraints of the method of sound generation; the interface need not even be physical. Consequently, a multitude of means of translating gestural input from the human body are readily available. By combining such gestural input methods with Pd, a broad range of people can now make their own gestural instruments.

Yet almost all computer musicians have bound themselves to the standard keyboard/mouse/monitor interaction model. To provide an engaging performance, musicians need to move beyond what the basic laptop offers. The human body is capable of a great range of gestures, large and small, communicating emotion in a manner similar to music. There are many distinct and some universal human gestures that are well established and easily understood. Since music is about expressing and communicating, using a broader range of gesture enables the performer to have a broader range of expression. Computer musicians should not be limited to the small set of gestures that a normal computer interface can capture. In order to expand the musician's interaction with the computer, other input devices are needed. Many music software environments are already capable of using data from Human Interface Devices (HIDs) such as joysticks, drawing tablets, gamepads, and mice.

Figure 1: "The gesture is embedded in the music"[10] (Blue Vitriol [3], (c) Patina Mendez; Hazard County Girls, (c) Larry Stern)

Performers of live computer music generally stare at the screen intently while performing, rather than interacting with the audience. What the performer is staring at is obviously important, judging by the intensity of the stare. However, what the performer is looking at is out of view for the audience. This is in stark contrast to traditional musical instruments, where the instrument is generally in plain view, and the audience is familiar with the mechanisms of that instrument. The absense of these two qualities further alienates the performer from the audience. This alienation can be alleviated when the musician can perform using an expanded range of gestures. The performer can come out from behind the computer screen, bringing back a closer connection between audience and performer. Michel Waisvisz's "The Hands" is a great example of such an interface. Built in the early eighties, it has been used to control a variety of different sound synthesis schemes. It is a novel interface that he has played for 20 years, achieving virtuosity. It allows him to stand on stage with nothing but "The Hands" and use gestures large and small to control sound and compose in realtime.

Haptic feedback is another element that is missing from the keyboard/mouse/monitor interaction in contrast to traditional musical instruments. "Haptic" means "relating to the sense of

touch to the skin and the sense of forces to the muscles and joints". Traditional instruments provide haptic feedback because the interface is producing the sound itself, so the vibrations can be directly felt. Practiced musicians rely heavily on this feedback, often correcting mistakes by feel before hearing them. During performance, computer musicians are obviously using feedback beyond just listening to the music, the intensity of their stare at the computer screen is a measure of this. They are relying on visual feedback that the software provides via the screen. Providing haptic feedback means the performer can rely less on the visual feedback and instead engage the audience.

Pd is a fertile platform for creating live musical performances in an environment that is accessible to a wide range of people with varying skill levels. It is a unified platform for a broad range of activities, combining realtime synthesis and manipulation of both audio and video, physical modeling, and more. Pd provides many options for data input/output including MIDI, networking, USB HID, and general serial communications. Since Pd is free software that runs on most operating systems, even musicians with very limited budgets can build their own computer music instruments. Up until recently, computer music has been out of reach to all but a select few. It is now possible to create an instrument using Pd that costs less than most traditional musical instruments, including the cost of the computer.

2. The Fundamental Breakdown

When approaching instrument design, the overall problem can be broken down into input, output, mapping, and feedback. The idea of input is straightforward: the data used to control the instrument. Output is also a simple concept: the desired result of playing the instrument. Mapping is a more complex idea: the processing and connecting of input data to parameters which control output. And last but not least, feedback is communication generated from the input, output, and/or mapping data.

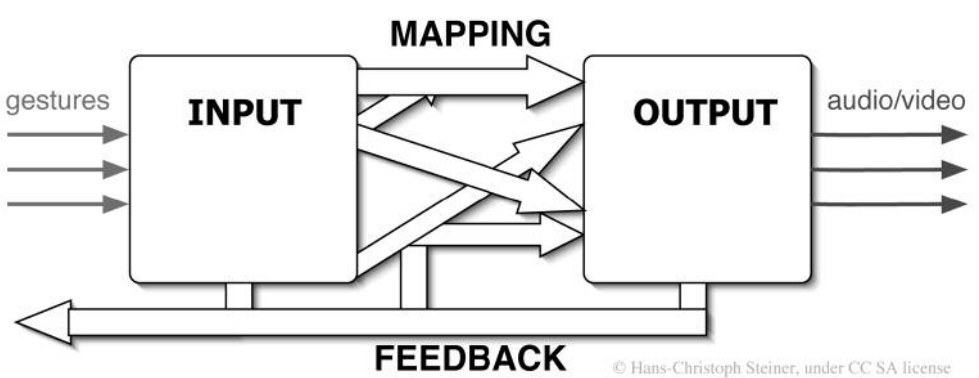


Figure 2: The fundamental breakdown of instrument design.

3. Getting Input Data

3.1. Human Interface Devices

When talking about interacting with computers, "HID" has become the standard term for devices designed to control some aspect of a computer. A wide range of devices are classified as HIDs, including standard computer devices like mice and keyboards, as well as gaming devices like joysticks and gamepads, to devices for more specific needs like drawing tablets. Pd now has a unified method for getting data from HIDs: the [hid] toolkit. Consumer input devices like mice and graphics tablets are affordable and readily available and have a lot of potential as musical controllers. They are familiar to most contemporary concert audiences. Using HIDs in performance therefore has the potential for making the experience much more understandable to the audience. There are a number of examples of contemporary musicians who have mastered using a standard HID as a musical controller: Gerard Van Dongen tours with his Saitek force feedback joystick[12] controlling Pd; Hans-Christoph Steiner has performed live with StickMusic[17], built using a force-feedback joystick and mouse.

3.2 Sensors and Microcontrollers

There is a huge variety of sensors, switches, buttons, displays, and electronic devices readily available, from force-sensitive resistors to accelerometers to infrared proximity sensors. Building from individual parts allows the designer to tailor the controller closely to his desires. Recently, there has been a surge in the development of various sensor boxes which allow users to easily get data from various sensors into their computers. The MultI/O-Box[8] is the easiest way to use arbitrary devices for input, converting sensor data to USB HID and MIDI. Such sensor boxes convert analog signals to digital signals, making them very easy to use within Pd. Microcontrollers such as the Microchip PIC[7] or the Atmel AVR[2] have become a popular method of getting sensor data into computers. They are cheap and run fast enough to track the output of an array of sensors. The downside is that a solid knowledge of electronics is needed to create reliable instruments. Also, many microcontrollers are too slow to provide good resolution.

3.3. MIDI Equipment

A wide variety of controllers use MIDI to communicate. MIDI guitars and breath controllers emulate traditional instruments but are usually a poor facsimile. The Kaos Pad[6] is a more esoteric controller, which can be used within Pd with Derek Holzer's Kaos Tools[14]. There are many variations of the mixing board, known as MIDI "control surfaces", which provide anything from rows of basic sliders to large consoles with sliders, knobs, buttons, etc. They generally are reliable and designed for musical applications, making them a natural choice for a musical controller. Nick Fells uses MIDI control surfaces in a number of different instruments.

His pieces "Words on the streets are these" and "Still Life"[13] are two examples. He uses the Peavey PC1600x control surface, mapping each slider to various parameters to be directly controlled in realtime. Since the roots of Pd lie in MIDI, it is very well supported. MIDI devices are generally low latency, but the MIDI protocol itself is designed around 7-bit resolution with some 14-bit resolution devices. 7-bit is a quite limited range for a musical controller, especially compared to other devices like USB tablets and mice.

3.4. Video

Computers that can do heavy video and graphics processing are now quite common. This opens up the visual dimension to the musician in a whole new way. Motion, color, and blob tracking using video processing allows all sorts of interactions that previously would have been quite expensive and difficult to implement. By extracting data from live video streams, a wide range of gestures can be captured and mapped as the instrument designer sees fit. There are three key methods of tracking gestures with video: color, motion, and shape. The most common one is using motion detection. With Gem, you can use [pix_movement] in combination with [pix_blob]; PDP provides [pdp_mgrid], which is grid-based motion tracking; GridFlow provides motion detection by subtracting the previous frame from current frame using [@-]. For color and shape tracking, PDP provides [pdp_ctrack] and [pdp_shape] respectively. Another option is to process the visual data using outside software and feed that data into Pd. reacTable[16] takes that approach, using OSC to communicate between the two pieces of software.

4. Mapping

In the same way digital synthesis has freed instrument design from the constraints of the interface generating the sound, instrument designers are also free to design the mapping between the interface and the synthesis separately from the design of the input and the output. Thus any arbitrary interface can be mapped to any given synthesis algorithm; indeed the mapping can also be designed to suit the goals of the designer[15]. Most input devices produce linear data, but mappings in expressive instruments are rarely linear. Sensors often have arbitrary curves which don't make sense in the context of a given instrument. More complex mappings usually create more engaging instruments. There are many common ideas that are frequently used when designing a mapping. For example, since humans perceive loudness and pitch on a logarithmic scale, the amplitude and frequency control data are generally mapped to a logarithmic scale as well. In most compelling instruments, the mappings are not one-to-one between input and output. Certain input parameters usually affect more than one output parameter. For example, how a guitarist plucks a string affects both loudness and brightness.

5. Feedback

In standard computer music performance environments, the screen is the sole source of feedback besides the audio itself. The software interface provides visual feedback, usually in a very concrete manner: by displaying the status of various parameters with virtual knobs, sliders, or even just numeric values. Since non-auditory feedback can greatly enhance the interaction of human and computer, such feedback should become a standard part of instrument design. Adding feedback also allows the musician greater control over the output. The feedback can be in the form of haptics, voice alerts, visuals, or even smell or taste, generated from data coming from any part of the system: input, output, or mapping. Since video synthesis and control is part of Pd, generating novel visual feedback could provide more real-time information to the performer while adding another dimension to the performance. By providing haptic feedback through the physical interface, the need to stare at the screen can be alleviated. Haptic devices have become readily available and affordable. There are numerous gaming HIDs, such as joysticks, gamepads, and mice, which can provide a range of haptic 'effects' from vibrations to forces to friction. Since the motor control in these haptic controllers has been encapsulated into haptic 'effects', they are generally quite easy to control. The [hid] toolkit provides a number of objects for generating haptic 'effects' such as [hid_ff_periodic] or [hid_ff_spring]. They follow the same conventions as the rest of the [hid] toolkit, so they should easily interoperate with the whole set of mapping objects.

6. A New Model of Instrument Design

A new model of instrument design is emerging, shifting away from instruments designed for a broad user base, such as the Theremin, the MIDI keyboard, and the vast majority of traditional instruments. Instead many instrument builders are using systems of building blocks that allow them to create their own instrument relatively easily and quickly. This has contributed to a shift in the idea that a musical instrument should be a device for playing a wide range of pieces. Individual musicians can create their own instrument tailored to their performance goals, or even tailor an instrument to a specific piece or performance. One great advantage of the old model of instrument design is that musicians can develop and share a body of knowledge about how to play that instrument. This is something that has been severely lacking in the world of new interfaces for musical expression: it is rare for anyone to achieve virtuosity on these new instruments, even among the designers themselves. Using standard interfaces such as joysticks and tablets allows people to build shared technique without sacrificing the ability to specifically tailor the instrument via the design of the mapping and the output.

7. Gestural Video

This text has been about musicians, since traditionally, gestural instruments have been used to create music. But this is no longer the case: the newfound power of the computer has opened up visual synthesis to gestural control. Some artists are starting to control visuals with gestures, breaking away from the standard on-screen mixing interfaces. A performer can control tightly synchronized audio and visuals generated from a real-time gestural interface. chdh[4] is an example of this kind of work, combining three-dimensional visual elements linked with matching timbres and themes. All of this is controlled in real time using MIDI control surfaces by the two brothers that make up the group. In conclusion, the power that the computer provides makes new forms of musical expression possible and accessible to almost every person. With environments like Pd, the instrument designer no longer needs to spend decades learning the trade, but instead can start experimenting quite rapidly, yet still spend decades perfecting their skills.

References

1 *Ableton Live*. 11 Dec. 2005 <http://www.ableton.com/index.php?main=live>.

2 *Atmel AVR*. 11 Dec. 2005 <http://atmel.com/products/avr/>.

3 *Blue Vitriol*. 11 Dec. 2005 <http://bluevitriol.com>.

4 *chdh*. 11 Dec. 2005 <http://chdh.net>.

5 GDAM. 11 Dec. 2005 <http://gdam.ffem.org>.

6 *Korg Kaoss Pad*. 11 Dec. 2005 <http://www.korg.com/gear/info.asp?A_PROD_NO=KP2>.

7 Microchip PIC. 11 Dec. 2005 <http://microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=74>.

8 *multI/O-Box*. 11 Dec. 2005 <http://multio.mamalala.de/>.

9 *SuperCollider*. 11 Dec. 2005 <http://audiosynth.com>.

10 B. Buxton. Nime05 keynote speech. 2005.

11 Cycling '74. Max/MSP. 11 Dec. 2005 <http://cycling74.com>.

12 Dongen, G.V. 11 Dec. 2005 <http://www.xs4all.nl/~gml/>.

13 Fells, N. *On space, listening and interaction: Words on the streets are these and still life*. Organised Sound 7, no.3 (2002): 287-294.

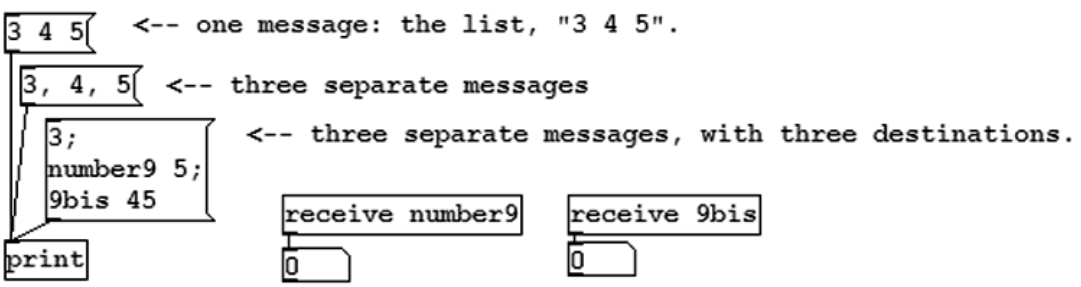
14 Holzer, D. *Kaos tools*. <http://puredata.org/Members/derek/kaos.tools.zip>.

15 Hunt, A. / Wanderley, M. / Paradis, M. *The importance of parameter mapping in electronic instrument design*. Proceedings of the Conference on New Interfaces for Musical Expression (NIME02). Dublin 2002.

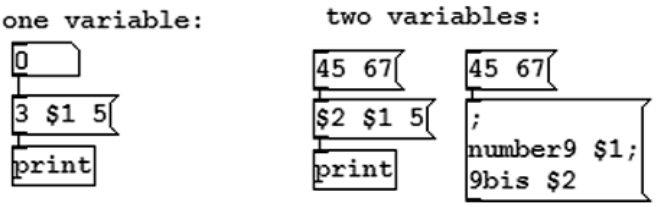
16 Kaltenbrunner, M. / Geiger, G. / Jordà, S. *Dynamic patches for live musical performance*. Proceedings of the Conference on New Interfaces for Musical Expression (NIME04). Hamamatsu, Japan 2004.

17 Steiner, H.-C. *Stickmusic: Using haptic feedback with a phase vocoder*. In Proceedings of the Conference on New Interfaces for Musical Expression (NIME04). Hamamatsu, Japan 2004.

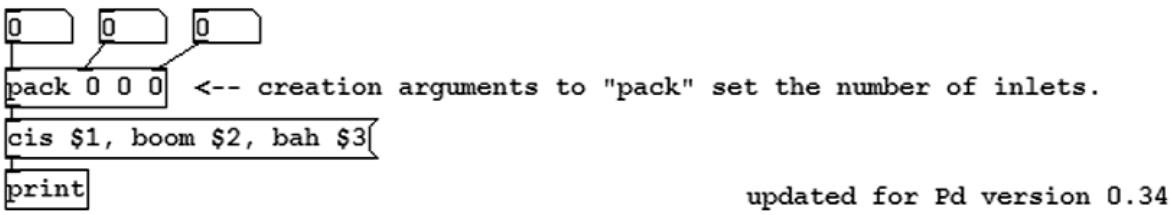
In addition to using semicolons to separate messages, you can use commas, which continue a stream of messages to the same destination. Thus:



You can use "$1", etc., as variables in messages. Send the message box a list whose elements supply the values. A number is just a list with one element.



But to really exploit the possibilities using multiple variables, you will need the "pack" object to get two or more values into the same message:



updated for Pd version 0.34

Music as a Formal Language

Bryan Jurish

Abstract

The main focus of this paper is the characterization of generic musical structure in terms of the apparatus of formal language theory. It is argued that musical competence falls into the same class as natural language with respect to strong generative capacity – the class of *mildly context-sensitive languages* described by Joshi (1985).

1 Introduction

The main focus of this paper is the characterization of generic musical structure in terms of the apparatus of formal language theory. Section 2 briefly describes some relevant aspects of formal language theory. Section 3 introduces that class of mildly context-sensitive languages assumed to contain the natural (spoken) languages, and presents an argument that musical competence also falls into this class.

2 A Brief Tour of Formal Language Theory

Alphabets and Strings

A formal language is simply a set L of (finite) strings[1] over some finite character alphabet Σ: $L \subseteq \Sigma^*$. Candidates for alphabetic characters in the domain of music include rhythmic, tonal, and timbral quanta. Such an inventory of quanta might be extended by additional "empty elements" to encode continuous aspects of musical structure within the context of a finite alphabet, much as the symbolic linguist's toolbox has been extended to include *traces, empty operators*, and other theoretical objects which cannot be directly observed in surface strings.

[1] Traditionally, formal languages are defined in terms of the *free monoid* $\langle \Sigma^*, \circ, \varepsilon \rangle$, where $\circ$ is the concatenation operator and ε represents the empty string.

Grammars and Automata

Any "interesting" language being infinite, it has proved useful to characterize formal languages by means of some well-defined finite specification. Traditionally, both grammars and automata have been used for this purpose. The most general notion of a grammar is usually defined as a 4-tuple $G = \langle V, \Sigma, P, S \rangle$ where:

- V is a finite alphabet, partitioned into disjoint subsets N (nonterminal alphabet) and Σ (terminal alphabet),
- $P \subset V^* \times V^*$ is a finite set of *productions* or *rewrite* rules, usually written as $x \rightarrow y \in P$ for $(x, y) \in P$.
- $S \in N$ is the distinguished start symbol.

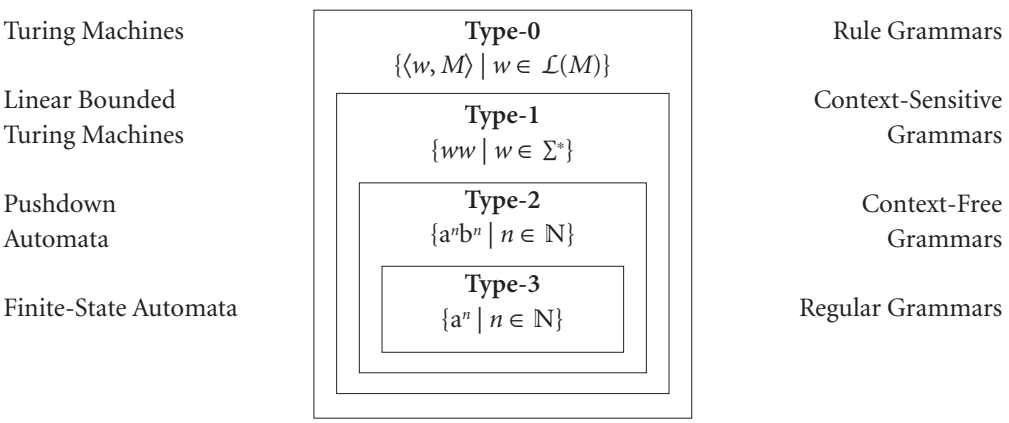


Figure 1: The Chomsky hierarchy of weak generative capacity

A grammar G may be used to specify the formal language $L = \mathcal{L}(G)$ generated by G by means of the *direct derivability* relation $\Rightarrow_G$ for G: $uxv \Rightarrow_G uyv$
iff $u, v, x, y \in V^*$ and $x \rightarrow y \in P$; defining[2] $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$.

Varying restrictions on the format of a grammar's rules give rise to different species of grammar, and has strong implications concerning the computational complexity associated with common language-related tasks such as parsing or generation.

[2] Here, $\Rightarrow_G^*$ is the reflexive and transitive closure of the direct derivability relation $\Rightarrow_G$ for G.

Generative Capacity

The weak generative capacity of a grammar formalism is just the set of string languages which can be described in terms of generation by that formalism. A grammar formalism's *strong generative capacity* on the other hand can be identified with the set of *complete derivations* in that formalism, thus capturing the structural properties encoded by a grammar in addition to its string output. The inclusion hierarchy of traditional grammar formalisms (Chomsky, 1957; Hopcroft and Ullman, 1969) is graphically depicted in Figure 1.

- **Type-3 (Regular Languages)**
  A notational variant of *regular expressions* such as those used by many UNIX utilities, the type-3 languages (regular grammars, finite-state automata) are computationally unproblematic: they are deterministically parseable (in linear time), and support a number of useful algebraic operations including union, concatenation, closure, and even complement and intersection. The major drawback of type-3 languages is their limited generative capacity[3].
- **Type-2 (Context-Free Languages)**
  Type-2 languages exhibit structures which can be described by trees, such as balanced parentheses or "mirror" structures. Most programming languages belong to the deterministically parseable subset of the type-2 languages. The type-2 languages in general are parseable in $O(n^3)$ time, but carry with them a number of sticky problems related to *ambiguity*, and are not closed under complement or intersection.

  Certain aspects of musical structure place musical languages at least in this category – an example is the "mirrored" melodic run in Figure 2(a).

Figure 2: Context-free (a) and non-context-free (b) tone dependencies

- **Type-1 (Context-Sensitive Languages)**
  The context-sensitive languages include the *copy language* (see Figure 1) among others. They are decidable, but are computationally quite complex in the general case. In some cases however, the adequate analysis of musical languages appears to require more structure than a type-2 grammar can provide, as suggested by Figure 2(b).

[3] As anyone who has tried to use `sed`(1) to automate tricky corrections in *C* code knows, regular expressions cannot perform balanced parentheses matching to arbitrary depths.

- **Type-0 (Recursively Enumerable Languages)**
  Type-0 languages are those recognized by some Turing machine (equivalently, those produced by some unrestricted rule grammar), and are usually taken to be identical to the (image of the) set of computable functions. Type-0 formalisms are very powerful, but famously intractable (Turing, 1936).

