

Traced back to its pure Unix roots, pluggability is very much a command-line affair maxed out by simple shell scripts and well tuned tools



Destruction in art and computation; day one of degenerative.php previous to character corruption triggered by each page visit

CONNECTIVITY, OPEN FLOW, PLUGGABILITY, MULTIMEDIA PATCHING, CALL IT WHAT YOU WILL, PINS DOWN TO PIPES AND APPLICATIONS, THE UNIX COMMAND-LINE ENVIRONMENT WITH TOOLS SUCH AS CAT, TAC, MKFIFO, AND DD

You've Got Pluggability

In the first of an occasional series examining the artistic uses of free software, Martin Howse specifies a totally pluggable environment for advanced audio and textual experimentation

Free software isn't just about open applications, it's about exposed and extruded code, and a modular, highly constructivist approach to data. We all know, without having to dig too deep into compiler technology, that code is just plain old data. Yet code can equally well bend data, information, and thus meaning, acting as a healthy foundation for contemporary artistic activity. Information theory meets expressionism as we can talk of a signal to noise ratio in relation to artistic intention. Terms and data bend and cross disciplines under strong cultural forces of attraction. And free software artists value openness over functionality on the road to a promiscuous operating system; totally active and open software which we'll map out across further instalments within this series.

Data bending and code promiscuity imply a fine-grained connectivity both between applications and between machines, across operating systems designed only with a functional, secure networked relation in mind. Data bending reads the equivalence of data and code espoused by the Unix way as a sign of promiscuity, of sheer pluggability, reversing the very terms of the originator of Unix philosophy, Ken Thompson, within his key text Reflections on Trusting Trust. Pluggability is both key to the Unix way, and at the same time offers a true road to security hell through necessary implication of the viral as a

direct result of the flattening of data and code; a lack of distinction effected by the compiler which is so central to Thompson's essay. Equally, the unwritten rules which comprise the Unix way, the true path to hacker heaven, stress a modularity, an ease of interface, a transparent simplicity of text streaming and format which implies openness and ease of transport. Self-creating or modifying code is equally praised, if somewhat feared as lazy programming paradigm. Unix sits on a knife edge which is pluggability, an engaging and dangerous realm of artistic exploration.

Connectivity, open flow, pluggability, multimedia patching, call it what you will, pins down to pipes and applications, the Unix command-line environment with tools such as cat, tac, mkfifo, and dd, and, with what can well be considered as further environments interacting with the glue of the shell, the mighty GNU Emacs and both Pure Data (PD) and SuperCollider. A full network can be expressed further with a veritable jungle of infrastructures and fruity hanging apps, such as Open Sound Control (OSC) and associates, and JACK (JACK Audio Connection Kit), with its almost endless list of helpers.

Pluggability under such a network of tools sets out on the very first steps towards the specification of a promiscuous operating system, a totally open, totally active, well connected patchable command-line driven system contrasting heftily with the dull determinism and static ring fencing of GUI containment.

At the same time pluggability means exploring the key artistic notion of OS nature, an active parallel to the aged landscape painting of visual art. Artistic endeavour is all about taking nothing for granted. The free and totally open field of coding in theory presents endless possibilities, yet a solid awareness of what has gone before and exactly what is implicated within the tower of abstractions ascending from bare hardware is essential. Speaking the language of the processor shifts our meaning irrevocably, as intention becomes seriously clouded. Yet when it

comes down to transparency, to an exposure of the givens of a contemporary operating system, the role of free software is clear. Meaning and intention are arrayed across megabytes of source code within an active community of coders. The idioms and idiosyncrasies of so-called high level languages can be unravelled. And, in addition to a vast pluggable tool-set we can trace and monitor active process with applications such as strace and GDB. OS nature operates as a shorthand for this artistic modus operandi expressed in the works of the author, Martin Howse, and other free software inspired explorers such as Erich Berger and Aymeric Mansoux of goto10.org fame.

PLUG AND PLAY

It may well chime as an ill-used phrase from the land of the proprietary, implying a tedious click-through dance with drivers, yet plug and play is perhaps better suited as a joyful motto for pluggability, again borrowing much from the Unix way. As we've seen, and under common everyday piping examples at the command-line such as grepping through mail or redirecting the output of find to a logfile, Unix is all about plugging together small, highly specialised tools and filters. It's a creative act which contrasts well with a GUI designed in Pavlovian labs for conditioned input. There's no question in that instance of thinking outside the widget-driven window. Piping sits on a par with programming; after all, using brackets, redirection operators such as > and >>, and with the | pipe, tee and named pipes or FIFO (First In First Out) buffers, together of course with a well honed array of Unix tools, complex applications can readily be constructed. Pluggability is very much about this environment-driven and highly creative process. Pluggability implies familiar programming terms such as glue code and modularity and stresses connectivity. Redirection is the first step on the road to pure patchability; assemblages of tiny well specified commands enlivened by the act of

data piping. It's easy to see how such a volatile process lends itself to live performance.

The usual state of affairs is to input our commands to the shell, the command-line, by way of keyboard, with such entry referred to as standard in or STDIN. It's no surprise that output to the terminal is to STDOUT. Yet we can choose to redirect to files and named pipes either at the input or output stages using the >, >> or < operator.

By way of noisy example cat /usr/src/linux/* > /dev/dsp throws the Linux kernel source code as raw 8 bit data straight at the soundcard. Use of the >> operator appends a stream to existing data. Yet, what's really revolutionary is the pipeline signalled by the | character; we can quite simply pipe data from the output of one tool or filter to the input of another. A classic example, provided by Tom Truscott, inventor of Usenet, prints out the frequencies of each word in a file:

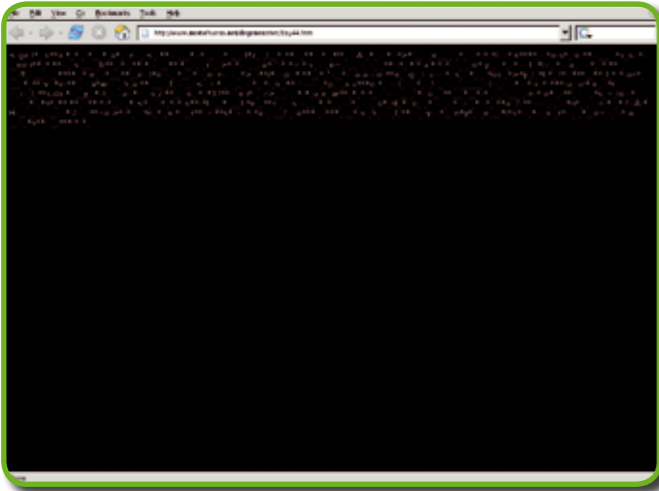
```
cat file | tr -cs '[A-Z][a-z]' '\012' | sort | uniq -c | sort -nr | more
```

It's an example which could well stretch the imagination of those working with text; digital poets such as Alan Sontheim, Ted Warnell and Mez Breeze swamping Net art mailing lists with a modern, fanciful and highly generative vocabulary. Generic piping tools such as cat, in concert with old friends like grep can well be used to further textual experiment which, in the manner of

concrete poetry, can well expand into the visual. Further tools of interest here would include sed, awk, sort, diff, and csplit. Indeed sed and awk feature prominently in a key text, UNIX for Poets, from Kenneth Ward Church of AT&T Bell Labs, birthplace of Unix.

Yet, it's no great surprise that sheer pluggability blossoms most fruitfully within the realm of advanced audio experimentation. Despite extensions such as GEM and PDP which pull PD kicking and screaming into the realm of the visual, it's worth remembering that PD was originally solely concerned with audio. And though PD code daughter Packet Forth (PF) is concerned with data packets which well approximate to a visual frame, when viewed as patching, pluggability is a more appropriate concept within the world of audio. Indeed, audio patchability, corralled within the

Archived online in all its glory, the 44th day of an intriguing degenerative net art project



computer, borrows very much in terminology, interface and workflow from a studio world of patch bays and corded analogue synths. There's little imagination at work in the titling of core application JACK. Untied from frame rates and fixed packet sizes, and also the sheer memory and processing demands involved in shifting and processing high quality visual material, audio presents a more fluid subject. And perhaps the pluggable model is simply less appropriate for the image. It's a complex argument which has much to do with meaning and the pure physicality of well amplified sound waves. Both also stand in a differing relation to time and meaning. Audio can reveal complex systems in a physical manner, systematics which well suit the paradigm of programming. In contrast, video in this context is all too ready to lean on the crutch of meaning, of narrative. The modern VJ is very much in need of a fresh, new model for visual construction.