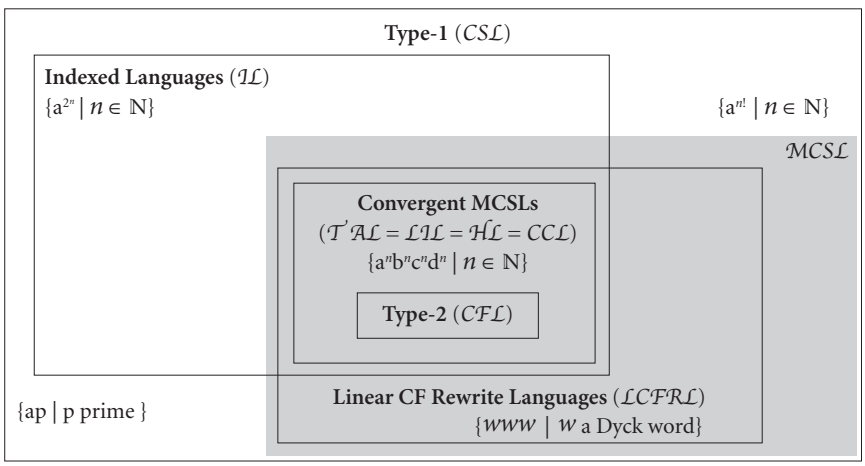


Figure 3: Generative capacity of some (mildly) context sensitive formalisms

3 Mild Context-Sensitivity

Natural languages are known to exhibit some phenomena the description of which goes beyond the weak generative capacity of context-free grammars (Huybregts, 1984; Shieber, 1985). Most prominent among these are *counting dependencies* (of order $m \in \mathbb{N}$) of the form $\{x_1^n x_2^n \ldots x_m^n \mid n \in \mathbb{N}\}$: contextfree (CF) grammars can encode such dependencies only for $m \leq 2$. This is easy to see informally when context-free analysis trees are considered: tree structures cannot have "tangling branches", and thus cannot encode such dependencies.

Joshi (1985) characterized the class of languages to which natural languages are expected to belong as that of the *mildly context-sensitive* languages (MCSL), the inclusion relations between some well-known examples of which are graphically depicted in Figure 3. This class of formal languages is informally identified by the properties of *constant growth*, *polynomial parsing complexity*, and *bounded cross-serial dependencies*, which are discussed individually in the following sections.

Constant Growth

Every known natural (spoken) language has a length-ordering on well-formed sentences for which there is a finite upper bound on the number of words which separate the sentences with respect to that ordering. The condition on constant growth excludes languages such as $\{\mathbf{a}^{2^n}\}$ from the class of MCSLs, thus eliminating Aho's (1968) Indexed Grammars as a candidate formalism.

Although I lack extensive knowledge of the empirical data in the case of musical languages, I think it safe to surmise that these also display the constant growth property. Taking melody again as an example, this means that for a musical language L there must exist a number $j \in \mathbb{N}$ such that for all melodies[4] $M_1, M_2 \in L$ with $|M_1| > |M_2|$ where there is no $M_3 \in L$ with $|M_1| > |M_3| > |M_2|$, then $|M_1| - |M_2| \leq j$. Informally, this amounts to the claim that whenever a melody can be extended (and possibly altered) to produce a longer melody, the maximum length of the shortest such extended melody is fixed by the musical system in question.

Polynomial Parsing Time

Essentially a constraint on computational tractability, MCSL membership for a given input string must be decidable in polynomial time with respect to the length of that string. Rephrased in terms of generation systems, an output string of length n should be derivable in $O(n^k)$ for some $k \in \mathbb{N}$ specific to the grammar (weak form) or MCSL subfamily (strong form) in question.

For practical purposes, it is certainly desirable to require some computational complexity constraint on any formalism used to describe musical structure. Concrete modeling proposals known to me (Xenakis, 1971; Lerdahl and Jackendoff, 1983; Steedman, 1984, 1996; Baker, 1989; Leman, 1989; Bel and Kippen, 1992; Bod, 2001, 2002) do in fact all fulfill this criterion.

Bounded Cross-Serial Dependencies

This condition is clearly motivated by the counting dependency evidence placing natural language beyond the domain of the context-free languages. It requires a limit $m \in \mathbb{N}$ on the number of cross-serial structural dependencies encodable by a given grammar (original, weak form), or grammar formalism (strong form).

It is not at all immediately clear whether such a restriction is desirable for musical languages, especially if phenomena such as theme and variation are to be encoded as real cross-serial

[4] Here, $|M|$ represents the length of M: $|x_1 \ldots x_n| = n$ for $x_1, \ldots, x_n \in \Sigma$.

dependencies rather than pure chance similarities. At the very least, we should reject the strong form of this criterion for musical languages. Such a weak (original) form of Joshi's restriction is compatible with the formal properties of contemporary theories for the description of natural (spoken) languages (Stabler, 1999; Michaelis, 2001).

Common equivalent mildly context-sensitive grammar formalisms include the *Tree Adjoining Languages* ($\mathcal{TAL}$) (Joshi, 1985, 1987), the *Linear Indexed Languages* ($\mathcal{LIL}$) (Gazdar, 1988), the *Head Languages* ($\mathcal{HL}$) (Pollard, 1984), and a restricted form of the *Combinatory Categorial Languages* ($\mathcal{CCL}$) (Steedman, 1990). This family of languages is distinguished by a hard limit of $m \leq 4$ counting dependencies.

Stronger formalisms which provide room for more counting dependencies on a per-grammar basis include the *Divided Index Languages* ($\mathcal{DIL}$) (Staudacher, 1993) and the *Linear Context-Free Rewrite Languages* ($\mathcal{LCFRL}$) (Seki et al., 1991).[5] Michaelis (1999, 2001) showed that the latter are equivalent to Stabler's (Stabler, 1997) formalization of the *Minimalist Grammars* proposed by Chomsky (1995) as a vehicle for the description of natural language.

4 Conclusions and Perspectives

Adopting the working hypothesis that musical structure can be expressed symbolically in terms of a finite alphabet of structural quanta, it becomes possible to model a musical system as a formal language, or set of strings. The generative capacity required for the characterization of musical languages must lie beyond that of the context-free languages, in order to adequately encode the cross-serial dependencies displayed by musical phenomena such as theme and variation. Empirical arguments were presented that musical languages exhibit the characteristic properties of the mildly context-sensitive languages, to which natural (spoken) languages are also assumed to belong.

One property considered important for musical languages which was not discussed above is the possibility of their incremental generation and analysis, also known as the "improvisation property". While empirical evidence exists that spoken language also displays this property, it is often unclear how it may be efficiently implemented for a strong language family such as the MCSLs.

Real-time musical processing environments such as Pd (Puckette, 1996, 2004) oriented toward generation and synthesis are usually Turing-complete (type0), thus opting for strong generative capacity and leaving issues of computational complexity within the user's discretion.

[5] The context-sensitive "pattern grammars" used by Bel and Kippen (1992) to model musical structure appear at first glance to fall into the same category, but a proof of this hypothesis is beyond the scope of the current work.

Analytically motivated approaches (Xenakis, 1971; Steedman, 1984; Bod, 2002) often tend toward the opposite extreme of low computational complexity at the cost of generative capacity.

One way to try and unify these disparate approaches is to use a powerful (but complex) processing tool such as Pd as a vehicle for the instantiation, manipulation, and interpretation of instances of a weaker (but generally efficient) generative mechanism. The most obvious candidate for such a weak embedded mechanism which also exhibits the improvisation property and which is familiar to many users is that of *regular expressions*, a notational variant of the type-3 languages. An abstract C library for regular expressions and weighted finite state machines with a Pd interface is currently under development (Jurish, 2004).

References

A. V. Aho. Indexed grammars. *Journal of the Association for Computing Machinery*, 15:647-671, 1968.

M. Baker. A computational approach to modeling musical grouping structure. *Contemporary Music Review*, 4:311-325, 1989.

B. Bel and J. Kippen. Bol processor grammars. In O. Laske, M. Balaban, and K. Ebcioglu, editors, *Understanding Music with AI – Perspectives on Music Cognition*, pages 366-401. MIT Press, Cambridge, MA, 1992.

R. Bod. A memory-based model for music analysis: Challenging the Gestalt principles. *Journal of New Music Research*, 31(1):26-36, 2001.

R. Bod. A unified model of structural organization in language and music. *Journal of Artificial Intelligence Research*, 17:289-308, 2002.

N. Chomsky. *Syntactic Structures*. Mouton and Co., The Hague, 1957.

N. Chomsky. *The Minimalist Program*. MIT Press, Cambridge, MA, 1995.

G. Gazdar. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69-94. D. Reidel, Dordrecht, 1988.

J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, MA, 1969.

R. Huybregts. The weak inadequacy of context-free phrase structure grammars. In G. J. de Haan, M. Trommelen, and W. Zonneveld, editors, *Van Periferie naar Kern*, pages 81-99. Foris, Dordrecht, 1984.

A. K. Joshi. Tree adjoining grammars: How much context-sensitivity is necessary for characterizing structural descriptions. In D. Dowty, L. Kartunnen, and A. Zwicky, editors, *Natural Language Processing: Theoretical, Computational, and Psychological Perspectives*, pages 206-250. Cambridge University Press, New York, NY, 1985.

A. K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 87-114. John Benjamins, Amsterdam, 1987.

B. Jurish. gfsm finite-state machine library. work in progress, 2004.
URL http://www.ling.uni-potsdam.de/~moocow/projects/gfsm .

M. Leman. Adaptive dynamics of musical listening. *Contemporary Music Review*, 4:347-362, 1989.

F. Lerdahl and R. Jackendoff. *A Generative Theory of Tonal Music*. MIT Press, Cambridge, MA, 1983.

J. Michaelis. *On Formal Properties of minimalist Grammars*, volume 13 of *Linguistics in Potsdam*. Univeristät Potsdam, Potsdam, Germany, 2001.

J. Michaelis. Derivational minimalism is mildly context-sensitive. *In Linguistics in Potsdam* (*LiP*), volume 5, pages 179-198. Universität Potsdam, Potsdam, Germany, 1999.

C. POLLARD. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. PhD thesis, Stanford University, Stanford, CA, 1984.

M. PUCKETTE. *Pd: real-time music and multimedia environment (version 0.37-4)*, 2004. <http://crca.ucsd.edu/~msp/software.html>

M. PUCKETTE. Pure Data: another integrated computer music environment. In *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37-41, Tachikawa, Japan, 1996.

H. SEKI, T. MATSUMURA, M. FUJII, AND T. KASAMI. On multiple context-free grammars. *Theoretical Computer Science*, 88:191-229, 1991.

S. M. SHIEBER. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333-343, 1985.

E. STABLER. Derivational minimalism. In C. Retoré, editor, *Logical Aspects of Computational Linguistics (LACL '96)*, volume 1328 of *Lecture Notes in Artificial Intelligence*, pages 68-95. Springer, Berlin, Heidelberg, 1997.

E. STABLER. Remnant movement and complexity. In G. Bouma, G.-J. M. Kruijff, E. Hinrichs, and R. T. Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*, pages 299-326. CSLI Publications, Stanford, CA, 1999.

P. STAUDACHER. New frontiers beyond context-freeness: DI grammars and DI automata. *EACL Proceedings*, pages 358-367, 1993.

M. STEEDMAN. A generative grammar for jazz chord sequences. *Music Perception*, 2:52-77, 1984.

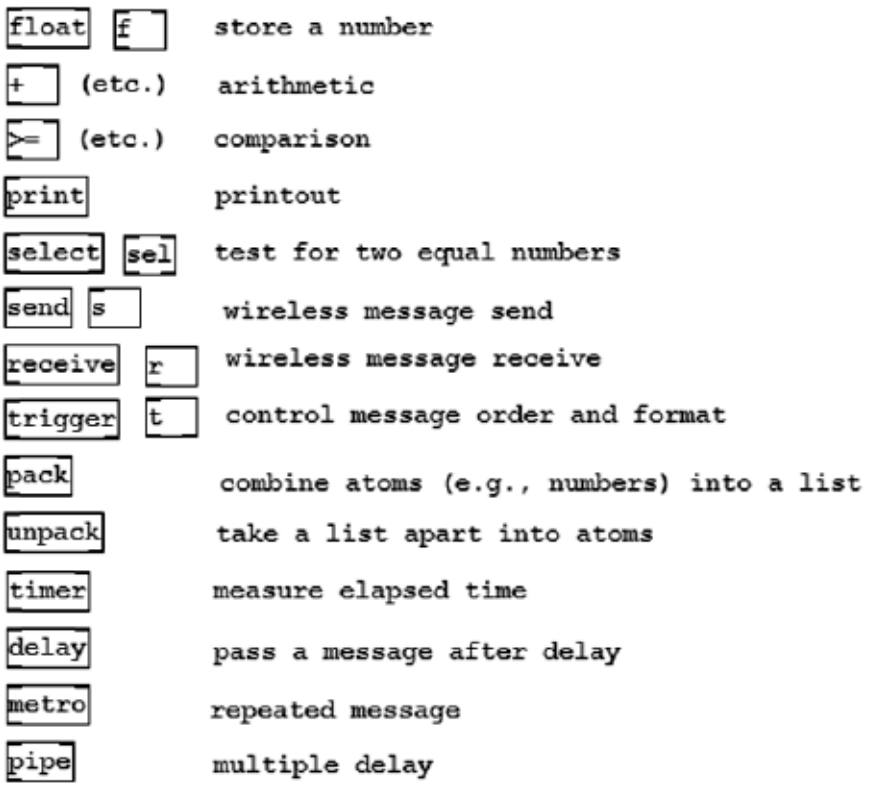
M. STEEDMAN. Gapping as constituent coordination. *Linguistics and Philosophy*, 13:207-263, 1990.

M. STEEDMAN. The blues and the abstract truth: Music and mental models. In *Mental Models in Cognitive Science*, pages 305-318. Erlbaum, Mahwah, NJ, 1996.

A. M. TURING. On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42): 230-265, 1936.

I. XENAKIS. *Formalized Music*. Indiana University Press, Bloomington, 1971.

So far we've seen the following objects, some of which have abbreviations. Right click on any one to get reference documentation:



There are many others... you can see a complete list in INTRO.txt in the reference patches (../5.reference).

updated for Pd version 0.34

Pd & Synaesthesia

James Tittle / IOhannes m zmölnig

Human evolution has created five prominent senses, but it seems they are not enough to satisfy the mind's desire to understand its environment.

Introduction

What is Synaesthesia?

Synaesthesia has long been a popular concept in the arts, but actually derives from a neurological condition where various sensory inputs are perceived as some other modality; well known examples include "hearing" color or "seeing" sounds. While potentially debilitating in the medical condition, a blending or cross-pollination of the senses is seductive to the artist in pursuit of different perspectives (and interpretations) of reality. However, as desirable as a blending of sensations may seem, historically it has been difficult to achieve in the fine arts.

History of Synaesthesia

Comprehensive historical accounts of the artistic realization of synaesthesia are widely available, but a short introduction is possible. It must be recognized that the realization of a merging of the senses has long suffered technological obstacles. One of the earliest examples of an attempt to physically enact sound and vision in one device is Louis-Bertrand Castel's *Ocular Harpsichord* (1735). This device consisted of a normal harpsichord mechanically extended by a frame with colored tape, which allowed candlelight to stream through in conjunction with key presses.

It wasn't until the widespread availability of electric light and other electromechanical devices that much further advances could occur. Early 20th century experimental filmmakers Viking Eggeling, Hans Richter, and Walter Ruttmann were among the first to extend static canvas-based painting to film, and immediately attempted to join the new "moving" paintings to sound. The same time saw the development of several specialized mechanical devices, such as Tomas Wilfred's series of *Claviluxes* or Oskar Fischinger's *Lumigraph*.

Film, Lights and Magic

The 1950s and 60s saw a second generation continue the earlier efforts in filmmaking, with a meticulous attention to detail exhibited by inscribing individual frames of film with sound induction patterns, such as seen in the work of Ernst Schmidt Junior in "tonfilm", Norman McLaren, and John and James Whitney. James Whitney is also noted as one of the earliest adopters and proponents of the computer in the arts, and devoted much of his work to the pursuit of a union of sound and visuals he termed "Digital Harmony".

How does Pd Relate to Synaesthesia?

Pd is a graphical computer-music language. Small graphical elements are combined into bigger functional units by wiring them together. While Pd-programs (commonly referred to as "patches") are often used to produce sound, writing such a program actually entails working within a graphical tool: in essence, the act of creating music is an act of creating an image.

Since a major hallmark of synaesthesia is a crossing of borders between different senses, the problem becomes one of quantification within different domains, and the borders of these domains are generally difficult, if not actually impossible, to cross. For instance, a somewhat simplistic example of a synaesthetic phenomenon consist of the literal "stars" you see when experiencing a collision of a hard object with your eye: even though both the perception of pain and vision are transmitted via the same means (electric charge carriers) in parallel nerve pathways, it takes a considerable amount of energy for the effect to take place. Fortunately, computers can ease this task.

Pd stands for *Pure Data*. As the name indicates, it attempts to treat *data* – which is everything a computer can think of – exactly as what it is: data. Pd does not make any assumptions on what this data might eventually represent. Whether it is sound or an image, a letter or a position in space, everything is expressed by numbers. And numbers can be added, subtracted and manipulated easily in any (mathematical) way. While these data are available as plain numbers in every computer software, high-level user applications usually do not allow one to access the data as such, but encapsulate it within artificial layers of abstraction, which encumbers the process of synaesthetic creation in an unnecessary way. Pd, on the other hand, does not differentiate between the various types of data presented, and thus allows one to more easily cross the borders between different perceptions.

Pd stands for *Public Domain*. Being available as a Free Software / Open Source project, Pd is easily extensible with plugins written in other programming languages. While most of the interaction and modification of data can be done in "plain Pd", extensions allow one to access low-level hardware interfaces from within the environment, thus making new sources of sensor data accessible. Through its technical and political openness, Pd has aggregated a large

number of software developers with both artistic and technical background, many of them adding new possibilities of interaction between various types of data as they see the need.

Examples with Pd

Many Pd projects have encapsulated synaesthetic ideas (intentionally or not!), certainly more than can be mentioned here; but a few merit current note. Since Pd is primarily considered an audio manipulation program, many pieces tend to use sound to generate video, but perhaps more interesting are the projects that convert video to sound. Erich Berger's "Tempest" uses the Pd extension GEM to produce 2D & 3D abstract animated shapes, which then control a sound generated from the electromagnetic radiation of the display monitor as received by AM radio. In a performance at the first International Pd Convention entitled "The Purple Haired Kitchen Experiment", Tom Schouten used his own library PDP and a video camera trained on the screen. This confrontation of the computer with its "self" produces self-referential/fractal visual feedback patterns, which in turn are used as the waveform of the sound. In both of these projects, a symbolic visual grammar and tight synchrony with sound not often found in synaesthetic pursuits is inherently produced.

The piece "Ectodermal Ways" by Nicole Pruckermayr uses stress to generate sound, video and atmosphere. Encased in a full body latex-suit, she isolates herself from the environment. Suspended from an elastic umbilical cord, she floats through space without any direct contact to the outside world. Since the latex-suit seals her skin, she produces an enormous amount of sweat, which is measured by a number of medical sensors. The distribution of her sweat, determines the structure of an environmental soundscape as well as the floating movement of a virtual camera through the 3D computer model of the artist's brain.

Finally, in Pd Workshops, Ben Bogart pays special attention to what he terms "Metaphorical Networks". Metaphorical Networks are defined as a mapping of one domain to another, such as the relation of a "sun rising" to an "eye opening". This conceptual framework is designed to establish an artistic ground within which further meaningful art is created.

Conclusion

Proposed "Taxonomy"

- metaphor
- transform
- implicity

While preparing this short review of synaesthetic art, we noticed that while the term can be broadly applied, perhaps some organization is needed to spur further exploration. To this end

we developed an embryonic "taxonomy" of artistic synaesthesia consisting of three basic types: transforms, metaphor, and implicity.

"Transforms" are the most basic of the three and are the easiest to implement in Pd; indeed, any patch created delivers some form of data transformation. This most basic level could represent projects that arbitrarily map data from different domains without regard for retaining a united meaning to the dual domains. In other words, the parameter space of the input and output domains are related only at the whim of the artist, and will likely produce wildly varying or unexpected results that evade control.

"Metaphors", then, would add an extra dimension of meaning to the work, such that one conceptual domain produces analogical information in a second domain. These metaphors or analogies are not arbitrary, but give a deeper understanding of the two domains that isn't readily apparent when the two domains are inspected separately. Obviously Ben Bogart's work mines this seam, and pieces such as "Ectodermal Ways" might also belong in a category like this.

"Implicity" points toward a deeper level of meaning. One domain cannot be separated from the other without losing the process as a whole. The parameter space of each domain is specifically mapped to the other in a naturally emergent manner. Both Erich Berger's and Tom Schouten's work falls into this category: the visual images or the sound forms considered alone have substantially less identity and emotive power than their actualization together.

Illustration 1: A graphical Pd patch

We offer these categorizations not to impose stifling limitations on artistic expression, but more in the hope that they will identify rich areas of inquiry waiting for further exploration. The historical record abounds with attempts at artistic synaesthesia, most of which are only half-realized, from our perspective: with tools such as Pd in our hands, one can hope for fuller works in the future.

Images

Illustration 2: Whitney's idea of "Digital Harmony"

Illustration 3: Pruckermayr performing "Ectodermal Ways"

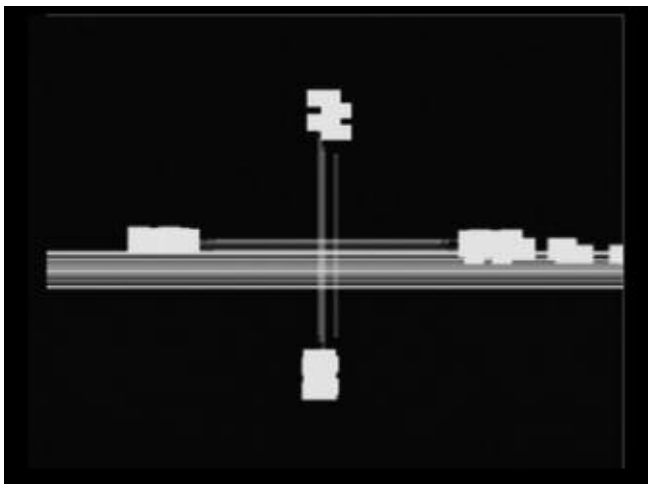


Illustration 4: Berger's "Tempest"

Illustration 5: Andrew Glassner's idea of a Shape Synthesizer
image copyright (c) Andrew Glassner

References

Bibliography on Synaesthesia. 10 Jan. 2006 <http://www.flong.com/synesthesia/index.html>
Pure Data. 10 Jan. 2006 <http://www.puredata.info/>
Schmidt jr., Ernst. Tonfilm. 10 Jan. 2006 <http://www.sixpackfilm.com/catalogue.php?oid=192>
Berger, Erich. tempest. 10 Jan. 2006 <http://randomseed.org/tempest/>
Glassner, Andrew. Shape Synthesis. 10 Jan. 2006
<http://www.glassner.com/andrew/cg/research/shapesynth/shapesynth.htm>
Tittle, James. A Shape Synthesizer. 10 Jan. 2006
<http://puredata.info/community/projects/convention04/lectures/tk-tittle/>
Pruchenmayer, Nicole. Ectodermal Ways. 26 April 2006
<http://umlaeute.mur.at/Members/nap/projects/ectoderme-wege/>





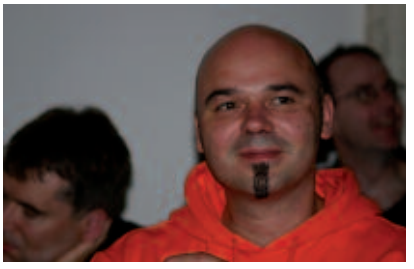












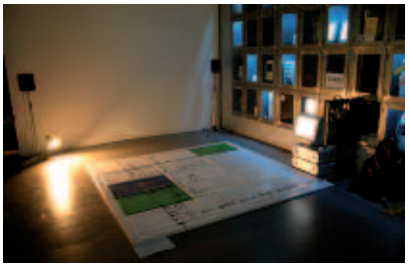








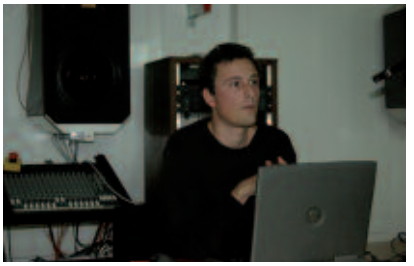










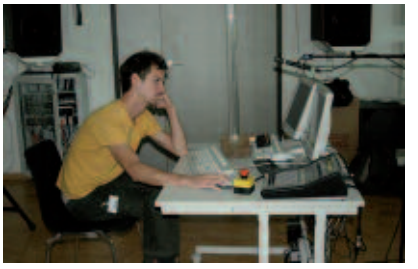




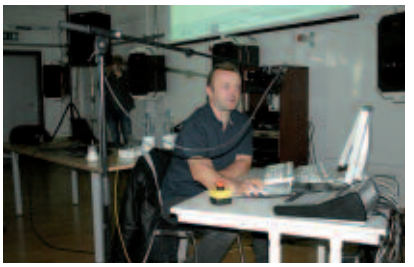




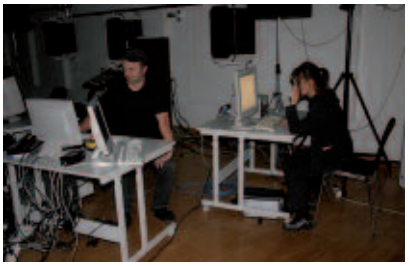
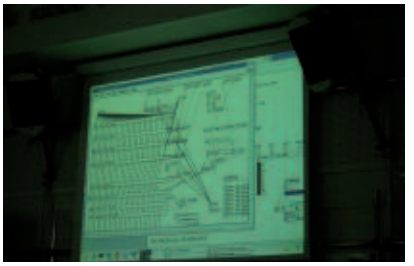




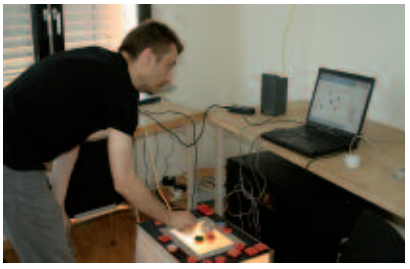




























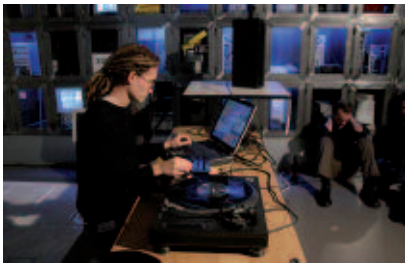


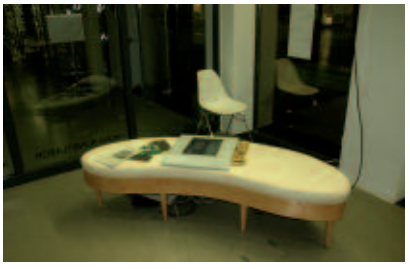


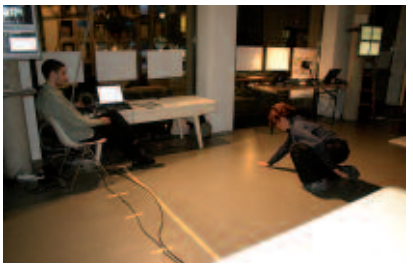


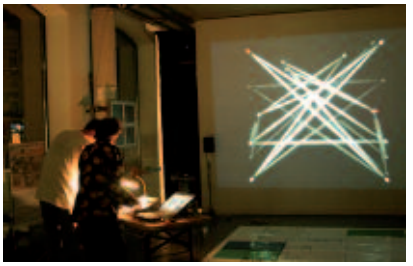


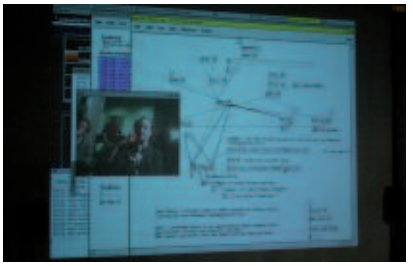




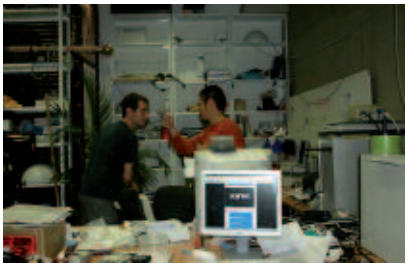


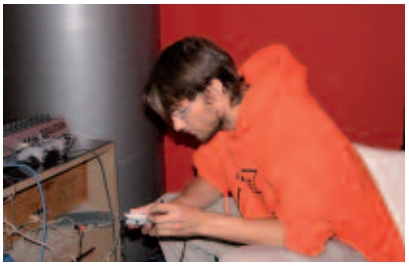




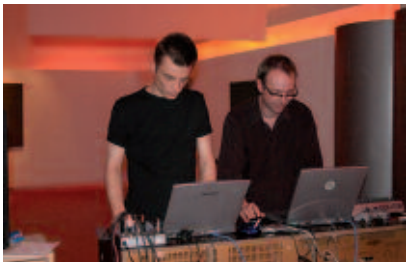



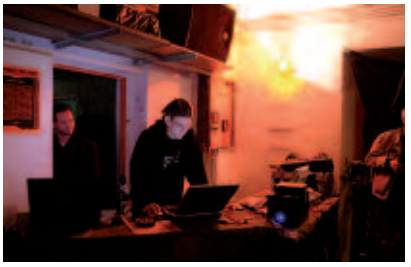
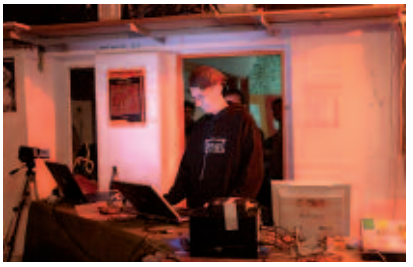


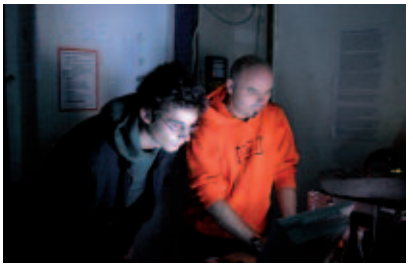
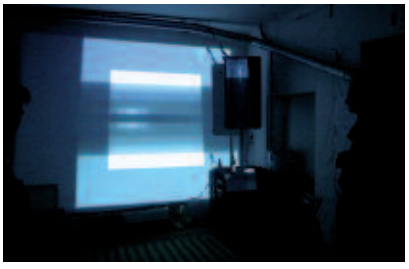


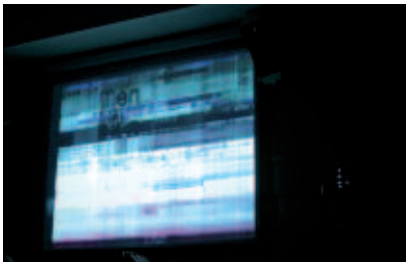





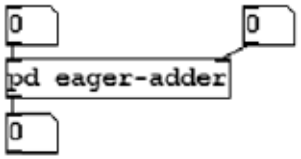






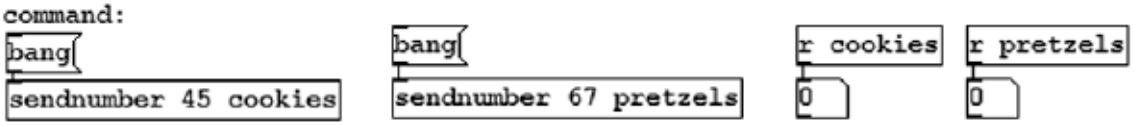
You can nest entire windows inside Pd boxes (and so on, as deep as you wish.) There are two different ways to do it. First, if you just want to add a room to the house, so to speak, type

pd sample-subpatch <-- you can give the window a name as an argument

If you click on the box (in run mode) the subwindow appears. Click on the one below to see how you give a subpatch inlets and outlets.



There is also a facility for making many copies of a patch which track any changes you make in the original. The subpatches are called abstractions. For example, here's a simple abstraction that sends a number to a "receive" on command:



There is a separate file in this directory named "sendnumber.pd" which is loaded every time you type "sendnumber" in a box. Click on a "sendnumber" box above to see it. You can make changes in the subpatch and save them. The changes will be saved back to sendnumber.pd and not as part of this (containing) patch.

If you change one copy of an abstraction the change isn't automatically made on any other copies. You must keep track, save the changes, and cause Pd to reload the other copies (for example, by closing and reopening the containing patch.)

note that "$1", etc, has a different meaning in object boxes (open one of the "sendnumber" abstractions for comments.)

updated for Pd version 0.34

Is Pd Art? No and Yes. Two Attempts

Jürgen Hofbauer / Marc Ries

First Attempt

To the structure of the question. The question of whether something is art or not, seems to arise with immutable persistence whenever something "new" appears.[1] Just as an era can be measured and understood by its answers, this question could be understood as the constituting moment of the age which with torturous maneuvering has been paraphrased as postmodern; the enemies of cryptomanic definitions would rather speak of a post-fascist or post-European age.[2]

In case the questions, which are asked at a time, in fact determine the character of their epoch – such as with the question about the essence of love a platonic age begins in an area of the erotic (see Foucault[3]), or an epoch of maturity emerging from the military ranks – with the question "*what is the Enlightenment*" (see Kant), or a post-communist age with the question "*who are the people*" then three further questions must be added to describe the style of the post-European age: *Is this democratic or fascist? Is this discriminatory or not? Is this sexist or not?* The circle of these four questions can be read as the core of a new anthropology which will slowly but surely supersede further Kant's anthropological questions which are understood as the center of an anthropology of the Enlightenment (*What can I know? What should I do? What can I hope for? What is a human being?*[4]). Nevertheless, there will still be structural analogies between Kant's questions and those of the postmodern age. For just as one cannot quietly stay with a single question in the *anthropology of the Enlightenment* without exerting a considerable influence on the answer of the remaining questions, in a post-fascist age one is also driven from one question to the next. (And the new as well can only be understood from these viewpoints: *Now, if it is not art, is it fascist? If it isn't fascist, is it perhaps nevertheless discriminatory? If it isn't that either, is it at least sexist or at least not?* – The cool ambivalence of

---

1 The question posed in the title was also a theme of a plenary discussion at the convention. It was eagerly discussed, then given up after a while (perhaps for strategic reasons).

2 The collapse of European fascism in the first half of the twentieth century and the *hype* of democratic structures strengthened by this support the definition of a post-fascist age (supports the idea of a post-European age). The fact that wars affecting the continent are no longer fought on it supports and after the events of *London 05*, Europe can be seen as collateral damage of US foreign policy.

3 See: Michel Foucault, *Der Gebrauch der Lüste, Sexualität und Wahrheit 2* (Frankfurt am Main: Suhrkamp, 1989), p. 294ff.

4 Immanuel Kant, *Werke in zehn Bänden*, Hg. v. Wilhelm Weischedel, Bd. 5 (Darmstadt: Wissenschaftliche Buchgesellschaft, 1968 [nochmals überprüfter Nachdruck von 1959]), p. 448.

the strait-laced thinking of political correctness is as unambiguous as tragic, above all for those artists who do not submit to the antagonisms of the new canon.) The entanglement of the system covers up the critical difference; postmodern questions no longer center on human beings but solely on the new, the novel. They are in fact displays of a decentralized subject (an anti-subject), a no longer anthropocentric thinking, a self-forgetting stance of the person receiving. At the same time, they give testimony to a ridiculous and tragic narrowness, a neo-fundamentalist worldly wisdom which can only measure each newly appearing phenomenon with the antagonisms of *aesthetic/unaesthetic, fascist/democratic, discriminatory/unprejudiced, sexist/asexual*. The eye of this anthropology is optimistic and as agape as greedy, focused on the new, but it is a damaged gaze, having become blind from the questions from its own tongue about the character of the new experience. The perhaps sometimes much too metaphysical babble of questions like Kant's gives has given way to the gray star of the bourgeois desire for definition.

The world, constantly on the lookout for the new, driven and professionalized by an "unenlightened" subject is the emblem of a new age.

Result and process oriented aesthetics. The question of what all this has to do with Pd must be answered with a reminder which leads to an essential paradox of art theoretical thinking, the differentiation between *process* and *result oriented aesthetics*. Reflection on the aesthetic value of Pd – which as mentioned before is a characteristic narcissism of the previous century – can be taken as a form of play of this more original and though forgotten, nevertheless very trusted conflict. It becomes possible to regard Pd as a process provided that a minimal readiness is found. The attempts to secure the dominance of process in the territorial waters of aesthetics may be too old to be blessed with a clear historical origin and understood. From the recent past, at least some of the following (without being exhaustive) come to mind: the equal opportunity of home recordings (from the *Basement Tapes* to *Smog* to *The Headphone Masterpiece*), the democratization of the means of production (formulated somewhat generally, but because of this, it is probably known what is meant), action painting, actionism, the whole area of media art, etc., etc. If Pd now claims – which is very obvious – to be a further step in the direction of a process oriented aesthetic (what in turn is obvious because the products of Pd artists often do not differ essentially in their phenomenalization of results from those of unimaginative software users), then a welding process with other process oriented aesthetics – media aesthetics first of all – is inevitably set into motion. Yet without doubt, the discourse about the primacy of the process is an archetypal genre of the twentieth century. (Although the prelude of this discourse goes back further than our heuristic-simplistic position may admit, isn't for example Lessing's celebrated question of inwardness, whether Raphael wouldn't have been just as great a painter if he had been born without hands, nothing more than a shift in the focus of attention in the direction of a process oriented aesthetic?)

But what if, despite the media age, the work of art suddenly returned to the forefront, if the epidemic of boredom left the quarantine of art galleries in order to return to seemingly dead categories – artists, people, and works, if not *great successes* – in the hope of an external recov-

ery in passion? If, in opposition to pessimistic labeling as reactionary restoration, result oriented aesthetic celebrates a renaissance?

Second Attempt

The question "Is Pd art?" can possibly receive a second question as an answer. This second question revolves around the problem of indistinguishableness: Can a sound or image made with Pd be distinguished from a sound or image produced with conventional software? The question should probably be answered in the negative; nevertheless, the nature of the question – the comparison – leads to a field that reaches far beyond that of traditional aesthetics. Pd sound does not just represent the intentionality of an artist; it represents the *decision* both for a certain procedure and for a culture-technical disposition instead of another. This means that Pd sound and images open up not only to a perception and interpretation of the receivers; in fact, they call for a "position" from them in the face of the said decision. It is precisely this moment which makes of Pd an artistic strategy which cannot be seen or heard but rather *understood* and *put into perspective*.

But what is this culture-technical disposition? It can be thought of in two ways: on the one hand, as the application of Pd as Open Source, thus the disclosure of the source code and its collective further development, on the other hand, the formally, completely unified control of sound and image, thus the structural equality of both: the resolution of the separation of the Apollonian and the Dionysian.

Pd as Open Source

Open Source begins when something is not yet a good or a thing, begins thus before a practical value which assumes material and process, already before a practical value that assumes material and process, and thus long before the exchange value invested in the practical value and the surplus which results from this. OS stands for the leaving open, the imperfection, the becoming of a production. This is considered discontinuous and contingent. It does not follow a master plan and is not goal-oriented; in its development it is unpredictable.

OS is presented in three steps:

– The "free" source code which is disclosed is reconstructed; all who want to use it think about it; it is copied, translated, transmitted: in this respect a "mimetic value" for OS can be spoken of here, a value which makes the copying and the representation of code in one's own computer possible. This mimetic value precedes the practical value.
– The code is enhanced for each special application; it is developed, it is differentiated.
– The results and the new elements of the code are made available again to everyone; they are thus shared with everyone; further thoughts are disclosed to everyone.

When Marx wrote that the circulation of goods, thus the circulation of goods as the coupling of practical value and exchange value, constitutes the origin of capital, "in the sixteenth century, international trade and the world market inaugurated the modern life story of capital", then the worldwide circulation of programs and operating systems as Open Source in the Internet does not constitute a new era of capital or consumption but one of the power of the collective; this defines a society of co-producers, not one of the spectacle.

OS can be translated not into capital values but directly into that of the *agency* of its users and developers, thus into "*performing* values." Even when OS is not profit or surplus oriented, its goal is not loss or the unproductive overestimation in the sense of Bataille's economy. The productive turns into the *performance*, into separate actions and positions, the interferences of countless users worldwide. It is neither a matter of possession nor of lacking possessions but of participation in the collective becoming and functioning of a system, of a program. OS as an abundance of the means of production belongs to those who are interconnected with each other, who develop it further. OS is not the conventional product of an industry and its shareholders but rather the respective product, in good German the *Erzeugnis* (product), whose production, which is as a matter of course never able to be finished, always resonates in it. Its testimony is that of micro and very small movements of all those who bring it about, use it coherently and develop it further.

The performative has monopolized the present as a concept and as an aesthetic practice. When you look closely, there are acts in a *performing way* everywhere. It seems that the time is ripe for the individual assure his or her power of action, of differential *agency*. It is not so much simply stated that there is political as well as economic power; in fact, something like a countervailing power is tested out in networked applications or also in the use of a "remote control". Pd as OS is committed to this countervailing power, committed to communicating it in hearing and vision as an element of knowledge, of the realization of perspectives in exactly these premises.

Pd as Tragedy

Nietzsche examined the problem of the aesthetic effect that comes into being, if the separated potencies of art – considered singly – or more precisely the Apollonian and the Dionysian view are actuated side by side. Or how music is related to conception, notion, idea and image.[5] Nietzsche had described the Apollonian view as an *artificial world of the dream* (as a *vigorous delusional vision* as *sensual illusions*) and esteemed the *speciousness* as *medial world of art what is again the principle of individuation then*.

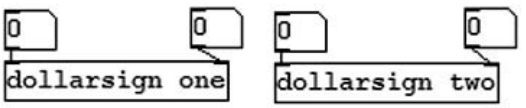
[5] Friedrich Nietzsche: "Die Geburt der Tragödie". In: KSA 1. Berlin/New York: Walter de Gruyter, p. 104

The Dionysian view, in contrast, is celebrated as the *artificial world of noise*, as the shattering to pieces of individuation, as *total self-oblivion* in the medium of music. Although both always appear together, the sonorous self volitions is tempered by a discharge of series of images.

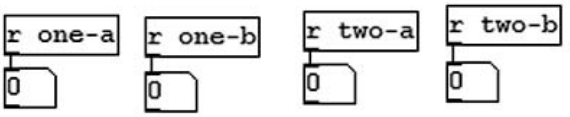
Nevertheless the question must be asked what an aesthetic effect arises when image and music not only exist next to each other but also *interact with one another*? If both are due to the same principle of origin – for example to the programming language called Pd, which enables "sound painting", "Pd (aka Pure Data) is a real-time graphical programming environment for audio, video, and graphical processing." In a primary stage, Pd was developed to enable the production of electronic music compositions independent of proprietary technology. Sounds are produced through *patches*, a kind of "diagrammatic programming" in a graphical environment; they have resemblance to geometric figures and numbers, which are – as the generic forms of all possible objects of experience – used a priori for all objects.[6] Schopenhauer, whom Nietzsche cites here at length, defines music as "a universal language to the highest degree" which works with *universalia ante rem*, thus with universalities which lie *before* things, seemingly without material or body, which are *pure form*. Its transformation through formal, algorithmic operations into the form of *patches* seems to favor this understanding of music. Independent from the physicality of the instruments, it is now developed on the "universals" of microprocessors. These too are not interested in things but in the pure forms of calculations. What is out of the ordinary is that the *patches* can similarly be deployed for the generation of images. What Pd generates in images are not "appearances", is not "something that appears to be", but just like sound, are a self-expression of the *will of the machine*, if you like. This strict equality of sounds and images in the process of their fabrication, the indistinguishableness of their "essence", is not recognized by the senses, but knowledge about Pd makes possible – comparable to the use of Open Source – a certain perception of the products, for instance the performing character of a concert. These perceptions should also lead to a reading of Pd as "art" and as an aesthetic intervention which equally dissolves the separation of manufacturing and using, like that between the art worlds of image and music. The transformation of one into the other and the separate areas becoming equal can indeed be described (with Nietzsche very affirmatively) as "tragic art", as a culture-technical *disposition* which unfolds in the shadow of the logic of power.

6 Ibid, p. 105

You can use dollarsigns in abstractions to get local sends and receives as shown here.



Open both copies to see what's happening...



updated for Pd version 0.34

What it takes to be a RRADical

Frank Barknecht

The acronym RRAD means "Reusable and Rapid Audio Development" or "Reusable and Rapid Application Development". The Pd patches that form RRADical are intended to solve real-world problems on a higher level of abstraction than the standard Pd objects. Where suitable, these high level abstractions should have a graphical user interface (GUI) built in. As I am more focused on sound production, the currently available RRADical patches mirror my preferences and mainly deal with audio, although the basic concepts also apply to graphics and video work using, for example, the Gem and PDP extensions of Pd.