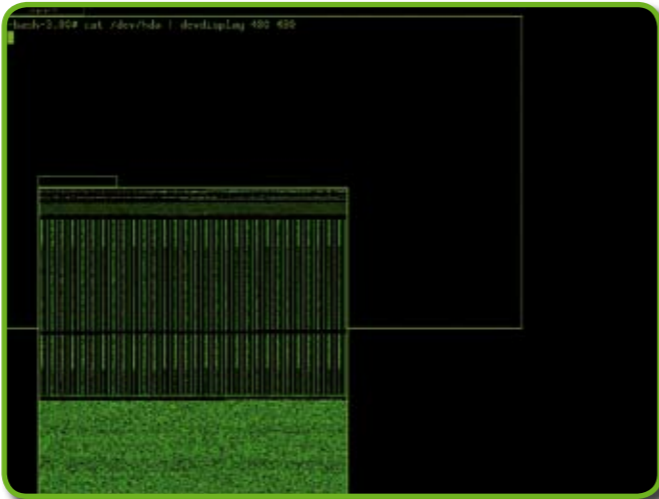
CAT POWER

Pluggability is all about assembling a towering, tottering environment of modular apps for purely playful pluggability. We've briefly encountered staple cat, which can readily be untied from a singular process with the use of FIFOs or named pipes. FIFOs are useful if we need to pipe to and

from an application which expects to deal with a file rather than STDIN or STDOUT. A regular FIFO is constructed using a mkfifo MYFIFO incarnation, and forthwith we can pipe to and from MYFIFO with wild abandon. However, it's worth remembering that only one copy of the input message cats its way through the FIFO. Forget using the FIFO as a multiplexer furnishing multiple copies of a single stream. The rather deprecated /dev/fanout tool is more appropriate here, yet cut and splice functionalities in the latest kernel series are more appropriate for the hardcore hacker artist's attention.

Cat can also be well considered in concert with it's looking glass playmate tac which does exactly the same thing in reverse. And tee is also a decent addition to the party, working to multiplex standard input to multiple files. Erich Berger is probably one of the most famous artists to explore cat, under an audiovisual performance of the same name premiered at world renowned art hacker festival Pixel in Norway last October. His cat nominates the GNU/Linux operating system as key instrument, a prime example of OS nature activity, and makes use of basic cat operations to generate audio and video, controlling feedback of these dual streams through mixer settings. Both data types are flattened to the physical, as audio is in some sense catted to the video projector through plain cable.

Yet, it's well worth bearing in mind that this cat still has claws and teeth, particularly when running rock-and-roll-style (live fast, die young) as root where much of the more exciting cat action is to be had. Always check where you're catting to, particularly in the case of devices. Never cat to /dev/hda, /dev/sda or any of their partitioned offspring unless



The author's own devdisplay script provides ready visualisation of OS nature, in this instance a walk through raw hard drive data

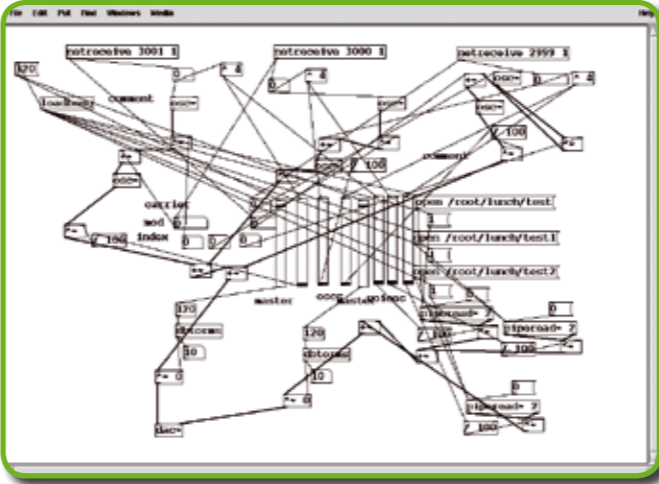
attempting an OS hari-kiri, or bluntly translating a Pete Townshend guitar and amplifier collision to the more staid world of live laptop performance (see Destruction in Code).

Cat can be used to implement simple audio feedback loops and playful filtering scripts can easily be constructed in BASH, Python, or, for concise effect, in CHICKEN Scheme. For recent work as part of the PLENUM project in collaboration with mavericks KOP (Kingdom of Piracy), Martin Howse briefly coded a few command-line apps in the latter language to ease piping operations. These include interfaces to OSC (Open Sound Control), another very useful connectivity framework which can be used to glue together all possible pluggables, a wrapper around a basic neural network implementation, and jekyll, an eccentric sample slicing app. Jekyll cuts piped data according to step size and sample size. A command such as:

```
cat /dev/hda1 | jekyll 20 10 > /dev/dsp
```

chops a raw stream of partition bytes into ten sample long chunks at intervals of twenty samples before passing these on to the soundcard.