Pre-fabricated high-level abstractions may not only make Pd easier to use for beginners, they also can spare lot of tedious, repetitive patching work. For example, building a filter using the [lop~] object of Pd usually involves some way of changing the cutoff frequency of the filter. So another object like a slider will have to be created and connected to the [lop~]. The connections between the filter's cutoff control and the filter can also be done in advance inside a so-called abstraction: in a Pd patch file saved on disk. If the graph-on-parent feature of Pd is used, the cutoff slider can be visible when using that abstraction in another patch. The new filter abstraction now carries its own GUI and is immediately ready to be used.

In my experience a lack of these kinds of abstractions in the Pd distribution is one big hurdle for new users. This is reflected in typical questions asked on the Pd mailing lists: Does someone know a good drum machine for Pd? Does anyone have a sequencer? One reason for the success of Reaktor, surely, is the availability of a lot of usable example patches. Reason takes this to the extreme in that it practically only consists of "example patches" and does not allow for building custom modules at all.

Problems and Solutions

In the course of designing RRADical as such a modularized system of high-level abstractions, several problems had to be solved. Two key areas are:

Persistence
: How to save the current state of a patch? How to save more than one state (state sequencing)?

Communication
: The various modules are building blocks for a larger application. How should they talk to each other?

It turned out that it is possible to solve both tasks in a consistent way using a unique abstraction. But first let's look a bit deeper at the two problems.

Persistence

Pd offers no direct way to store the current state of a patch. Here's what Pd author Miller S. Puckette writes about this in the Pd manual in the section "2.6.2. persistence of data":

> Among the design principles of Pd is that patches should be printable, in the sense that the appearance of a patch should fully determine its functionality. For this reason, if messages received by an object change its action, since the changes aren't reflected in the object's appearance, they are not saved as part of the file which specifies the patch and will be forgotten when the patch is reloaded.

Still, in a musician's practice some kind of persistence turns out to be an important feature, which many Pd beginners do miss. And as soon as a patch starts to use lots of graphical control objects, users will – and should – play around with different settings until they find some combinations they like. But unless a way to save this combination for later use is found, all this is temporary and it is gone as soon as the patch is closed.

For RRADical I chose Thomas Grill's [pool] external to handle persistence. Basically, [pool] offers something that is standard in many programming languages: a data structure that stores key-value pairs. This structure is also known as hash, dictionary or map. With [pool] these pairs also can be stored in hierarchies, and they can be saved to or loaded from disk. Several [pool]s can be shared across patch borders by giving them the same name. The [pool] object in RRADical patches is hidden behind an abstracted "API", so it could be replaced by some other object in the future.

Communication

Along with persistence, it also is important to create a common path through which the RRADical modules will talk to each other. Generally, the modules will have to use what Pd offers them, and that is either a direct connection through patch cords or the indirect use of the send/receive mechanism in Pd. Patch cords are fine, but tend to clutter the interface. Sends and receives on the other hand will have to make sure that no name clashes occur. A name clash is when one target receives messages not intended for it. For a "library" like RRADical it is crucial that senders in RRADical abstractions use only local names with as few exceptions as possible. This is achieved by prepending almost all RRADical senders with the string "$0-", that is replaced with a number unique to a certain instance of an abstraction.

Still we will want to control a lot of parameters and do so not only through the GUI elements Pd offers, but probably also in other ways, for example through hardware MIDI controllers, through some kind of score on disk, through satellite navigation receivers, etc.

This creates a fundamental conflict:

We want borders
: We want to separate our abstractions so they do not conflict with one other.

We want border-crossings
: We want to have a way to reach their many internals and control them from outside.

The RRADical approach solves both requirements in that it enforces a strict border around abstractions, but drills a single hole in it: the **OSC-inlet**. This idea is the result of a discussion on the Pd mailing list and goes back to suggestions by *Eric Skogen* and *Ben Bogart*. Every RRADical patch has (must have) a rightmost inlet that accepts messages formatted according to the OSC protocol. OSC stands for *Open Sound Control* and is a network transparent system to control (audio) applications remotely. It has been mainly developed at CNMAT in Berkeley by Matt Wright.

OSC can control many parameters over a single communication path (like a network connection using a definite port can transport a lot of different data.)

The OSC-inlet of every RRADical patch is intended as the border-crossing: everything the author of a certain patch intends to be controlled from the outside can be controlled by OSC messages to the OSC-inlet. The OSC-inlet is strongly recommended to be the rightmost inlet of an abstraction. All of my RRADical patches follow this guideline.

Trying to Remember it All: Memento

To realize the functionality requirements laid out so far, I resorted to a design pattern called Memento. In software development Mementos are quite common. Computer science literature describes them in great detail, for example in the Gang-Of-Four book "Design Patterns" (Gamma, E. et al. 1995). To make the best use of a Memento, science recommends an approach where certain tasks are in the responsibility of certain independent players.

The Memento itself is a kind of state record. A module called the "Originator" is responsible for creating this state and managing changes in it. The actual persistence, that could be the saving of a state to hard disk, but could just as well be an upload to a web server or a CVS check-in, is done by someone called the "Caretaker" in the relevant literature. The Caretaker only has to take care that the Mementos are in a safe place and no one else fiddles with them. It does not deal with changing the content of a state.

Memento in Pd

I developed a set of abstractions that follow this design pattern. Memento for Pd includes a [caretaker] and an [originator] abstraction, plus a third one called [commun] which is just a thin extension of [originator] and should be considered part of it. In fact, a soon to be published new version of [originator] seeks to get rid of the [commun] objects and replaces them with special messages to the [originator].

Due to the constraints of the scope of this text, I have to omit a detailed explanation of how the objects work in practice and would like to encourage you to read through the help patches that accompany RRADical and Memento. They are free.

Using Memento it is possible to "freeze" the whole state of a patch to a file on disk. A state here not only consists of the current position of sliders etc., but also includes "possible states", that is, for example, the position of a slider at a different part of a piece of music. Through OSC it is possible to select which of the "possible states" should become the next active state. A way to interpolate between states, to use weighted sums of various possible states, is currently being researched, however, I must admit that this is indeed a tricky problem to solve inside the infrastructure that Memento provides.

Putting it all to RRADical Use

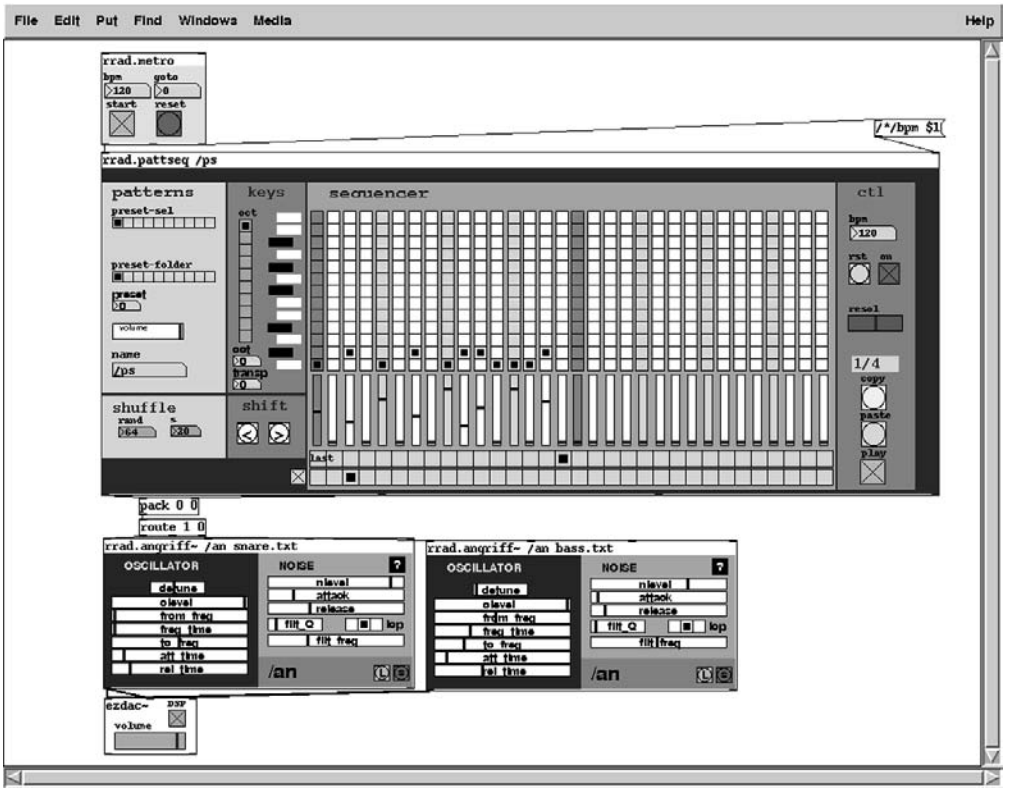
I developed a growing number of patches that follow the RRADical paradigm, among these are a complex pattern sequencer, some synths and effects and more. All of these are available in the Pure Data CVS, which currently lives at *pure-data.sourceforge.net* in the directory "abstractions/rradical".

The RRADical Patches using Memento have two main characteristics:

Rapidity
: Ready-to-use high-level abstraction can save a lot of time when building larger patches. Clear communication paths will let you think faster and devote more attention to the really important things.

Reusability
: Don't reinvent the wheel all the time. Reuse patches like instruments for more than one piece by just exchanging the Caretaker-file used.

An example of how the patches can be used is shown in the following screenshot:



Here you see a step sequencer module called [rrad.pattseq] which drives two drum synths, [rrad.angriff]. A patch like this can be made very quickly even as a Pd beginner, and it provides instant gratification.

Much, but not all is well yet

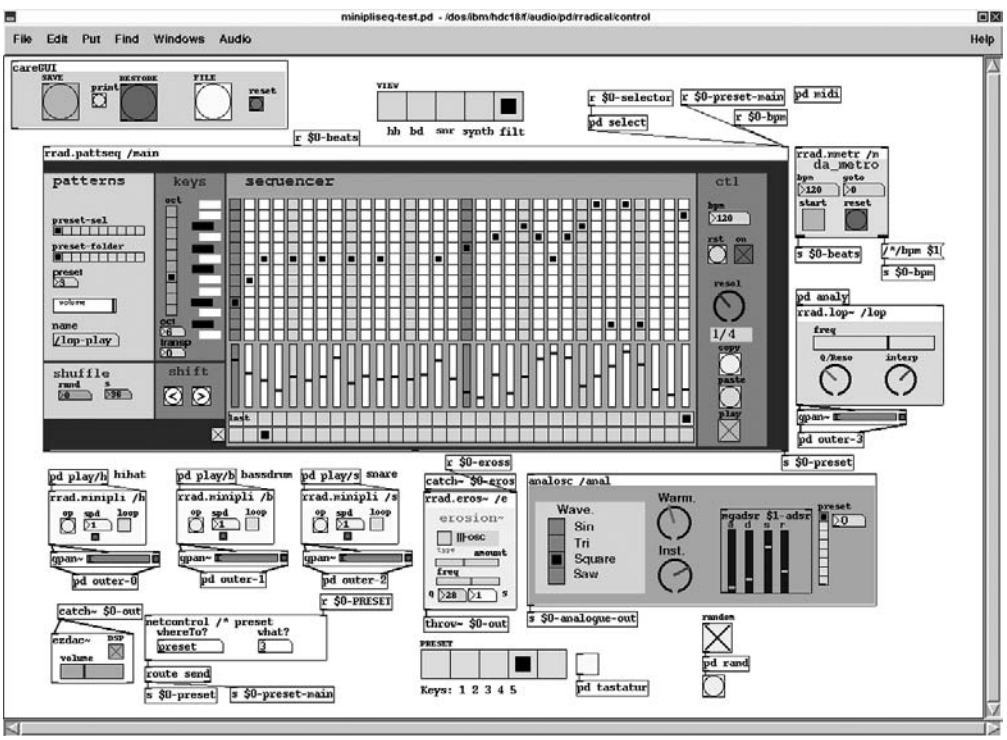
Developing patches using the Memento system and the design guidelines presented here has had a remarkable impact on how my patches are designed. Before Memento, quite a bit of my patches' content dealt with saving a state in various, crude and non-unified ways. I even tried to avoid saving states at all, because it always seemed to be too complicated to bother with it. This limited my patches to being used in improvisational pieces without the possibility of preparing parts of a musical story in advance and "designing" those pieces. It was like being forced to write a book without having access to a sheet of paper (or a hard disk nowadays). This has changed: having "paper" in great supply has now made it possible to "write" pieces of art, to "remember" what was good and what should preferably not be repeated, to really "work" on a certain project over a longer time.

RRADical patches also have proven to be useful tools in teaching Pure Data, which is important as the usage of Pd in workshops and at universities is growing – also thanks to its availability as Free Software. RRADical patches can be used directly by novices, as they are created just like any other patch, but they already provide sound creation and GUI elements that the students can use immediately to create more satisfactory sounds than the sine waves usually taken as standard examples in basic Pd tutorials. With greater proficiency the students can later dive into the internals of a RRADical patch to see what is inside and how it was done. This allows a new top-down approach in teaching Pd, which is a great complement (or even alternative) to the traditional, bottom-up way.

The use of OSC throughout the RRADical patches created another interesting possibility: collaboration. As every RRADical patch can not only be controlled through OSC, but can also control another patch of its own kind, the same patch could be used on two or more machines, and every change on one machine would propagate to all the other machines where that same patch is running. So jamming together and even the concept of a "Pd band" is naturally built into every RRADical patch.

References

Gamma, E. / Helm, E. / Johnson, R. / Vlissides, E. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley 1995.



An abstraction's creation arguments may be either numbers
or symbols. Gory details are inside:

dollarsign2 three 4

updated for Pd version 0.34

The Search for Usability and Flexibility in Computer Music Systems

Günter Geiger

1. Introduction

> "He (the user) would like to have a very powerful and flexible language in which he can specify any sequence of sounds. At the same time he would like a very simple language in which much can be said in a few words." – Max Mathews

Computer Music, that is, music created on a computer with the help of mathematical formulas and signal processing algorithms has always been a challenging task, one that traditionally has been difficult to handle both by humans, because of the need of a fundamental understanding of the algorithms, and by machines, because of the demands on computational power when implementing the algorithms.

The first person who took this challenge and came up with a system for computer generated music was Max Mathews, working at the Bell Laboratories in the late 50s. We are about to celebrate the 50th birthday of computer music, and we therefore have to ask ourselves what our computer music systems (CMS) evolved into, and where to go from here.

Nowadays the processing power of everyday computer systems can easily handle most of the problems of computer music: can we say that the human factor in computer music is equally developed? Has the software matured in a way similar to the increase in hardware power? I will attempt to outline the levels of abstraction that we have reached and describe the established metaphors with a sketch of the main properties of computer music systems and a historical overview of the most influential systems and their implementation.

Intuitivity, Usabilility and Efficiency

The goal of software is to facilitate tasks for humans. This means that the achievment of the task is the minimal requirement, the quality of the software is measured by the "efficency" with which a specific task is solved.

During the last 50 years the definition of the task in computer music has broadened considerably, today taking into account almost everything that can be done with computers and music. In the beginning, producing digitally generated sound in realtime was unthinkable, so the systems did not take interactivity and other realtime aspects into account.

The broadness of the field makes it difficult to come up with one general model for computer music, and despite the evolution of computer music during the last 45 years, computer musicians today still fall back into using low level languages for expressing their algorithms.

A useful computer music system should therefore offer a reasonable level of abstraction, but it should still offer ways for expressing new concepts. Beside the abstraction of well know algorithms, computer music languages are also concerned with the performance of these algorithms.

2. From Music I to GROOVE

The era of computer music started in 1957 when Max Mathews wrote the first software synthesis program called "Music I". It ran on an IBM 704 Computer at Bell Laboratories were Mathews was working.

At that time programs where mainly written in assembler for performance reasons and because of the lack of real high level languages, so Music I, the first incarnation in a series of programs known today as Music-N, was also written in assembler. Mathews was going through iterations of updates on the Music I program, which eventually led to a version that was called "Music V", written in the high level language Fortran this time.

Music V can be considered as the breakthrough of the line of programs referenced as Music-N. Mathews' book about Music V[15] is still one of the main reference works about the structure of computer music systems, and the Music V system produced and still produces systems that follow its basic principles. One could say that Music V defined the structure of computer music programs and influenced and formed the way we think about computer music, especially if we talk about software synthesis practice.

At the time when Music V was written, computers were not fast enough to calculate audio signals in real time, so Music V was a program that was used to calculate sound files in non-real-time. In the early 70s, Max Mathews started another project called the "GROOVE" system[27], the first system that was designed for realtime control of synthesizers.

Other important works from the early era include the "MUSICOMP"[21] system by Lejaren Hiller, author of the first published computer music work in 1958, the "ILLIAC suite". This system was a composition system, and not a sound synthesis package.

The "MUSIGOL" system was a sound synthesis package based on the ALGOL language, one of the first high level languages that appeared during that time.[25]

3. Music-N Descendants

The Music series of programs was a success, and the fact that Mathews shared the code with other institutions, along with the need to port the code to different computer architectures, led to several extended versions of Music-N.

One line that came out of this process was the "Music4B" and "Music4BF" from Godfrey Windham and Hubert Howe. The F in Music 4BF Stands for Fortran, which means that parts of that system were implemented in Fortran too.

Building on Music 4B Berry Vercoe wrote a CMS for the IBM System/360 which he called "Music 360" in 1968, later on in 1974 a program for the "PDP-11: Music 11". A direct follow up of Music 11 was "CSound", coming in 1985, today maybe the best known and most used incarnation of a MUSIC-N system.

At Stanford, yet another advanced incarnation of Music V was developed, called "MUS10", featuring an ALGOL-like instrument language and enhanced set of unit generators.

In terms of software, the language of choice for high level performance programming in the late seventies, early eighties was "C", and so several systems that were implemented in C appeared, like Paul Lansky's "Cmix" program. Cmix is more a library than a computer music language, as Cmix is compiled and linked by a C compiler. Still, it has its own instrument description language, called "MINC", which is used to translate Cmix instruments into C source code.

Together with the book "Elements of Computer Music", F. Richard Moore wrote a system he called "cmusic". Cmusic can be seen as a reimplementation of Music V in C.

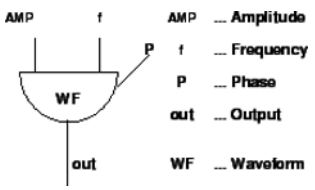
Another system widely in use today that came out of the Music V tradition is "CLM"[36], which stands for Common Lisp Music.

Although these are the more obvious successors of the Music-N languages, all modern CMS share some of the ideas of Music-N, so that we can say that its principles are generally accepted.

4. Operating Model of Music-N Languages

4.1. The Unit Generator

"There are certain strains of elements of our musical beings
that seem right be-cause they help us solve the problems that we need
to solve, such as, for instance, the unit generator" – Gareth Loy



[height=5.5cm] ugens
Figure 1.1: A unit generator diagram

```
1  INS 0 1   ;
2  OSC P5 P6 B2 F1 P30   ;
3  OSC B2 P7 B2 F2 P29   ;
4  OUT B2 B1   ;
5  END   ;
6  GEN 01100   .99 20   .99 491 0 511   ;
7  GEN 01200   .99 205 -.99 306 -.99 461 0 511   ;
8  NOT 012  1000 0.0128 6.70   ;
9  NOT 211  1000 0.0256 8.44  ;
10  TER 3   ;
```

Figure 1.2: Example Music V orchestra and score

Probably the most influential idea introduced by Mathews was the concept of the unit generator (UG). The idea of the UG is derived from analog synthesis, where modules such as oscillators, LFO (low frequency oscillators), envelope generators and filters get plugged together in order to realize a specific sound generation algorithm. The data paths through which these modules communicate with each other are driven by voltage signals. The fact that all elements deliver the same kind of signal makes the construction of new algorithms very flexible in analog synthesizers. This idea was directly mapped to the digital domain, the signals sampled and the modules implemented in software. The implementation of UGs in software is not straightforward though. Digital computation takes time, and the computation of one sample for every channel of audio at a rate of 44100 Hz was quite a task at the beginning of computer music. Even today, we still want to minimize the time spent computing, so we are always looking for a good balance between performance and quality.

Normally one looks at a unit generator as a single fundamental, non-divisible process, an implementation of a basic calculation primitive. This is used in order to implement higher level algorithms. The code of each unit generator can run for a specific time, "consuming" and "generating" audio data. After that a scheduler has to decide which unit generator has to be run next. If the time that a unit generator has run is shorter than its life span (in Music N the live span would be a note), then the UG has to remember its state for the next run.

4.2. The Score

While the unit generator modeled the analog synthesizer circuitry with its voltage levels as data, the main responsibility of the score is to handle timed events. On an analog synthesizer this corresponds to the human interaction like pressing keys and the trigger functions of the analog sequencer.

Besides the generation of lookup tables (the `GEN` entries in Figure 1.2, line 6 and 7) a score consists of timing information, instrument information and parameters for the instrument. It is a static data description language where each line specifies a record.

Generation of sound output is triggered by the `NOT` entries (Figure 1.2). The first parameter denotes time, the second one duration, the rest is passed to the instrument as parameters. Several `NOT` entries can be started at the same time or overlap.

The Music-N score language was soon found to be too limited for several tasks. First it is lacking verbosity, parameters can not be omitted, and there is no support for default values. But more limiting than that, it doesn't allow for the expression of algorithms.

In general the concept behind the score is still valid, if we look at it as the output, an interim representation, of the dataflow between a controller or a computer music algorithm and the synthesizer. A method for realtime scheduling and dispatching of the score of a CMS is described by Dannenberg[12].

5. Control Languages

Control languages are computer music languages that don't include the synthesis of sounds, but are used on a higher level for organizing and structuring how sound is handled in a CMS. The Music-N style score could be seen as a primitive control language. Nevertheless a score is only an explicit list of events with times and parameters, and as such does not qualify as a programming language, but is more of a descriptive language. Descriptive languages (such as score languages, MIDI or OSC) are not included in this survey, because they mainly deal with structure, not with algorithms. Our concern is not how to define structure but how to generate it automatically, termed generative languages by Loy et al.[23]. Music V supported a simple sys-

tem that was using a routine called `CONVT` that could translate and combine instrument parameter values.

The aforementioned MUSICOMP system by Hiller was one of the first higher level score description systems that implemented traditional harmonic rules. After the establishment of the Music-N paradigms, a logical step to take was to compute score files using algorithms written in (already existing) higher level languages, or designing languages that can express these algorithms.

One of these score languages was "SCORE"[38], generally a preprocessor for Music V score files which translated CPN (Common Practice Notation) to a score file, but this language was not able to generate a score automatically. Expanding on the idea of CPN notation, the system PLA[37] was implemented as an extension to the SAIL (Stanford Artificial Intelligence Language) programming language and Lisp, making automatic score generation very flexible.

"FORMES" is a score generating System written in VLISP, an object oriented system for high level description of musical processes, developed at IRCAM.
Other score preprocessing languages appeared, like "CScore" and "Cybil" for CSound. With the availability of realtime software synthesizers in the late 70s, forcus started to shift to realtime control languages, languages that were used to control synthesizers and had to react to realtime input from composers or players.

5.1. Implementation of Control Languages

Traditionally the work of a composer consists in writing a score. Besides the fact that computer music theoretically allows the composer to go down to the sample level and control every single aspect of music, in practice we find that at the lowest level of control, the amount of needed data is prohibitive. The unit generators represent the first layer of abstraction, and they are offering "instruments". Still writing a score for these instruments might be too tedious in practice and additionally a computer music composer wants to use the full power of the computer in order to generate music, not only automize the signal processing.

Whereas the set of algorithms for signal processing is relatively small, the set of algorithms that can be used for controlling these DSP blocks is nearly endless. This is the reason why several control languages were actually extensions of a general purpose language, making it possible to implement new algorithms and generate new structures easily.

This also means that at this level the composer's task includes actually programming or working together with a programmer.

5.2. General Purpose High Level Languages

Several systems have been implemented in general purpose languages which were not primarily designed for computer music. There are few languages which have been chosen for this task, the most important of which are probably Smalltalk and Lisp, both supposed to be representatives of two different styles of programming, namely object oriented and functional. There are only few languages in common use that only implement one of the identifiable programming paradigms. There might be as many programming paradigms as there are problems to solve. Maybe one of the future goals of a CMS is to find the programming paradigm that fits best to the problems in the domain.

Although it is not a high-level language in its modern definition, the "Formula" System[4] used a Forth Interpreter as its implementation language.

The "MODE" (Musical Object Development Environment)[30] is a system written in Smalltalk which supports structured composition and flexible graphical editing of high and low level musical objects. It includes a score description language called "SmOKe".

Another Smalltalk based System is "Kyma"[34], a system that depends on the separation of synthesis engine on a DSP and the control interface, which in Kyma's case is mainly graphical.

Currently the most widely used system based on a Smalltalk-like language is probably "SuperCollider"[29]. The SuperCollider synthesis engine can be controlled by other means too, for example Python or Q[17].

There are several Lisp systems, starting from the aforementioned CLM and including "Patchwork"[24], "OpenMusic"[26] and Roger Dannenberg's Nyquist "Nyquist"[11], a language evolved from ARCTIC, based on xlisp. Lisp itself was pretty successful and accepted by composers, it has a clear syntax, is interpreted and therefore highly interactive. The handling of lists of dynamically allocated data lends itself directly to the idea of writing a score, which then can be manipulated by the language in one environment. Lisp syntax is really easy conceptually, but in the long term it might be hard to read.

Another example of an application of functional language to computer music is the "Haskore"[11] system, written in Haskell, a modern functional language.
In general it can be said that the most important feature of the successful languages in this domain share one common principle, which is interactivity. This topic will be briefly discussed later.

6. Realtime Control Systems, Sequencers

We already mentioned the GROOVE system, which controlled analog synthesizers by generating sampled function values as control values in realtime on a digital computer. This allowed for algorithmic, although somewhat crude control of analog synthesizers and interaction by a player.

A more elaborate language for realtime control was "PLAY". "PLAY" could generate and manipulate control streams which could be individually clocked and sent to the synthesizer's synthesis modules. PLAY was a dataflow language, which means that the control it produces is constantly flowing, it didn't have the notion of instantaneous events. Several principles in music are event based though.

"4CED"[2] was a control language for the 4C synthesizer at IRCAM, and its score generating system was based on event processing. This means that instead of having constant dataflows, the control changes are triggered by events, which can in turn trigger other events or phrases of scores.
Max Mathews' "RTSKED"[28] was another event-based system, which had some notion of multitasking built in.

A descendant of RTSKED is the "Music 500"[31] system, which replaced the traditional score file with a system based on Mathews' "trigger and wait" idea.

"Flavours Band"[16] is another LISP based system for realtime control, which was built on the notion of phrase processors that can be manipulated and applied to phrases and has influenced systems like Stephen Travis Pope's "MODE".

Other early systems that can be mentioned in this domain are "MOXIE"[9] and "FMX", all reflecting the state of the art of realtime CMS at that time.

Another very interesting approach in terms of language is the ARCTIC[13] System by Roger Dannenberg. ARCTIC is a functional language with descriptive elements. The main semantic concept consists of prototypes for events. These events, once instantiated, can trigger other events or produce output functions. Time was an explicit value in ARCTIC. The event propagation was similar to that of 4CED.

Because of the constant hardware improvements, only a few control languages of that time survived the platform they were written for. In the mid-80s, Miller Puckette was working on a program to control the newly developed 4X machine at IRCAM. His program was called "MAX"[33] (after Max Mathews), and it was written for an Apple Macintosh computer.

Several coincidences led to the survival of MAX. One is certainly the upcoming market for personal computers like the Macintosh and the establishment of the MIDI protocol, which

started in the mid-80s. Max is an event based system, just as RTSKED or Flavours Band. It had one novelty, and that was its graphical representation of the language statements. Operation could be patched together just like in the representation everyone knew from the Music V orchestra explanations. Max communicated with the 4X via MIDI, which also made it useable for any other commercial synthesizer, and probably not as useful for computer music. MAX was also used as a control language for the development of the IRCAM Signal Processing Workstation (ISPW), the followup of the 4X[14].

The popularity of mainstream Synthesizers, that led to the definition of the MIDI standard in 1983, also led to a set of programs for controlling these devices. Due to the limited parameter space of the MIDI protocol, these control programs, called sequencers, were basically not more than a graphical user interface to a stripped down version of Music-N scores.

Although problematic, the separation of control and synthesis for realtime systems remained unavoidable until the mid-90s, when general purpose computers started to be able to take over the task of dedicated hardware.

6.1. Programming Models of Realtime Control Systems

Realtime control systems add an additional difficulty to the task of score generation. As scores generally control more than one voice or more than one event stream at the same time, the realtime control system has to allow for parallelism.

Expressing parallelism in a traditional programming language is awkward, as parts that take place at the same time have to be written sequentially, and therefore it is not trivial to get the synchronisation and timing between different event streams right.

Realtime control systems can not work with absolute time, such as some score generation systems do. The systems should therefore be able to schedule events in a determined future and to react to events from the outside immediately.

Reacting to events is commonly solved by callback functions, while the scheduling needs a queue where events can be scheduled and triggered at the correct times. These events can then be processed just as events from the outside (some controller or user interaction).

It is desirable for realtime control to be able to change the behaviour of the system while it is running, in order to make it possible to experiment with the system. This means that the callbacks can be rewritten and reloaded into the system.

These properties also call for an interpreted language to make realtime control processing feasible. For this reason, signal processing and control processing are separated in most systems today, since signal processing is hard to do efficiently in interpreted languages.

7. Realtime DSP Languages

Soon after computers were fast enough to do signal processing in realtime, systems for sound synthesis began proliferating. Several of the surviving synthesis programs of the post Music V era started to work in realtime, interestingly with only a few adjustments. Software sound synthesis in the style of CSound, cmusic and Cmix had their own evolution during the years of expensive and dedicated computer music workstations. The time that it took to render a CSound score to disk got smaller and smaller from year to year, until it was shorter than the actual piece, meaning that it could run in realtime.

An interesting adaptation of the Music N orchestra file for realtime interaction is the "M orchestra language"[32] of the Music 500 system, running on a special purpose array processor.

Besides the Music V style languages, another survivor of the era of signal processing on general purpose computers was Max, and specifically its successors Pure Data, jMax and MAX/MSP.

Having the whole system running on one computer facilitated several aspects of the control problem, but did not alleviate it entirely. Obviously the problem of controlling sound processing and synthesis specifically in realtime is not a problem of the synthesis/control separation, but lies deep within the algorithms themselves[20].

The 90s saw other trends in computer technology showing up, one of which is the internet. In order to be able to handle bandwidth problems, there was yet another modern incarnation of the Music V paradigm, which was called "SAOL" (Structured Audio Orchestra Language) with its score language "SASL"[35]. SAOL is an MPEG-4 standard, but it was not widely adopted, perhaps because of the lack of an efficient interpreter.

Another field that has opened with the availability of fast desktop computers are libraries and frameworks that can be used to build CMSs. They cover different areas, from simple sound processing libraries to complete cross-platform solutions. Systems that should be mentioned here are the Synthesis Toolkit (STK)[10] and the sndobj[22] library, which implement signal processing or instrument algorithms, and and the CLAM[5] framework, which offers everything from signal processing, scheduling, graphical user interface, sound hardware access up to a whole metamodel of music computation[3].

8. Current Developments

Some of the systems mentioned above are still heavily in use and evolving. Other systems are very recent, and it remains to be seen whether they will be of the same importance as some of the Systems we have seen so far.

As a quick wrap-up, I want to mention some of these systems. First, there are more Max style implementations, "Open Sound World"[1] can be considered as one of these. It tries to overcome several shortcomings of the (already 20-year-old) Max paradigms, fighting against a strong user-base of Max and Pure Data.

Other interesting additions to the Max paradigm are graphic rendering libraries. With these, the Max paradigm also attracted interest from visual artists.

A new approach on how to handle the problems in Computer music is being tried by "ChucK"[41], which introduces the concept of a "strongly timed language", referring to its sample synchronous scheduler, and the expression of "on-the-fly" programming, a technique used in order to improvise computer music, also known as live-patching, just-in-time programming and live-coding.

The "CLAM"[5] framework is on its way to setting a new standard with higher level datatypes for spectral processing and in the rather new field of music information retrieval. CLAM is also seen as a replacement for traditional scientific tools such as Matlab.

The "Marsyas"[40] system is yet another analysis/synthesis library with a special focus on music information retrieval, but lately it also handles synthesis.

Other recent systems are based on functional programing (e.g. Q-lang)[17], like "Chronic"[7] or the research system "Varese", based on the Lisp dialect scheme[18].

Java also has its CMSs in "JSyn"[8] and "JMusic"[39], and for music description the "JMSL" (Java Music Specification Language), successor of "HMSL", the Hierarchical Music Specification Language.

Several systems are less concerned with programmability than with implementing a system based on the more traditional concept of modular synthesizer/sequencer combination, like "AudioMulch"[6] or Bidule.

Also SuperCollider shows a modern, object oriented system that offers a language shell for interaction and a low latency signal processing server for number crunching.

The proliferation of computer music software has definitely changed the ways in which composers work with computers. Nevertheless, there is still an interest in CMS developments, especially for interactive art and live performances, as in most of these cases the system plays a key role both in aesthetics and in the technical effort that is needed for its realization.

9. Computer Music Systems Today

Today, computer music systems in use can be split into several classes. The biggest one is probably the commercially most successful one. It includes sequencers, general software synthesizers and systems that are based on the traditional principle of mouse interaction. These systems tend to be easy to use, but due to their lack of flexibility, they are not useful for certain tasks of computer music and interactive art.

The more interesting group comprises systems such as those described above. One could say that these systems are built on two main concepts today.

One concept is traditional programming, represented by CSound, ChucK and SuperCollider, for example, the other one is graphical programming, represented by Pd, MAX, Reaktor and several others.
Evolving are hybrid forms, like CSound instrument editors or scripting language support for graphical applications.

9.1. Traditional Programming

Traditional programming forces the user or programmer to be able to abstract the task to a high degree. When using these systems, two principles come into play. The first one is the memorization of the elements of the language. Languages can generally be defined by a fundamental set of allowed tokens or words and a syntax that is used to define allowed statements. The memorization of the tokens is normally an easy task, but nevertheless one that has to be done before being able to work with the language. This task can be made easier by a built-in list of allowable tokens and auto completion.

The syntax is the way these words can be put together in order to form correct sentences. The difficulty of general programming languages starts when trying to learn the syntax of a language. For a programmer, this is easier, because programming language syntax tends to be based on the same principle. However, computer music systems might need some less common extensions. Nevertheless, before being able to start to use a system based on traditional programming, the user has to learn the syntax, at least to a point where it admits him to express a handful of useful statements. This can be compared with learning foreign languages. (It is not by chance that the first programs people normally write in a new language are for printing, "Hello world", "hello" being the first word someone would learn in a foreign language too).

There is still a third level that has to be reached when acquiring the knowledge to be able to use a text-based computer music system, it is called the semantics. The semantics of a language is the part that gives sense to the sentences. Although one might build perfectly correct sentences (statements), the idea that they express might still make no sense.

Textual computer music systems traditionally do not offer much help for learning. Generally the user has to go through a steep learning curve in order to be able to express ideas within the systems. Furthermore, some of the ideas behind computer music systems are fairly abstract. Most of the time, textual systems offer little or no help for understanding these concepts, which makes them harder to use for some users.

The task of debugging in textual systems generally consists of printing important program-internal information on the screen and checking whether the program is doing something wrong.

The advantage of a well designed textual computer music systems is its flexibility. Especially if the system is built upon a general purpose programming language, the possibilities are endless.

9.2. Visual Programming

Some of the problems with textual languages might be alleviated by visual programming paradigms such as those offered by modular synthesizers and, in a more flexible way, by programs like Max or Pd. It is still necessary to learn the meanings of the program tokens (objects in this context). The language can provide help with the syntax problem, by allowing certain connections and refusing to connect non-compatible objects, actually performing the syntax check after each word that gets added.

The semantics of the language are difficult to master, especially if the language offers a reasonable amount of flexibility. Low flexibility systems like modular synthesizers are easy to understand and program. If high flexibility is needed, more effort has to be put into designing and programming a system.

Even with flexible systems, most of the time it is difficult to express algorithms that would be just a few lines of code in a normal textual programming language. This has led to a proliferation of custom written extensions to Max and Pd. The goal has not yet been reached of having a complete, self-contained system with a small set of principal objects that can be used to express almost every algorithm.

Probably the biggest advantage of visual programming languages is their interactivity. Pd, for example, allows changing the program while it is running, making it possible to develop, debug and design a patch at the same time. This is very appealing to users that do not come from a programming background and has probably been the key to the success of the Max paradigm. On the other hand, this same property sometimes leads to badly written (patched) programs.

9.3. The Future

Visual programming is certainly a more helpful model for program design, although visual languages are still far from being able to compete with textual languages. The best solution seems to be a hybrid system, where textual subroutines or objects can be embedded into a visual system. This would allow for flexibility, cross platform development and extensibility of the system, without a proliferation of custom written binary extensions.

For example scripting extensions exist for Pd, but they are not standard, and a good integration of scripting and visual language would probably require a slightly different design of the system in general.

Experience shows that scripting languages are generally easier to handle and support under different platforms than compiled languages. Theoretically though, there is no reason why a compiled language could not be just as easy to use, as long as the compilation is not too time-consuming.

Although existing visual extensions for computer music systems have not been discussed here, this is another interesting field, where several solutions are already in use. Combining audio and visuals in one system worked out pretty well in the MAX languages.

It is evident that there is still room for the evolution of new computer music systems, which should make computer music even more accessible and easier to handle.

Bibliography

1 A. Freed A. Chaudhary and M. Wright. An open architecture for real-time music Software. *In Proceedings of International Computer Music Conference*, San Francisco, 2000. International Computer Music Association.

2 Curtis Abbott. The 4CED program. *Computer Music Journal*, 5(1), 1981.

3 Xavier Amatriain. *An Object-Oriented Metamodel for Digital Signal Processing*. PhD thesis, Pompeu Fabra Univesity, 2004.

4 David P. Anderson and Ron Kuivila. Formula: A programming language for expressive computer music. *Computer*, 24:12-21, July 1991.

5 X. Arum, P. Amatriain. CLAM, an object-oriented framework for audio and music. In *Proceedings of 3rd International Linux Audio Conference; Karlsruhe, Germany*, 2005.

6 ROSS Bencina. Oasis rose the composition – real-time dsp with audiomulch. In *Proceedings of the Australian Computer Music Conference*, pages 10-12, Canberra, July 1998. Australian National University.

7 Eli Brandt. *Temporal Type Constructors for Computer Music Programming*. PhD thesis, Carnegie Mellon University, 2002.

8 Phil Burk. JSyn: Real-time synthesis API for Java. In *Proceedings of the Int. Computer Music Conference*, San Francisco, 1998. International Computer Music Computer Association.

[9] D. J. Collinge. MOXIE: a language for computer music performance. In *Proceedings of the International Computer Music Conference*, San Francisco, 1984. International Computer Music Association.

[10] P. Cook and G. Scavone. The synthesis toolkit (stk). In *Proceedings of the International Computer Music Conference*, San Francisco, 1999. Computer Music Association.

[11] R. Dannenberg. The implementation of Nyquist, a sound-synthesis language. *Computer Music Journal*, 21(3):71-82, 1997.

[12] Roger Dannenberg. A realtime scheduler/dispatcher. In *Proceedings of the International Computer Music Conference*, pages 239-242. Computer Music Association, September 1988.

[13] Roger B. Dannenberg. Arctic: A functional language for real-time control. In LFP '84: *Proceedings of the 1984 ACM Symposium cm LISP and functional programming*, pages 96-103, New York, NY, USA, 1984. ACM Press.

[14] M. Puckette E. Lindeman and E. Viara. The IRCAM signal processing work-station – an environment for research in real-time musical signal processing and performance. In *Euro Micro Proceedings*, 1990.

[15] M. V. Mathews et al. *The Technology of Computer Music*. MIT Press, Cambridge, Mass., 1969.

[16] Christopher Fry. Flavors band: An environment for processing musical style. *Computer Music Journal*, 8(4): 20-34, 1984.

[17] Albert Grf. Q: A functional programming language for multimedia applications. http://lac.zkm.de/2005/proceedings.shtml.

[18] Peter Hanappe. *Design and Implementation of an integrated Environment for Music Composition and Synthesis*. PhD thesis, University of Paris 6, 1999.

[19] Paul Hudak. Haskore music tutorial. In *Advanced Functional Programming*, pages 38-67, 1996.

[20] Julius O. Smith III. Viewpoints on the history of digital synthesis. In *Proceedings of the International Computer Music Conference*, pages 1-10, San Francisco, 1991. Computer Music Association.

[21] A. Leal L. Hiller and R. A. Baker. *Revised MUSICOMP manual. Tech. Rep. 13*. School of Music, Experimental Music Studio, University of Illinois, Urbana, III, 1966.

[22] Victor Lazzarini. The sound object library. *Organised Sound*, 5(l):35-49, 2000.

[23] Gareth Loy and Curtis Abbott. Programming languages for computer music synthesis, performance, and composition. *ACM Comput. Surv.*, 17(2):235-265, 1985.

[24] Laursen M. and Dithen J. Patchwork, a graphical language inpreform. In *Proceedings of the International Computer Music Conference*. International Computer Music Computer Association, 1989.

[25] Donald Maclnnis. Sound synthesis by Computer: MUSIGOL, a program written entirely in extended ALGOL. *Perspectives of New Music*, 7(1) :66-79,1968.

[26] Carlos Agon Marco. High level musical control of sound synthesis in open-music.

[27] M. V. Mathews and F. R. Moore. GROOVE program to compose, store, and edit functions of time. *Commun. ACM*, 13(12):715-721, 1970.

[28] Max Mathews and J. Pasquale. RTSKED, a scheduled performance language for the crumar general development system. In *Proceedings of the 1981 International Computer Music Conference*, page 286, San Francisco, 1981. International Computer Music Association.

[29] James McCartney. A new, flexible framework for audio and image synthesis. In *Proceedings of the International Computer Music Conference*, pages 258- 261, San Francisco, 2000. International Computer Music Association.

[30] Steven Travis Pope. The Musical Object Development Environment (MODE) – ten years of music software in Smalltalk. In *Proceedings of the 1994 International Computer Music Conference*, San Francisco, 1994. International Computer Music Computer Association.

[31] Miller Puckette. Music 500: A new real-time digital synthesis System. In *Proceedings of the International Computer Music Conference*, San Francisco, 1983. International Computer Music Association.

[32] Miller Puckette. The m orchestra language. In *Proceedings of the International Computer Music Conference*, San Francisco, 1984. International Computer Music Conference, International Computer Music Association.

[33] Miller Puckette. The Patcher. In *Proceedings of the International Computer Music Conference*, pages 420-429, San Francisco, 1988. International Computer Music Association.

[34] Carla Scaletti. Computer music, languages, Kyma, and the future. *Computer Music Journal*, 26:69pp, Winter 2002.

[35] E. Scheirer and B. Vercoe. SAOL: The mpeg-4 structured audio orchestra language. *Computer Music Journal*, 23(2):31-51, 1999.

[36] William Schottstaed and Rick Taube. Machine tongues XVII: CLM – Music V meets Common Lisp. *Computer Music Journal*, 18(2):30-37, 1994.

[37] William Schottstaedt. PLA: A composer's idea of a language. *Computer Music Journal*, 7(1):11-20, Winter 1983.

[38] Leland Smith. Score-a musician's approach to computer music. *Journal of the Audio Engineering Society*, 20:7-14, 1972.

[39] Andrew Sorensen and Andrew Brown. jMusic – Music composition in Java. <http://jmusic.ci.qut.edu.au/>.

[40] George Tzanetakis and Perry Cook. Marsyas, a framework for audio analysis. *Organised Sound*, 4(3):169-175, 1999.

[41] Ge Wang and Perry Cook. ChucK: A concurrent, on-the-fly, audio programmgin language. In *Proceedings of the International Computer Music Conference*, San Francisco, 2003. Computer Music Association.

ARRAYS

Arrays in Pd provide a unified way to deal with lists of numbers, treating them as either audio samples or for "control" uses. To make one, select "array" on the "new" menu. Dialogs appear to help you choose the name, number of elements, and various flags.

You can also change the array size using the "resize" message shown below. Arrays live in graphs and graphs may hold more than one array--however, graphs containing more than one array won't know how to readjust themselves automatically when the arrays are resized.