Further code includes devdisplay, which opens a user-specified window to display raw data piped to the command. And, more recently, the code base has expanded to allow for piping data, borrowing code from ancient cassette tape interfaces for home computers, across the airwaves.



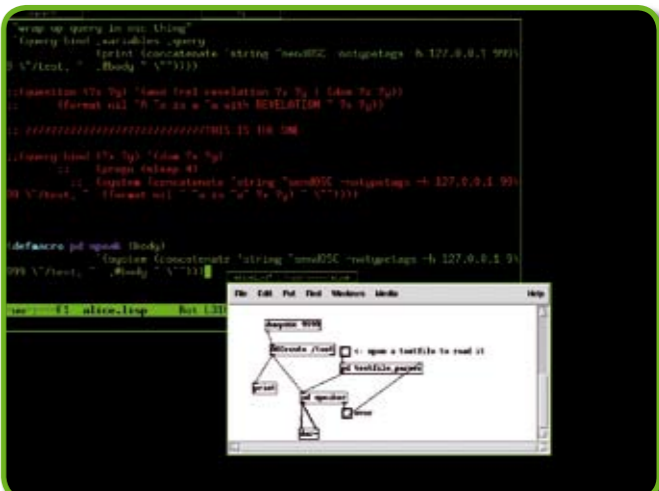
PURE PIPES

Of course the king of all pluggable apps, indeed one which, alongside the Unix model, has spawned so many extensions and extension coding paradigms that it could be dubbed the father of modern creative computing, is Pure Data. Yet though PD presents the ultimate pluggable environment, as can readily be garnered from screen shots exhibiting networked tangles of connected code objects and abstractions, PD chooses rather selfishly to play less well with others, and perhaps could be accused of enforcing a less than code-centric approach to pluggability. Nevertheless, users always have a well designed extension API to work with and can equally well batter down the gates of PD with the blunt tools of piping, or more refined OSC and JACK.

Aside from obvious internal piping of signal and control data, by way of the visible interconnects or patch cords and simple send and receive objects which assist in clearer patching, PD supports access to named pipes only by way of the piperead~ and pipewrite~ extensions bundled as part of the ext13 library. After downloading and installing from the central PD Sourceforge repository such extensions must be specified within the user's .pdrc or at the command-line with a -lib path/to/library switch. Usage is well documented, yet, as with all FIFOs, care must be taken in opening and closing named pipes to avoid breaking or blocking the pipe. PDP, a packet oriented, largely graphical extension for PD authored by Tom Schouten, supports a piped interface under the pdp_rawin object.

THE USUAL STATE OF AFFAIRS IS TO INPUT OUR COMMANDS TO THE SHELL, WITH SUCH ENTRY REFERRED TO AS STANDARD IN OR STDIN. IT'S NO SURPRISE THAT OUTPUT TO THE TERMINAL IS TO STDOUT. YET WE CAN CHOOSE TO REDIRECT TO FILES AND NAMED PIPES

A workaday Pure Data (PD) patch, composed for a wireless performance project, leverages basic netreceive connectivity and raw piping



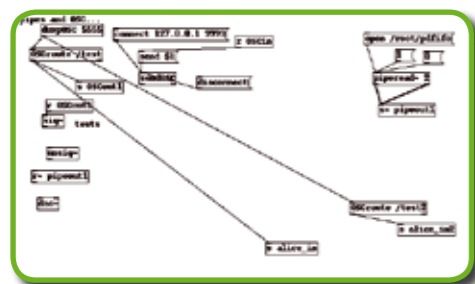
Pluggability is very much dependent on interface. In this instance OSC takes care of Common Lisp to PD communications, with macros wrapping up otherwise complex operations

PDP, or Pure Data Packets, furnished a starting point for one of the most lively experiments in code pluggability, Packet Forth (PF), also from Tom Schouten, which can be described in short as the ultimate multimedia glue language, based on Forth and inspired heavily by PD's patching metaphor. PF interfaces well with the Guile embedded Scheme implementation and connects to virtually every framework out there. PF code objects and interpreter are readily accessible from within PD as prefixed by the initials pf followed by a space and then the name of the scripted PF object. It's a simple enough approach to extension, but there's much more to it, with PF as standalone interpreter or GNU Emacs interfaced live coding environment. Under a read-raw-packet operator, PF can well deal with named pipes which are used within an mencoder-driven PF example to enable AVI

reading. Goto10.org, home of the pure:dyne distro, provides repositories and helpful HOWTOs for PF and associated code.