You can send messages to an array object:

0 <-- set to a constant value

/ 100

; array99 const $1

0 resize

; array99 resize $1

print size

; array99 print

Fourier synthesis (resizes table)

; array99 sinesum 64 0.2 0.2 0.2 0.2

; array99 cosinesum 64 0.2 0.2 0.2 0.2

normalize to 1 or otherwise

; array99 normalize

; array99 normalize 0.5

read a text file

; array99 read 15.file.txt

read a soundfile

read ../sound/voice2.wav array99

soundfiler

write a text file

savepanel

; array99 write $1

write a WAV format soundfile

savepanel

write $1 array99

soundfiler

...and audio signals:

tabread~ tabread4~ tabsend~ tabreceive~ tabosc4~

Audio & Video Multi-Source Mixing and Streaming: Hack the Media

Ramiro Cosentino

Introduction

It is well known and still true today that information is manipulated by private media corporations and even governmental institutions under pressure from private groups pursuing their own economic interests and oriented to the harmful capitalist system. Because they obviously seek to control the masses, there is a need for free and open ways of providing horizontal communication relationships, which could enable the development of a truly participative system of social and political construction, modeling, deconstruction and re-creation.

This is the premise on which my work is based.

Global Audio & Video Content Production and Distribution

Considering the saying "a picture says more than a thousand words" leads to the idea that audiovisual contents could result in a much richer resource than just written, legible text.
We are happily entering the post-textual era.

This is the context, from which the al-Jwarizmi project emerged (<http://al-jwarizmi.sf.net>). It is the collaborative work of around seven people, who are part of the hackitectura.net crew.

The need for participation requires converting passive users (consumers) into active producers, thus achieving a two-way or even multi-way interaction between them and others. Here is where Pd enters the scene, as a powerful and flexible tool for developing the desired platforms for these kinds of communication methods.

Art as an expression of feelings and/or thoughts, and the wish of bringing it closer to other people reinforces the statement about globally distributing contents produced anywhere around the globe. Then the conjunction of Pd plus streaming technologies seems to serve as the core basis for this communication platform and as a way of connecting people and cultures that we call: al-Jwarizmi.

As stated in the al-Jwarizmi FAQ, if we regard TV as a failed cultural project, we should seriously think about replacing TV and even radio, or at least re-creating them with the help of new technologies.

However, this replacement or re-creation is mostly intended to re-think the concept of these mass communication tools, and direct their use towards a cultural resource for learning and promoting positive social changes.
That means this project does not really aim to completely suppress TV or radio, but to achieve a better interfacing of Internet streaming technologies with them, to reach a wider audience where the Internet cannot or bandwidth is not sufficient.

We can imagine tuning into a netradio from Angola from the studios of a regular FM station somewhere else and the stream being re-broadcasted over FM waves locally on another continent.

This interfacing will help the treatment of well known issues of the digital divide and thus help develop solutions.

Worldwide Distributed Participation

In my honest opinion, no one should be forced to participate, but everyone must have access to the possibility, in order to achieve a global balance. Otherwise only some will take part and make decisions for others.

In CorveraHack event (Asturias, Spain 2003) we experimented with provisioning a webform posting with a PHP script, which sent us text that the users were able post to the NOC (Network Operations Center). There it was merged with the video render windows which were being streamed out.

In short: people watching the audiovisual stream could post feedback through a webform, and a few seconds later (because of net delays) they could see it on their streaming window.

Re-Broadcasting Others' Contents

To ensure the freedom of information, as stated in the first item of the hacker ethic, we propose that the same idea is to be applied to media.

"The belief that information-sharing is a powerful positive good, and that it is an ethical duty of hackers to share their expertise by writing free (libre) software and facilitating access to information and computing resources wherever possible."
(taken from <http://en.wikipedia.org/wiki/Hacker_ethic>)

Facilitating the free flow of media for topics of interest will support their conceptors, produc-

ers and distributors, so I strongly believe in the need for this idea of redistribution as a way of freeing the contents, thinking about them as pieces of information.

Audioflow & Videoflow Patches Overview

I have made two patches for Pd with the basic function of mixing several sources into one final mix to play locally and to stream it out to the Net.

Both of them are able to send the final mix directly to other peers running Pd and any patch capable of receiving peer2peer audio (using [mp3streamin~]) or video (using [pdp_i]). [Audioflow] and [Videoflow] can also receive peer2peer audio and video respectively.

In a typical environment where there are hosts connected to the same LAN or other fast network, each [Videoflow] and [Audioflow] can be hooked together with others. This results in a wide network of distributed content production.

[Audioflow] is also capable of tuning into two MP3 and two OGG streamings at the same time and mixing them with another source, such as the aforementioned p2p, hard disc OGG audio files and live soundcard inputs.

[Videoflow] is very similar to [Audioflow].

Several experiments with mixing multiple audio and video sources and also rebroadcasting remote streams have been done by myself and others. And better yet, a lot more will come :)

This paper was written as an overview of my talk/presentation at the Graz Pd~Convention, which took place from September 27th to October 3rd, 2004.

MORE ON ARRAYS

Arrays have methods to set their values explicitly; to set their "bounds" rectangles, to rename them (but if you have two with the same name this won't necessarily do what you want) and to add markings. To set values by message, send a list whise first element gives the index to start at. The second example sets two values starting at index three. Indices count up from zero.

```
;
array98 0 -1 1 -1 1 -1 1 -1 1 -1
```

```
;
array99 3 -0.5 0.5
```

renaming an array:

```
;
array99 rename george
```

```
;
george rename array99
```

setting the bounds rectangle:

```
;
array99 bounds 0 -2 10 2
```

```
;
array99 bounds 0 -1 5 1
```

adding x and y labels: give a point to put a tick, the interval between ticks, and the number of ticks overall per large tick.

```
;
array99 xticks 0 1 1
```

```
;
array99 yticks 0 0.1 5
```

adding labels. Give a y value and a bunch of x values or vice versa:

```
;
array99 xlabel -1.1 0 1 2 3 4 5
```

```
;
array99 ylabel 5.15 -1 0 1
```

You can also change x and y range and size in the "properties" dialog. Note that information about size and ranges is saved, but ticks, labels, and the actual data are lost between Pd sessions.

last updated for release 0.33

A Divide Between 'Compositional' and 'Performative' Aspects of Pd [*]

Miller Puckette

In the ecology of human culture, unsolved problems play a role that is even more essential than that of solutions. Although the solutions found give a field its legitimacy, it is the problems that give it life. With this in mind, I'll describe what I think of as the central problem I'm struggling with today, which has perhaps been the main motivating force in my work on Pd, among other things. If Pd's fundamental design reflects my attack on this problem, perhaps others working on or in Pd will benefit if I try to articulate the problem clearly.

In its most succinct form, the problem is that, while we have good paradigms for describing *processes* (such as in the Max or Pd programs as they stand today), and while much work has been done on *representations of musical data* (ranging from searchable databases of sound to Patchwork and OpenMusic, and including Pd's unfinished "data" editor), we lack a fluid mechanism for the two worlds to interoperate.

1 Examples

Three examples, programs that combine data storage and retrieval with realtime actions, will serve to demonstrate this division. Taking them as representative of the current state of the art, the rest of this paper will describe the attempts I'm now making to bridge the separation between the two realms.

1.1 CSound

In CSound[2], the database is called a *score* (this usage was widespread among software synthesis packages at the time Csound was under development). Scores in Csound consist mostly of 'notes', which are commands for a synthesizer. The 'score' is essentially a timed sequence. A possible score might be as shown:

```
i1 0 1 440
i1 1 1 660
i1 2 1 1100
e
```

[*] Reprinted from the 1st International Pd~convention, Graz, 2004: <http://puredata.info/community/projects/convention04/>

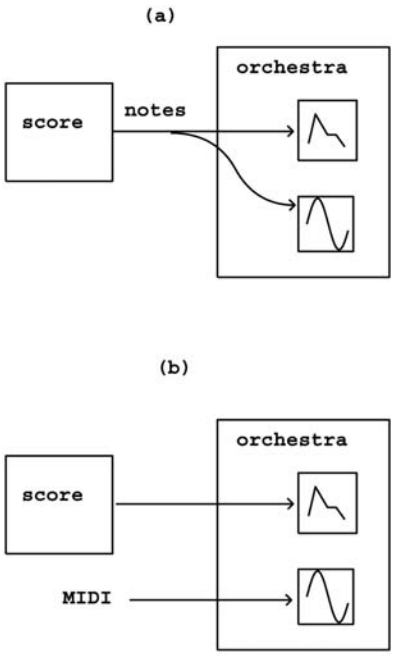


Figure 1: A Csound performance: (a) score and orchestra; (b) using real-time control via MIDI.

A Csound performance works as shown in Figure 1. Part (a) shows the "classical" performance configuration, in which parameters in the notes update synthesis control values, each note acting at an effective time also calculated from the note's parameters.

Part (b) of the figure shows how to use real-time inputs (here from MIDI messages) in a Csound performance. The real-time inputs are simply merged with the (pre-scheduled) notes. In effect, there is no facility for intercommunication between the two control streams; they simply affect different variables in the orchestra, and the orchestra's audio output is controlled by the union of the two sets of variables.

1.2 Patchwork and OpenMusic

Michael Laursen's Patchwork program[4] and its descendant, OpenMusic, by Carlos Agon and Gérard Assayag[1], offer a much tighter integration of data. Figure 2 shows a simple OpenMusic patch.

The semantic of OpenMusic (and Patchwork) is one of demand-driven dataflow. Each object is essentially a function call, which recursively evaluates its inputs, precisely as a Lisp form is evaluated. Compared to Pd, the relationship between data and process is reversed. There is no notion of real-time events or even of real time itself; rather, the contents of a patch are static

data. The paradigm gets its richness from the fact that the data types (which in the pictured example are just numbers) can in general be any lisp data structure, and so can easily describe whole sequences such as a Csound score.

OpenMusic supplies a sequencing function which, given a sequence as an argument, plays the result out the machine's MIDI port or sends it to a software synthesizer. The data managed in the patch itself are all entirely out-of-time; the sequencer's function of putting the data in time is a primitive operation. The lisp object or objects which hold rhythms, pitches, and even timbres are queried by the sequencer which does the data mining as a black box.

This is ideal from the composer's point of view, since the creation of a musical score is essentially an out-of-time activity. But performers will have little use for OpenMusic since, in live performance, the instrument doesn't query the

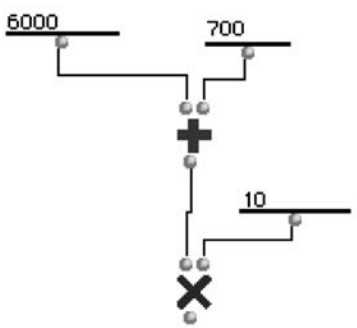


Figure 2: An OpenMusic patch (borrowed from IRCAM's documentation).

performer, but rather, the performer sends messages to the instrument. This is the Pd (and Max) organization, the reverse of that of OpenMusic.

1.3 Max and Pd

In Pd (the third example), the fundamental transaction goes in the direction favored by the performer. This idea goes back to Max, and that orientation might have been the most important single reason that Max and Pd are in wide use today. But as noted before[3, 5], the message-sending paradigm does not fundamentally lend itself well to storing and retrieving data. One is almost forced to set data aside in containers – databases, essentially – and to use a coterie of accessor objects to store and retrieve data under real-time, message-passing control.

Max's approach to data is both simple and evasive: special data-container objects such as table, qlist, etc. are provided; the data are essentially hoarded inside the container objects, and for

each kind of container object, a particular *ad-hoc* approach is taken to its storage, its editing, and its interfacing with the rest of the patch.

Retrieval (the great majority of database transactions!) is the worst fit with Max because messages don't have return values; the retrieved data must be sent as a separate return message. This leads to much misery for Max users.

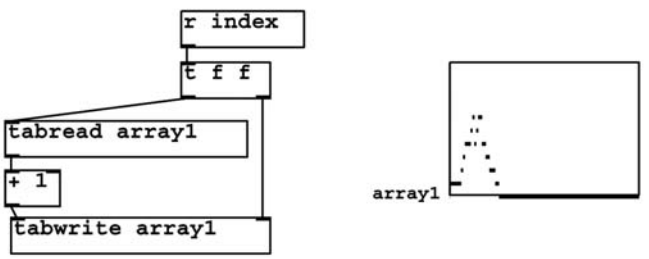


Figure 3: Incrementing an array element. The receive object can be sent integers to specify which element to increment.

Pd faithfully recreates the data-storage paradigms of Max, but in addition the design of Pd includes a more advanced paradigm that might eventually replace the Max one.

The original, defining idea behind Pd was to remove the barrier between eventdriven real-time computation (as in Max-style message passing) and data (as in points of an array or notes in a score). In Pd the two (object boxes and data structures) can easily coexist in a single window. This promiscuity, however, does not in itself make the functional objects and the data intimately connected. In fact, in the present design, data access still has to be done through a suite of accessor objects. It is far from certain that Pd will, in the end, relieve the Max user's misery.

2 Data-Plus-Accessor-Object Design Model

An example of Pd's data-plus-accessor arrangement is that maintained by floatingpoint arrays (either graphical or via the table object), and the suite of objects tabread, tabwrite, and all their relatives. For example, Figure 3 shows how to use accessor objects to increment a variable element of an array (you would do this to make a histogram of incoming indices, for example.) Here the task is straightforward, and the separation of the storage functionality of the actual "array1" from the accessor objects is not particularly troublesome.

Moving to a more interesting case, we now build a patch to do something corresponding to this using the (still experimental) "data" feature of Pd[6]. For completeness we give a short summary here, which will serve also to introduce the central example of this paper.

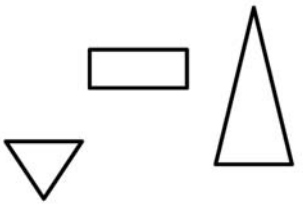


Figure 4: Two data structures in Pd.

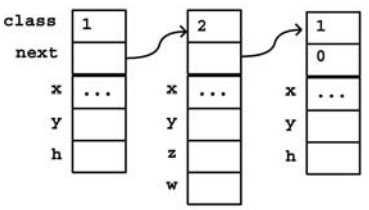


Figure 5: Possible structure for the objects in Figure 4.

Pd's "data" are objects that have a screen appearance; many such objects can be held in one Pd canvas. The canvas holds a linked list of data. A datum belongs to some data structure, which is defined by a patch called a *template*. The template also defines how the data will look on the page. Lists of data are heterogeneous; a canvas in Pd can hold data with many different structures. Figure 4 shows a canvas with three data objects. They belong to two types: two triangles and one rectangle.

As data structures, this list could appear as shown in Figure 5. The elements of the list need not all have the same structure, but they have a "class" field that determines which of the several possible data structures the element actually belongs to.

```
struct struct1 float x float y float h

filledpolygon 999 0 2 0 0 20 h 40 0
```

Figure 6: Pd definitions of the data structure for the triangles in Figure 4.

The data structures are defined by struct objects. Figure 6 shows the struct object corresponding to the triangles in Figure 4. The canvas containing the struct object may also contain *drawing instructions* such as the drawpolygon object. This object takes three creation arguments to set the interior and border colors and the border width (999, 0, and 2), and then any number of (x, y) pairs to give vertices of the polygon to draw; in this example there are three points and the structure element h gives the altitude of the triangle.

For clarity, and for the sake of comparison, we'll consider C and Pd approaches to defining and accessing the data in parallel. In C, the definitions of the two data structures might be as shown in Figure 7. In order to be able to mix the two structures in a single linked list, a common structure sits at the head of each. This common structure holds a whichclass field to indicate which structure we're actually looking at, and a next field for holding a collection of these structures in a linked list.

Now we define a task that might correspond to that of Figure 3. Suppose, given an integer *n* and a linked list of data structures, we wanted to find the *n*th occurrence of struct1 in the list and increment its h slot. (Note that incrementing the h slot of an instance of class2 wouldn't make sense.) A C function to do this is shown in Figure 8. This is an inherently more complicated problem than that of Figure 3; there, a corresponding piece of C code might simply be:

```
array[n] += 1;
```

Instead, we have to make a loop to search through the heterogeneous list, checking each one if it belongs to struct1, maintaining a count, and when all conditions line up, incrementing the h field.

An equivalent Pd patch is shown in Figure 9. The loop is managed by the [until] object. The top [trigger] initializes the [f] and [pointer] objects (corresponding to `count` and `ptr` in the C code.) The messages, "traverse pd-list" and "next", correspond to the initialization and update steps in the C loop; the two possible exit conditions of the loop (the check on `ptr` and the break in the middle) are the two patchcords reaching back to the right inlet of the [until] object.

The [pointer struct1] object only outputs the pointer if it matches "struct1"

```
struct anyclass                 /* common header in both structs below */
{
#define CLASS1 1
#define CLASS2 2
    int c_whichclass;           /* whether CLASS1 or CLASS2 */
    struct anyclass *c_next;    /* next in linked list */
};

struct class1                   /* first one */
{
    struct anyclass c1_header;  /* common part */
    float c1_x;                 /* part that is specific to class #1 */
    float c1_y;
    float c1_h;
};

struct class2                   /* second one */
{
    struct anyclass c2_header;
    float c2_x;
    float c2_y;
    float c2_z;
    float c2_w;
};
```

Figure 7: A C equivalent for the data structures of Figure 4.

```
extern struct anyclass *thelist; /* externally defined head of linked list */

void incrementclass1(int n)      /* increments the nth "class1" in list */
{
    struct anyclass *ptr;
    int count;
        /* search the list, stopping when pass "n" or run out of items */
    for (count = 0, ptr = thelist; ptr; ptr = ptr->c_next)
    {
        if (ptr->c_whichclass == CLASS1)       /* check class of this item  */
        {
            if (count == n)                         /* if it's the nth one: */
            {
                ((struct class1 *)ptr)->c1_h += 1;  /* increment "h" field */
                break;                              /* and we're done */
            }
            count++;                                /* increment count */
        }
    }
}
```

Figure 8: A C function to increment "h" for the nth occurrence of class1.

(corresponding to the first if in the C code.) The [sel] object in the patch corresponds to the check whether `count and n` are equal in the C code. The remainder, below the second [pointer] object, is much the same as in Figure 3.

3 Discussion

The Pd patch looks more complex than the C code. One possible reason for the complexity is the difficulty of sequencing actions in Pd patches, which lack the natural sequentiality of a text programming language like C. Another is the relative lack of names; only three names (other than Pd class and message names) appear in the patch ("n", "struct1", and "h"), compared to eight in the C code (`thelist`, `n`, `ptr`, `count`, `c_next`, `c_whichclass`, `CLASS1`, and `c1_h`). Of these, two of the Pd names ("n", and "pd-list") might be considered to act as variables, compared to four of the C names.

The complexity of the patch needed to accomplish this task might be reduced somewhat if either the [value] and/or [expr] objects were extended to deal with pointers. For instance, a [value ptr] object (unsupported in the current version of Pd) could hold the output of the first pointer object in readiness for the incrementing step at bottom. So far, though, mockups using features such as this have not been observed to reduce the number of objects and lines in patches equivalent to the one shown here.

The [expr] object could conceivably handle data structures and slots with the addition of a few C-like constructs, and could also be fixed to set and retrieve the contents of value-style variables. This would cause the Pd and Max versions of [expr] to deviate from one another (they currently share the same code, maintained by Shahrokh Yadegari[7]. In general, it seems problematic to lean in too fundamental a way on [expr] as the fundamental mechanism for getting and retrieving data.

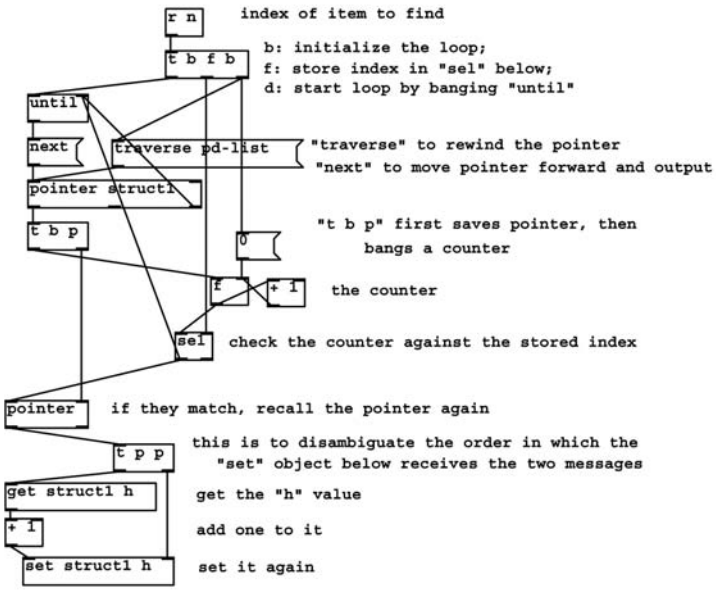


Figure 9: Pd patch to search for the *n*th instance of class1 and increment its value of h.

On a more general plane, the relationship between the data and the code that accesses it is the same in Pd as it is in C. One wishes that the functionality could somehow reference the look and feel of the data themselves, or possibly even be built into the template patch (as methods go with class definitions in C++.) So far no model has emerged that accomplishes this smoothly.

Another aspect of the question, not touched on in this paper, is the utility of somehow catching user operation (with mouse and keyboard, perhaps among other ways) with the graphical data. There should be a way to provide hooks to data when certain operations are carried out on them. Perhaps this should be realized as a way of fielding Pd messages (via [pointer] objects?) sent to objects to get or set their state.

The data structure accessor objects could easily be back-compatibly replaced or augmented with others if a clean design can be found for getting and setting the data. This "data" feature was the original motivating force behind Pd's design; it is interesting that it now appears likely to be the last aspect of Pd to be defined.

References

1 Gerard Assayag et al. Computer assisted composition at ircam: From patchwork to openmusic. *Computer Music Journal*, 23(3):59-72, 1999.

2 Richard Boulanger, editor. *The Csound book*. MIT Press, Cambridge, Massachusetts, 2000.

3 P. Desain and H Honig. Letter to the editor: the mins of max. *Computer Music Journal*, 17(2):3–11, 1993.

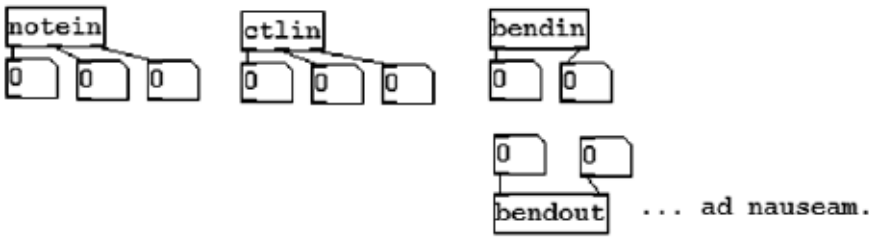
4 Mikael Laurson and Jacques Duthen. Patchwork, a graphical language in preform. In *Proceedings of the International Computer Music Conference*, pages 172-175, Ann Arbor, 1989. International Computer Music Association.

5 Miller S. Puckette. Max at 17. 26(4):31-43, 2002.

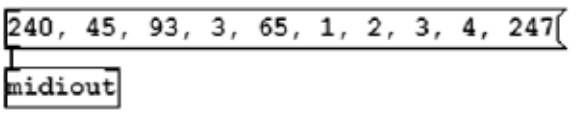
6 Miller S. Puckette. Using pd as a score language. pages 184-187, 2002.

7 Shahrokh Yadegari. A general filter design language with real-time parameter control in pd, max/msp, and jmax. In *Proceedings of the International Computer Music Conference*, pages 345-348, Ann Arbor, 2003. International Computer Music Association.

Pd offers input and output objects for MIDI:



You can format your own SYSEX messages as shown:



and receive SYSEX via: sysexin

updated for Pd version 0.34

Creating from informal communication and Open Source

Werner Jauk

Art as process necessitates not only the communication of "ephemeral products" as a method of creating but more consequently the communication of the methods of creation itself. The development of tools is thus a part of the process art as creating from communicative behavior.

These assumptions follow the model of communication in the community of science as a method of gaining knowledge – the authorship is thus the context of the text. Scientific methods attempt to minimize the subjective influences of the actors in the research process on knowledge – like the discussion in science of the subjective worth of the participants in collective as well as collectivizing processes (cf. de Kerkhove 1995), a joint action should be inserted into the discussion.
Due to the deconstruction of the interconnection of product and creator/owner, collective and informal creation is part of a socio-political impetus toward informalization and thereby part of a horizontalization process – in addition to technical/economic availability, psychological availability could broaden this horizontalization.

By definition, Open Source focuses on access to and the supply of source code for the controlled collective development and adaptation of software for the generation of mathematical processes but also the development of communication interfaces between humans and machines – the definition also affects statements about accessibility, thus about the people acting and their tools for communication – it addresses political correctness only generally.

> Open Source[1] doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:
>
> 1. Free Redistribution
> 2. Source Code
> 3. Derived Works
> 4. Integrity of The Author's Source Code
> 5. No Discrimination Against Persons or Groups
> 6. No Discrimination Against Fields of Endeavor

---

1 <http://www.opensource.org/docs/definition.php>
The Open Source Definition Version 1.9
Origins: Bruce Perens wrote the first draft of this document as "The Debian Free Software Guidelines", and refined it using the comments of the Debian developers in a month-long e-mail conference in June, 1997. He removed the Debian-specific references from the document to create the "Open Source Definition." Copyright (c) 2005 by the Open Source Initiative

7. Distribution of License
8. License Must Not Be Specific to a Product
9. License Must Not Restrict Other Software
10. License Must Be Technology-Neutral"

The availability of the conditions for access is specific to Open Source communities: psychological availability is a subordinate theme; technical and legal/economic aspects are in the center of attention.

This implies the participation of those with specific knowledge in Open Source communities and explicitly excludes those without this knowledge. The acquisition of specific knowledge is correlated with the habituation to certain attitudes, values, and behavior patterns. Knowledge – formulated in symbolical languages – is not free of cultural imprints like the implications of cultural imprinting. This still necessary elite behavior stands observably in opposition to the ideological statements of participants who speak of horizontal structures and also transfer them to political systems; in comparison with the hierarchical (in the sense of the authority) participation in helping work for exclusive or primary uses, Open Source is intended to be an alternative process of production in an alternative economic system.

The approach to the general and the ideological excess of Open Source is not only dependent on technical/economic conditions but also on individual, socio-psychological, and historic/political values – these are in part the results of present social living arrangements and their economic and political basis.

I – Individuality, the Power of Personal Borders

One's own level of activation is a physiological condition of a personality trait which expresses itself in social behavior: introverts are people for whom their own high activation is enough to reach the activation level they prefer. Extroverts raise their small activation level by outer activities which extend into the area of social behavior; they appear more communicative (cf. EYSENCK 1967, 1990). In conjunction with cognitive styles of processing information (e.g. open vs. close-mindedness), individual predispositions are mentioned as a potentiality for human social positioning – the interaction of individual and social processes is ultimately varied.

Organizational and group psychology recognize a distinction in the efficiency of group achievement depending on the kind of task in formal and informal group structures (cf. Weinert 1998), in groups with central creation and in those with local (self) organization.

In general, problems which can be solved linearly and logically are solved more efficiently – that is to say more quickly – in formally structured groups than those tasks whose solutions are less able to be determined in advance or who ultimately are insolvable through additive

individual performance but are dependent on the specific (also emotional) strengthening of individual performance.

For a first-class solution of tasks, informal groups additionally require the solution to the problem of the structuring of the group and thus the paths of information transfer (Guetzkow & Simon 1955). In the second kind of task defined, structuring and information transfer are part of the problem to be solved and adjust in relation to one another – preordained structures inhibit unintended solutions. The structuring of the group and the structuring of the problem are implications of collective generation which mutually refer to one another. Informal groups are formed through communication. In contrast to a mechanistic understanding of information transfer (cf. Shannon & Weaver 1949), communication occurs here not reactively (cf. Popper 1975) but interactively (Bales 1950, Jauk 1995).

What may play a smaller role within certain confinable tasks and problem solving strategies because objective requirements outshine subjective abilities and interests becomes an explicit creative problem for the group and its creation in informal structuring of vaguely circumscribed groups and their indifferent function: the individual readiness to assimilate into the group, to assume the development of dynamic roles in the process of forming groups; the rank of the members of a group seems to outshine the network structure concerning their efficiency (Moore, Johnson & Arnold 1972). Individual integration is partly determined by the motivation to participate, which is determined by the attractiveness of the group to the individuals.

Thus the striving for power and the submission to power are opposite poles of individual interests. In the formation of informal groups, both dispositions do not necessarily lead to a position as leader (as regards emotion or content) or followers. General acceptance of an individual person's behavior determines his or her position, popularity, and the attribution of competence. Reversals are born by counterpoles and outsiders in regard to content and emotion; their meaning is accompanied by a change in direction of the content and of the strategies for solution.

II – Enculturation, the Power of Cultural Inscription

Apart from dispositions determined by socio-psychology and personality, historical burdens (cf. Gadamer 1960) and political imprinting as determinants of individual behavior are of interest in regard to the ability to create informal structures.
Science picks out this kind of creation as a central theme in parallel to art in the fifties. Self-organization as an idea of W.A. Clark and G.B. Farley is stamped by the liberation from mechanistic limits of determination. "They recognized, that operators in a closed relationship are somehow stabilized and – still without knowing a theory of recursive functions or of peculiarities – observed the phenomena that certain closed systems develop stable forms of behavior

after a certain amount of time." (Heinz von Foerster und Bernhard Pörksen, 1998, p. 92) Self-organizing systems are complex, which means that their parts are interconnected with each other through reciprocal, permanently changing relationships; they are self-referential, which means each behavior of the system has an effect on itself and becomes a starting point for further behavior; they are redundant, which means they know no separation between organizing, creating, or managing parts; they are all potential creators without hierarchy; they are autonomous, which means interactions in the system are only determined by this alone. Bales (1950) describes the self-organization of (socio-psychological) groups concerning their organization and the content they treat, a model which becomes fruitful later in the interactive and communication arts and separates the cybernetic from a communication theoretical view.

Adorno (1947) combines informal organization forms and on the other hand collective free improvisation – both tested in the fifties – in his definition of the communicative art form of music as a (historically altered and therein) formalized form: polyphony is the objectivation of the we. Pop makes this avant-garde popular in the sixties; the group determined form of the "we" in music entered into public consciousness as a collective creation.
The visual arts postulated cooperation, which is typical of music. Within music, collective creation was placed as a methodical step toward *get together*[2], to *art as teamwork* with the structuring of informal communication in free jazz of the fifties. *Self-organizing systems* and *horizontalization* were hackeresque reactions to the action affirming hierarchical structures of the previous generations which for the time being were being put into practice in artistic life (Jauk). Autonomy led further to self-determination and finally to a self-organizing structuring.
Collective decentralized creation has its avant-garde long before the tools supplied over electronic networks which had favored the fiction of a horizontal net-art and with it of a horizontal society. The Graz Forum Stadtpark – founded in 1959 – is an early example of a collective from the beginning of the sixties. Along with classical and the at that time early media art (photography), science is one of the institutions in the multidisciplinary collective. The Forum is based on interdisciplinary "self-organization […] which at that time was not yet recognized as a political achievement." (Mixner 1975 p. 15)

Nevertheless: "The […] hoped for revolution in interpersonal communication – even among artists – has not occurred. The high costs of hardware and communication rates are only one part of the problem – more decisive are the lethargy and inertia of 200 years of industrial culture and its consumerist repercussions. No one in our culture, artists included, is trained or encouraged to let others share in his or her creativity […]. The capability for shared creative activity is a necessary precondition for the interactive use of communications technology. We are all used to the producer/consumer relationship of the manufacture of things for consumption by others." (Robert Adrian X 1989, p. 147)

---

2 Cf. Get Together. Kunst als Teamwork, hg. von Kunsthalle Wien. Wien, 1999.

III – Tools for Communication, the Power of Symbols

The high valuation of the ability to represent nature symbolically – along with a lower assessment of the natural and with it the physical, however – lies deep in the consciousness of self-image in European culture. This cultural attitude implies the priority of the use of certain tools for communication.
Efficient formal languages with little lack of clarity of expression (in redundancy as well as in ambiguity) were used to solve relevant problems adequately.
Net specific interactions and forms of communication are at a "low" level of formal languages and thereby are often phrased in the language of commands.
The use of special languages can only conditionally be legitimized as an adequate form of communication by disciplinary measures. It is fundamentally the "trauma" of a horizontalization stipulated by ideology; it is ultimately an elitist form of communication and rules out open communication and thereby a generalization used in many cases of the Open Source platform Pd, Pure Data, as public domain.
Ideological generalizations about open forms of communication on the net as creation from communication presuppose the general possibility for communication.

Open Source is directed against commercial systems which on their part use that general accessibility, psychological availability. They are optimized to be user-friendly and to appeal to the largest possible market and hence use intuitive forms of communication and icon-oriented – ultimately self-explanatory – tools.

Experimentally documented by Clynes (1977), body-oriented forms of interaction with high universality and an understanding which reaches across cultures are seldom if at all used. They have shown themselves to be not very fruitful on the commercial (game) market. Perhaps they are unfamiliar because they are too distant from previously experienced forms of interaction and thus are not accepted. As a direct physical manner of expression, they are ideologically devalued in comparison with the high cultural form of symbolic representation – precisely in their directness, they allow not only the description of emotions, not only their iconic representation but also the direct communication of the condition of the emotions. Clynes (1977) names those simple interfaces which express emotional qualities in the narrower sense of the word *sentics*.

Basal synesthetic aspects of the components of evaluation and of the components of intensity and activity are understood via these sensitive forms of expression which specifically adjust to the form of the physical movement. Intercultural comparisons cover the forms of communication which reach across cultures and are an unleashing of such implications shaped by empirical "enlightened" cultures – idealistically shaped forms do away with humble, direct physical expression and general pictorial communication and show the grammatical arrangement of syntactic events of arbitrary signs as the form of communication with the highest value of information, the highest restriction on uncertainty. They are cultural implications

and the antithesis of transcultural communication aside from economic and political claims to power.
A European conception of culture outshines non-European cultures and reduces globalization to a monoculture of Eurocentric origin.
The form of communication primarily used is loaded with socio-political traditions; both become habits individually; they are ultimately those "handicaps" which refuse general participation.

If the alternative position mixes up the general public with the corresponding popular slogan of mediocrity, isn't general availability a precondition of general codetermination? Average measure is not judged to be mediocrity, but is an implication of the turning to the general instead of the elite observation of the particular – a consciousness turning toward the general is a political and scientific position that does not attempt to give reasons for the universal validity in the particular.
A social democratic position melts with a science based on this scientific mindset in the Vienna circle as well as in CCCS (cf. Sandner 2001, 2002) – thus everyday life moves into the center and culture is understood as culture from below (Blaukopf et al 1983).

In comparison to the motive possibly giving reasons for and supporting instruction in modern science in order to bring certainty into our lives and its accomplishment, tolerance stands for something else – other cultural ways of thinking.
The quest for certainty nurtures the supremacy of those in the know and the danger of false certainty (able to be controlled methodically in part), of false precision (an artifact of scientific methods), and leads from scientific knowledge into ideological positions due to personal inadmissibility.
In "How I See Philosophy", Popper (1975/77) calls for overcoming power-obsessed vanity and gives arguments for living with uncertainty in the face of the danger of undemocratic prescriptions of individuals allegedly in the know, in the face of the experience of many who feel.

Living with uncertainty is a political precondition for the recognition of political alternatives, living with recognition instead of openness.
In principle, the generalization of Open Source to general political positions seems to be tied to personal susceptibilities which are not congenital and to personality forming processes oriented toward the imbalance of power beyond the handed down cultural conditions of industrialization and to fail because of this internalization – the power to do this may be the striving for certainty. A unidirectional way of thinking which progresses rationally and linearly – corresponding to the safeguarding of personal ways of thinking – increasingly gives way to the quality of experience of pararealities.

Language-oriented cultures formalize serial thinking in monocausal referencing. References are logically understood as true and false; the qualities of the criteria defining the goal are often undiscussed ideological premises. Holistically oriented cultures permit at the same time various qualities of determination, the criteria that define the goals, and the self-generation of

the goal as well. If speech-oriented cultures are visually dominated, then holistically thinking cultures seem to be close to auditorily dominated (cf. McLuhan 1995; Thall 1996). These alternative patterns of life of digital culture, which defines itself as musicalized and hedonistically newly defining itself (Jauk 2003) – a culture of that which can be deliberately done using codes distinguished from one of that which is naturally given – are formalized in a logic of the auditory (Jauk 2000)[3].
The idea of culture as the symbolic representation of reality (cf. Cassirer 1964) places the written word in the highest position of the means of communication and by focusing on the what of the message factors out the how, an oral quality which enters only slightly into the written language by means of punctuation marks and grammatical references. Iconic forms of communication are deemed too inexact; their advantage lies precisely in the ambiguity and the inclusion of connotative qualities. Physical forms of communication are ascribed to everyday and thus hardly noteworthy communication – emotional and thus social references among those acting are indexed optimally therein.
Our idealistic culture demands of itself to be a rationally controlled representation of reality and its generation on the level of symbols. Based on a semiotic understanding of culture, it ascribed communication via signals to lower beings. Today this is increasingly emerging as an alternative form of communication which permits uncertainty and ambiguity and regards emotionally determined forms. Seemingly "lower" levels of communication which implicitly or explicitly exclude the written language and highly formalized language prove to be alternatives for coping with life: such forms of communication are essential parts of languages of other cultures whose own understanding defies this rationality as an a priori feigned control of reality.

Pop has in the meantime become a global culture not created by the mass media but surely supported by it. Considered systematically, pop as a physical culture (Wicke 1998, 2001) has a high likelihood of being a global culture. If a push to be less formal has occurred through pop as physical culture (cf. Browne 2000), an informalization will thus occur from the unmediatized, direct physical forms of expression of an intuitively comprehensible communication, one which will furthermore encourage horizontal forms of society – despite all the free market and political interests allied with pop or (as McLarenesque punk showed) an undermining, hackeresque use of them.

[3] Furthermore, a musicalization is seen in the dynamization (Rötzer 1991, Charles 1989, Flusser 1985) to the racing standstill (Virilio 1992), to all-at-onceness (McLuhan 1995; cf. Thall 1996).

IV – Epilogue

Basically, the visually controlled body-environment interaction (Gibson 1982) has created a mechanistic view of things (Levy 2000) which finds its formalization in the logic of language. It seems there is a return of digital culture (JAUK 2003) to phylogenetically older auditorily controlled body-environment interactions, formalized in the collective form of communication of music and a theoretical guideline: not to negate a visual one but to extend this and thereby to produce a cultural interface to other cultures. A body-environment interaction at the same time controlled by several senses and its specific formalization in parallel logics leads to a thought and its communication by means of tools which transgress the boundaries established by culture.
Technological limitations of networks do not allow the legitimization of a global monoculture of the word and a linear logic. The instrumentalization of emotional physical expression and its communicative quality in music is a model for a hedonistically ordered interaction in electronic as well as transcultural space. Aside from all ideas about the ideologically charged stereotype of music as a language which joins people together, the paradigm of music as a particular cultural transformation of slightly mediatized basal forms of communication of the sounds that express emotions (Knepler 1977) and the behavior of expressions (Blacking 1977) holds the potential within itself to be an interface to a transculturality – as a form of expression and at the same time as a presentative sign (Langer 1953), as a physical expression valid across cultures (Clynes 1977).

Aside from the derived generation of knowledge, the adopting of ways of life with parallel realities rests on the personal capacity to give up the attempt for certainty/security in favor of live/lived uncertainty/insecurity. It pays to learn this as a cultural life technique which permits a transcultural way of life.
Living with uncertainty is thereby not a matter of altruism. In the manner of social evolution, one's own survival is the motivation and the reason for the inadequacy of coping with the modern way of life.
In global communication, being open is not only a matter of the source, of the means of communication; it is a matter of the actors and their own survival.

References:

Adorno, T. W. (1958). *Philosophie der Neuen Musik* [1947]. Frankfurt am Main: Suhrkamp.
Bales, R. F. (1950). *Interaction process analysis*. Cambridge.
Blacking, J. (1977). Towards an Anthropology of the Body. In: J. Blacking (Hrsg.), *The Anthropology of the Body* (S. 1-28). London: Academic Press.
Blaukopf, K., Botinck, I., Gardos, H. & Mark, D. (1983). *Kultur von unten. Innovationen und Barrieren in Österreich*. Wien: Löcker.
Browne, R. (2000). Journal of Popular Culture. In: *Comparative Studies in the World's Civilization*, 34/1, 157.

CASSIRER, E. (1964). *Philosophie der symbolischen Formen*. Darmstadt.
CHARLES, D. (1989). *Zeitspielräume. Performance Musik Ästhetik*. Berlin: Merve.
CLYNES, M. (1977). *Sentics: The Touch of emotions*. New York: Anchor Press/Doubleday.
COLLINS, Barry E.: A social psychology of group processes for decision-making / Barry E. Collins ; Harold Guetzkow. – 3. print. New York, NY [u.a.]: Wiley, 1966. – X, 254 S.
EYSENCK, H. J. (1967). *The Biological Basis of Personality*. Springfield, IL: Charles C. Thomas.
EYSENCK, H. J. (1990). Genetic and environmental contributions to individual differences: The three major dimensions of personality. In: *Journal of Personality*, 58, 245-261.
FLUSSER V. (1985). *Ins Universum der technischen Bilder*. Göttingen: European Photography Verlag.
FOERSTER, H. v. und PÖRKSEN, B (1998) *Wahrheit ist die Erfindung eines Lügners*
GADAMER, H.-G. (1986). Wahrheit und Methode. Grundzüge einer philosophischen Hermeneutik [1960]. In: *H.-G. G. Gesammelte Werke, 10 Bde. Hermeneutik I*. Tübingen: Mohr.
GIBSON, J. J. (1982). *Wahrnehmung und Umwelt*. München: Urban & Schwarzenberg.
GUETZKOW, H. & SIMON, H.A. (1955) The impact of certain communication nets upon organization and performance in task-orientated groups. In: Management Science 233–250.
JAUK, W. (1995). Interaktivität statt Reaktivität. In: H. Leopoldseder & C. Schöpf (Hrsg.), *Prix Ars Electronica 95* (S. 23-27). Linz.
JAUK, W. (2000). The Auditory Logic: An Alternative to the "Sight of Things". In: H. Nowotny, M. Weiss & K. Hänni (Hrsg.), *Jahrbuch des Collegium Helveticum* (S. 321-338). Zürich: Hochschulverlag Ag an der ETH Zürich.
JAUK, W. (2003). The Transgression of the Mechanistic Paradigm – Music and the New Arts. In: *Dialogue and Universalism*, 8-9, 179-186.
JAUK, W. (2004). Kunst und (Natur-)Wissenschaft. Aspekte der Theorien der Neuen Künste. In: In: E. List & E. Fiala (Hrsg.), *Grundlagen der Kulturwissenschaften. Interdisziplinäre Kulturstudien* (S. 225-244). Tübingen.
KERCKHOVE, D. de (1995). Kunst im World Wide Web. In: H. Leopoldseder, C. Schöpf (Hrsg.), *Prix Ars Electronica 95* (S. 37-49).
KNEPLER, G. (1977). *Geschichte als Weg zum Musikverständnis. Zur Theorie, Methode und Geschichte der Musikgeschichtsschreibung*. Leipzig: Reclam.
LANGER, S. (1953). *Feeling and Form. A Theory of Art Developed from Philosophy in an New Key*. London: Routledge.
LÉVY, P. (2000). Die Metapher des Hypertextes [1990]. In: C. Pias et al (Hrsg.), *Kursbuch Medienkultur. Die maßgeblichen Theorien von Brecht bis Baudrillard* (S. 525-528). Stuttgart.
MCLUHAN, M. (1995). *The global village: der Weg der Mediengesellschaft ins 21. Jahrhundert* (Übersetzung: C. P. Leonhardt, Einleitung: D. Baake). Paderborn.
MIXNER, M. (1975). Ausbruch aus der Provinz. Zur Entstehung des Grazer "Forum Stadtpark" und der Zeitschrift "manuskripte". In: P. Laemmle & J. Drews (Hrsg.), *Wie die Grazer auszogen, die Literatur zu erobern. Texte, Porträts, Analysen und Dokumente österreichischer Autoren*. München.
MOORE, J. C.; JOHNSON, E. B. & ARNOLD, M. S. C. (1972). Status congruence and equity in restricted communication networks. In: *Sociometry* 519–537.
POPPER, F. (1975). *Art, action and participation*. London: Studio Vista
POPPER K. (1977). Wie ich die Philosophie sehe. In: Coceptus XI, Nr. 28-30, 11-20. (1975). In: Lührs et al (Hrsg.), *Kritischer Rationalismus und Sozialdemokratie*. Berlin / Bonn-Bad Godesberg.
RÖTZER, F. (1991). Mediales und Digitales. Zerstreute Bemerkungen und Hinweise eines irritierten informationsverarbeitenden Systems. In: F. Rötzer (Hrsg.), *Digitaler Schein. Ästhetik der elektronischen Medien* (S. 9-78). Frankfurt am Main: Suhrkamp.
SANDNER, G. (2001). Die Politik des Kulturellen: Cultural Studies in Wien und in Birmingham. In: U. Göttlich, L. Mikos & C. Winter (Hrsg.), *Die Werkzeugkiste der Cultural Studies. Perspektiven, Anschlüsse und Interventionen* (S. 201-222). Bielefeld: Transcript.
SANDNER, G. (2002). From the Cradle to the Grave. Austromarxism and Cultural Studies. In: *Cultural Studies*, 16 (6), 908-918.

SHANNON, D. E. & Weaver, W. (1949). *The mathematical Theory of Communication*. Urbana III: University of Illinois Press.

THALL, N. (1996). *The McLuhan Millennium*. Cambridge, MA.

VIRILIO, P. (1992). *Rasender Stillstand*. München: Hanser.

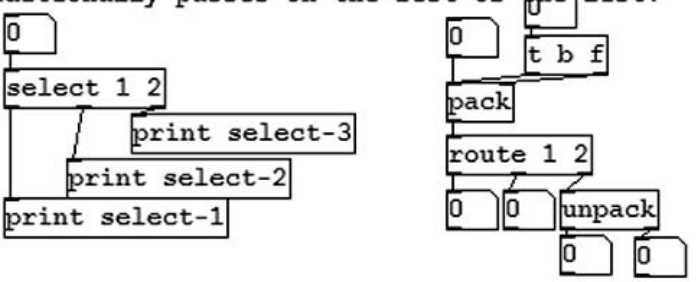
WEINERT, Ansfried B.(1998): Organisationspsychologie: ein Lehrbuch / Ansfried B. Weinert. – 4., vollst. überarb. u. erw. Aufl. Weinheim [u.a.]: Beltz, Psychologie-Verl.-Union.

WICKE, P. (1998). "Move Your Body". Über Sinn, Klang und Körper <http://www2.hu-berlin.de/fpm/texte/wicke6.htm>, *Referat gehalten auf dem Symposium "Sehen und Hören in der Medienwelt" der Gesellschaft für Ästhetik Hannover*, 2.-4.10.1998. [Version vom 12.07.2005].

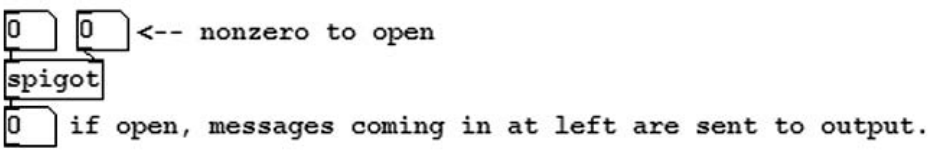
WICKE, P. (2001). Sound-Technologien und Körper-Metamorphosen. Das Populäre in der Musik des 20. Jahrhunderts. In: P. Wicke (Hrsg.), *Handbuch der Musik im 20. Jahrhundert: 8. Rock- und Popmusik* (S. 11-60). Laaber: Laaber.

X, R. A. (1989). Elektronischer Raum. In: *Kunstforum International*, 103, 142-147.

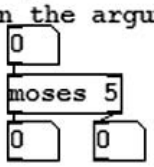
Pd provides at least four objects for doing conditioonal computations. The "select" object tests its input against its argumt(s), and outputs "bang" when they match. The "route" object works similarly but also copies data. In other wors, "route" takes a list, tests its first element, and conditionally passes on the rest of the list.



You also get "spigot" which turns a flow of messages on and off (like the Gate object in Max, but with the inputs reversed):



And finally, "moses" sends numbers to the left if they're less than the argument, right otherwise:



updated for Pd version 0.26

Two Rooms
A Short Conversation with Miller Puckette

Christian Scheib: In a discussion during the first Pd-convention in Graz you described Pd as having a kind of white canvas ideology. What was on the canvas before Pd was invented?

Miller Puckette: Your first question made me think a long while, because the question of what comes before Pd is something like, "everything I knew by the year 1996". And it's hard, even in retrospect, to know what was important and what was not. I think one central thing was a wish that computer music could be made in an even less constraining, and more open-ended, way than was made possible by Max, Pd's predecessor. I avoided re-creating the Max objects at first, and focussed instead on graphing arrays of numbers, which I needed to do anyway to make figures for a paper I was writing at the time. I thought a lot about how to visualize complicated arrangements of data that a composer might use to represent a musical idea. But although I wanted to make it possible to make complicated structures within Pd, I wanted Pd itself to be as simple as possible.

Another aspect of the blank canvas that preceded Pd, was my desire to make a unified way of handling sound recordings ("samples"), images, and control data. I started Max with the IRCAM computer music production scene in mind, but Pd's mental beginnings were more abstract. The unified approach to data storage, not making reference to specific media, was one of my strategies for making Pd as flexible as possible. The one thing I did give Pd ideas about was time and scheduling; and this is perhaps unavoidable, but time passes in the same way (from a physical point of view at least), regardless of what one is doing. But everything else about the structure of a work of art in Pd looks like undifferentiated data.

A third, and more social, aspect was the intellectual property situation. IRCAM made it clear that they didn't want me involved in the further development of Max/FTS (the branch of Max they maintained) – this was one reason I left IRCAM in 1994. There were many changes I wanted to make in Max/FTS, but in the end, I was forced to start from scratch. I made Pd an open source program so that people would want to use it, and sure enough, Pd soon had many more users than Max/FTS. In an odd twist of fate, soon after I started releasing versions of Pd, IRCAM released Max/FTS on the GNU General Public License (much to the happiness of the maintainers). If they had done this before I started Pd, I might never have started it. But by that time (1998, I think) it was too late; Max/FTS was already marginalized and Pd was coming into wide use.

Christian Scheib: If I understand you correctly, something "constraining" must have been on the canvas, before you started whitening it by developing Pd. Constraining in a social way as far as copyright is concerned, constraining in a technical way as far as the difficulty with differ-

ent tools for different media is concerned and constraining in an artistic way as far as pre-produced clichés are concerned. Since there are three reasons, there are three questions.

Manual Digitalisation

Christian Scheib: Observing the outcome of Pd-usage: How has Pd worked artistically so far and is there another development you would still wish for it? Has music become less constraining and more open-ended than before Pd was available?

Miller Puckette: It's hard to describe this, but I keep hoping the computer will be able to function more like a musical instrument (less like a computer) than it has before. The usual mode of doing computer music is still very much like working in a studio (that culture lies at the root of computer music after all). I'd like to see more time-sensitive ways of responding to real-time inputs that would allow human control over the way sounds evolve. But I think this is likely to be a hard research problem.
By and large, Pd is at least as good as any other environment I know of for making computer music, and lots of really excellent work is being done in Pd. I think the limitations that now confront us are more fundamental than just a choice of software (Pd vs. Supercollider, for instance).

Christian Scheib: Hasn't the development of graphic surfaces been the "fall from grace" (in the biblical sense) of computer work anyway? Betraying understanding of what one does by pseudo-analogies that have been invented under the pretext of making everything easier or even more understandable? In other words: hasn't – on a totally different level, but still – a wish for "reacting in a more sensitive way of responding to real-time inputs with more human control" just produced the opposite in the past already? What is the difference in hope and Pd's approach rooted in? (I, too, think it is about more than just a choice of software like Supercollider, yes or no.)

Miller Puckette: I think there are two things going on. The first is that computers are tools for automation. They allow humans to work in patterns, instead of working in details. This makes it very easy to do certain kinds of things (making hundreds of sinusoids, for instance, or drawing fractal trees). But the elementary operation, the putting of a dab of paint on a canvas, for instance, is easier to do by hand than with a computer. So computer art naturally looks different from manual art. Perhaps this is paradoxical, but I hope Pd is less automated, and more "manual", than, say, a purely prescriptive programming language. It's got the right level of automation for making banks of sinusoids without too much trouble, but doesn't encourage making top-down, completely pre-planned compositions, and instead encourages moving forward through experiment.

The second thing is graphical interfaces, and the feeling of intimacy that they give with the computer's workings – a feeling that is, as you suggest, purely illusory. I think the answer is that a graphical interface can be honest or dishonest. An example of a dishonest interface is Microsoft's desktop, on which you don't actually see your files, but only those certain files that happen to be placed so that they're visible there (and aren't hidden). On the other hand, lots of things show up which aren't files at all, and might not even live on your own computer. GUI people call this a "metaphor" and hide behind that word when offering the user things which look alike, but which don't have the same functionality. The result is that you never really know what you are doing.

In contrast, I hope that people who use Pd actually "see" what they are really doing, with nothing hidden and nothing aliased to look like something different. And perhaps that allows a more intimate control of the actual making of computer music or art in Pd than is offered by more metaphor-laden systems.

Transgressing Media

Christian Scheib: To what extent has the unifying approach changed the way people use different media? In which directions are the needs, wishes, developments of the community going in this respect? Does this have aesthetic consequences?

Miller Puckette: I think that in the last couple of decades many artists and composers have become quite fluent at mixing audio and image production and passing controls and information between media. There are some pitfalls (for instance, it's easy to make things that are too predictable or pedantic), but good artists can see and avoid them with experience. I've seen some wonderful work recently, particularly at the Pd convention itself. I think it's really the artistic community which is driving this more than Pd itself, which just happens to be a good way to realize these sorts of things.

Christian Scheib: So the old antagonism between artist and tool/media/instrument is reappearing in some new form? In other words, aesthetically it has been one of the core functions of the range of instruments and/or material to define and provoke what the artist is doing or can be doing or might be trying in order to transgress. What if you were successful in providing the idea of the technologically imagined metaphor of the white canvas? What is left to transgress? Should we just abandon the concept of transgressing? But then look at or listen to the Pd-community's work in Graz: Some kind of transgression has been involved in all the convincing examples. So if the medium inevitably plays such a central role in the production of art, what would you – as an artist and as a programmer – think Pd's role is in this respect?

Miller Puckette: In any form of art-making there has to be tension. However, I would rather see the tension lie in the artistic imperatives than in a contest between the artist and the limita-

tions of his or her tools. This is an aesthetic stance, by the way: plenty of artists actually seek tension in the difficulty of the realization (Iannis Xenakis often did this). But somehow, I think that an artist who really needs difficulty of realization can always find it, and it's not my role to supply difficulties.

A favourite metaphor of mine is to compare two rooms, one tiny one and one large one. If you live in the tiny one, you might want to move to the larger one in order to have fewer boundaries. But of course the larger room simply has a larger surface, and hence more boundaries than the small one had. In the same way, the more transparent, malleable, and powerful the tool is that you use, the more ways you can hit the wall with it. So perhaps, even though Pd tries to give you the most freedom, it ends up giving you the richest possibilities for frustration.

Open Source

Christian Scheib: You mentioned that the reason for the decision for open source was "so that people would want to use it". This is a very practical way of putting something that is also heavily loaded with lots of ideology. Does a more theoretical or ideological thinking have some (hidden or intentional) influence on your work?

Miller Puckette: Well, I hate seeing big corporations rob people, so yes, to that limited extent. I see in particular the increasing use of patents as tools for keeping small players out of the game, and I think we should all resist that. But as for Pd itself, I don't see it as a political act, just as the best way I can see to navigate the situation and get tools in the hands of people who need them.

Christian Scheib: Okay, so let's stay practical instead of ideological here. Getting tools into the hands of as many people as possible may not be, but may well end up soon becoming a marketing concept. Linux for Munich or so. Are you concerned with that? (Sorry, I know this is not practical, but pseudo-ideological.)

Miller Puckette: I'm very excited by the movement toward open source in general, because I see it as breaking the strangle-hold that software corporations have over their users. It's simply unconscionable to prohibit a person from knowing how something he or she owns (a computer) actually works inside. It's also unconscionable for any democratic government to allow itself to be locked into a private vendor in order to carry out essential functions; this endangers the populace needlessly. The situation is different for artists; they aren't in charge of keeping the trains from crashing into each other. So there's nothing immoral about an artist using a proprietary piece of software. However, an artist who thinks carefully about preserving his or her work will naturally prefer the open source solution, because it's much easier to keep running than a proprietary one can ever be.

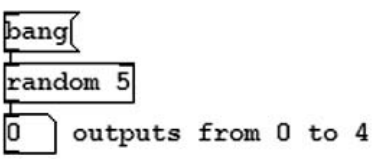
Automated Metaphors (Conclusion)

Christian Scheib: I just love your metaphor about the tiny and the large room with its widening and enlarging of boundaries at the same time. This seems to sum up pretty exactly what the potential of Pd is in the sense of possibilities as well as frustration. This leads to two closely related questions: from your observation and judging from questions and feedback you get, has the community pushed the development of Pd unambiguously into this direction of more and more "manual" openness, or are there also some tendencies to "serialize" or automate? And again from your personal observation, has art/music been developed in recent years that owes its essential quality to these characteristics of the metaphorical large room?

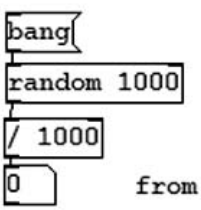
Miller Puckette: Hmmm....
1. I think most people attracted to computers in the first place (including most Pd users) have a tendency to like to automate things. The fact that Pd users are computer artists or musicians in the first place means the population has already been selected for that trait. So I think it natural that one sees more of a tendency to automate Pd usage than to de-automate it. But there are a couple of interesting counter-examples. In particular, the small fringe of people who are actually using the experimental graphical data editing functions of Pd seem to like it for its very explicit, even tedious, detail. So my best answer is, "both".

2. I don't know any examples myself... since it's really the artist, not the viewer, who experiences the possibility/limitation of the software, if he or she wanted to incorporate that, itself, into an artwork, he or she would have to somehow portray or offer an experience of using Pd. Running a camera while the artist works on Pd and showing the video as the artwork, for example. It might be hard to figure out how to make a convincing artwork out of that idea.

Use the "random" object to make pseudo-random integers. To get continuously variable random numbers, make a random number in a large range and divide:

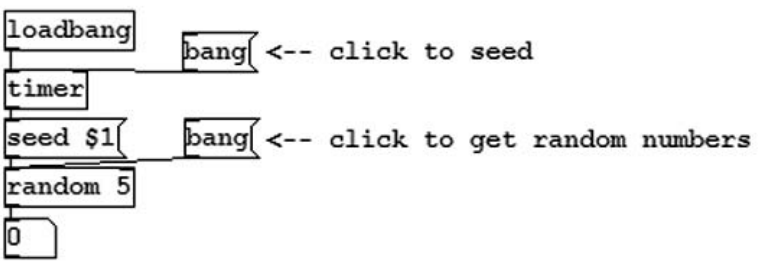


outputs from 0 to 4



from 0 to 0.999

If you don't want the same behavior every time you run the patch, use the time from load to first click as a seed:



If you give two randoms the same seed they give the same sequence. If you never seed them, you'll get different sequences out of each one.

updated for Pd version 0.26

Biographies

Frank Barknecht
Journalist for *Deutschland Radio* and the computer magazine *c't*. Open Source software developer, sound artist and founder of the *RRADical* Pd project.

Reinhard Braun
Articles, lectures, research commissions on photography and media history. Project-oriented cooperation with artists in the field of media/telecommunications. Since fall 2003 curator and editor with *Camera Austria* at the Kunsthaus Graz.

Ramiro Cosentino
Internet and Pd developer. Member of the collectives hackitectura.net, riereta.net, barcelona.indymedia.org, sindominio.net, radio madrid, platoniq.net BCN, straddle3.net BCN. Especially interested in an Open Source platform for global communication. Involved in P.i.D.i.P., a Pd external for manipulating videos.

Günter Geiger
Artist and Pd developer.

Thomas Grill
Composer and sound artist. Open Source software development for Windows, Mac OS, Linux and for Pd, Max/Msp, jMax. Lectures at ELAK (Institute for Composition and Electroacoustic) and at the University for Applied Arts in Vienna. Performances with the *Low Frequency Orchestra*, *the Fruitmarket Gallery*, *Klement/Castello/Grill*, and many others.

Cyrille Henry
Pd developer with an emphasis on interface implementation in Pd patches.

Jürgen Hofbauer
Author, composer and media theorist. Currently lives in Vienna. Co-founder of the Viennese label *Niesom*.

Reni Hofmüller
Reni Hofmller was born in 1966 and lives in Graz as an artist, musician, composer, organizer, curator, activist.
She is co-founder of: ESC im labor, an experimental non-commercial art space, initiated in 1993; Radio Helsinki, a non-commercial local radio station, started in 1995; mur.at: the net art platform mur.at, established in 1998, is a strategic alliance of art initiatives and artists of Graz

and is dedicated to creating an electronic network. Initiator and member of: 42, a female artists group for new media production. Member of: Eva & Co, a feminist artists group, 1990–1992; LTNC – Lady Tiger Night Club, 2002–2005; pd-graz; IMA-Institute for Media Archeology.

Werner Jauk
University assistant professor and lecturer for Systematic Music Theory at the University of Graz with a social / cultural theories focus on music / technology / society perception and (New) Media.

Brian Jurish
Worked as a study assistant at the Academy of Sciences in Berlin-Brandenburg and finished his study of linguistics in 2002 at the University of Potsdam. He wrote the Pd external *Ratts* (Realtime Analogue Text-To-Speech).

Andrea Mayr
Research and development work for *THE THING* in New York. Academic work at the *Zentrum für Soziale Innovation* and the *Internet Center for Education – Vienna*. Co-moderator of *net-time-l*, an international mailing list on net and media culture. Artistic activities: 194.xxx.xxx.xxx -*userunfriendly femalepressure.org* and support for projects like >*idrunners*< and >*toywar*<.

Thomas Musil
Pure Data software developer for Peter Ablinger (D), Bernhard Lang (A), reMI (A), Olga Neuwirth (A). Developer of the IEM libraries, a collection of important extensions (software libraries) for Pure Data.

Michael Pinter
Contemporary artist (painter, composer, author; since 1996 priorities on sound_video_ media_net_computer art mainly with pure data). Various national and international exhibitions and presentations. Supports and organises Open Source projects. Lives and works in Graz_AUT, Zeist_ NL and Berlin_DE. For more info visit http://n21.mur.at.

Miller Puckette
Worked as software developer at MIT and IRCAM. He developed the software Max and wrote the core of the software Pd. He currently works at the University of California, San Diego. Since 1997 he has also been involved in the *Global Visual Music* project.

Marc Ries
Teaching and research work at German and Austrian universities, most recently temporary professorship for Comparative Image Theory at the Friedrich Schiller University Jena. Author for the art magazine *Die Springerin*.

Winfried Ritsch
University Professor at the Institute for Electronic Music and Acoustics at the University of Music and Performing Art Graz. Developed electronic music devices and experiments for interactively generating computer music.

Andrey Savitsky
Designer, photographer and multimedia artist from Minsk.

Christian Scheib
Studied instrumental music and music theory in Vienna and Berlin. Since 1992 journalist for New Music (ORF) and producer of the Radio Österreich 1 radio series *Zeit-Ton*. Since 1995 program director of the festival musikprotokoll im steirischen herbst.

Susanne Schmidt
Political scientist and project manager for Linux based software companies. Freelance author and journalists with articles published in *c't*, *iX*, *Datenschleuder (CCC)*, *Linux Magazin* and *Linux Enterprise*.

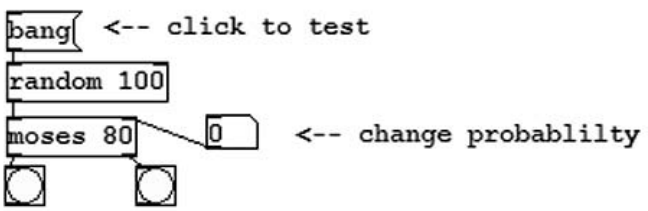
Hans-Christoph Steiner
Interested in robot development, computer music and interactive art projects. Involved in computer music and sculpting in Silicon Valley. Currently lives in Brooklyn working on his masters degree at the *Interactive Telecommunications Program* of the New York University.

James Tittle
James is a Louisville, KY native who graduated from the University of Kentucky with both a BS in Biology and an MS in Neurobiology. He has since squeaked by as an artistic programmer who specializes in interactive video/3d: notable projects include works for the Speed Museum, the Humana Festival of New American Plays, workshops & lectures in Chicago/Bergen/Graz, visuals for Parlour, and numerous gallery shows.

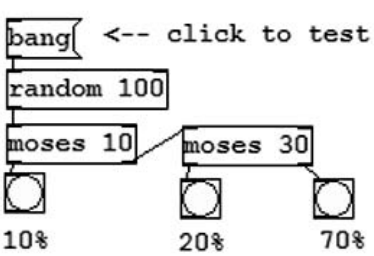
Harald Witsche
Publications on scientific theory, phenomenology and system theory, numerous lectures nationally and internationally. Since 1995 systematic investigation of electronic and electroacoustic music and continental football in theory and practice. Artistic work in the field of electronic and electroacoustic music. Author for *Skug – Journal für Musik*.

IOhannes m zmölnig
Lives and works as a media artist / software developer in Graz. Since 1995 Zmölnig has been involved in electronic music and since 1998 in Free Software. Lecturer at the Institute for Electronic Music and Acoustics in Graz (IEM). Since 2001 maintainer of the Open Source project *Gem*.

You can generate weighted random numbers from uniformly distributed ones. If you just want two possible outcomes with a varying probability for each one, you can do as shown:



This outputs a number at left 80% of the time, otherwise at right, unless you override the "80" using the number box. You may extend this to more than two possible outcomes, for instance like this:



updated for Pd version 0.35

pd~ Release 0.1
Works form the pd~convention 04

Audio + Video DVD

The first release of pd~ is a potpourri of audio, video, audio-visual and documentary works by artists that participated at the 1st international pd~convention in Graz, autumn04.

Tracklist:

1. Convention Trailer (XXkunstkabel)
2. Corrosion (Thomas Grill, Martin Pichlmair)
3. Sinus2b (IOhannes m zmölnig)
4. This Is Pop (Miha Ciglar)
5. Cypod (Beau Casey)
6. Showcase (Frank Barknecht)
7. Att (Georg Holzmann)
8. Moocow Meta Sub Remix (Bryan Jurish)
9. Automata Inak Vre (re_MI, Thomas Musil)
10. Hans Apd (Wolfgang Schwarzenbrunner)
11. Suite For Laptop And A DJ-Setup (Tim Blechmann)
12. r23 (Yves Degoyon)
13. Conica (Tom Shouten)
14. Worm (DJ chew_Z, VJ luX)
15. Pd Patches and Community (Fränk Zimmer)
16. Esoterisches Geplänkel (Bernhard Neugebauer)
17. Improvisation (Miller Puckette)
18. 1002 (Florian Hollerweger)
19. Tragmatics Fragment (Andrey Savitsky)
20. CHDH (Cyrille Henry)

http://pd-graz.mur.at