The netsend and netreceive objects offer primitive UDP or TCP enabled pluggability for network connected PD players, and the FUDI protocol used for communication is simple enough here to be exploited in self-coded applications. The two bundled command-line apps, pdsend and pdreceive, can equally well be used or source code examined as a decent guide. Yet OSC offers a far more versatile and readily supported protocol for inter-pluggable networked and local communications. OSC is supported by all our pluggables, including PD, PF and SuperCollider, as well as receiving excellent library and implementation attention across a range of common languages, and environments such as Csound. It's an easy protocol to get to grips with and PD extension OSCx, available again from the Sourceforge repository, is a good starting point, furnishing excellent example patches. Command-line apps dumpOSC and sendOSC, wrapped by the author in both Common Lisp and as more pipe friendly tools, are bundled with the extension for experimentation.

For pure audio-centric pluggability the JACK



Pipes and OSC plug the whole world into PD thanks to a few handy externals together with abbreviated send and receive objects for greater patch abstraction

daemon and its troupe of companion apps such as the Qjackctl GUI controller are hard to beat, particularly when using sound cards with multiple I/O possibilities. JACK allows for extensive audio patching between often abstracted hardware and software, and does function as an infrastructure which PD can easily make use of, yet the patch bay and connections presented by Qjackctl are simply too rooted in traditional studio methodologies to allow for advanced experimentation. Rather it's more enticing to project the artistic misuse of technology in taking software engineering metaphor and running with it out of bounds. A JACK leak app could easily be coded along a train of thought from piping to plumbing and thence to unfortunate leaks and overflows which rightly mirror memory leaks, buffer overflows allowing entry into the viral. Such an architecture of leaks, drips, echoes and accompanying collectors and containers could be implemented thanks to infrastructures such as PD with extensions such as k_jack~, which abuses PD's audio abstraction layer to allow for objects to map directly to JACK inlets and outlets.

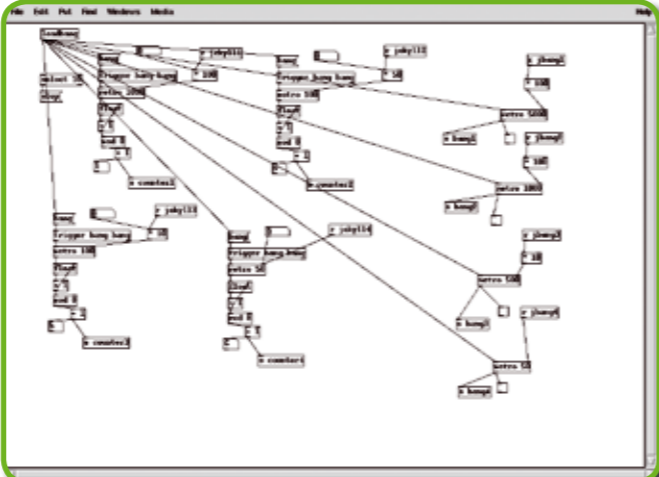
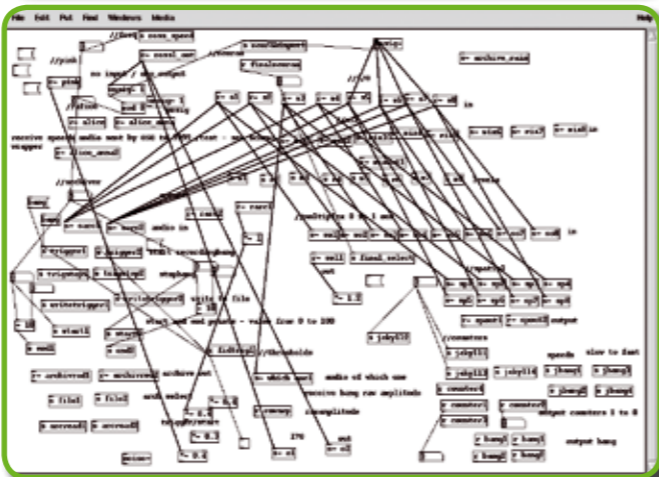
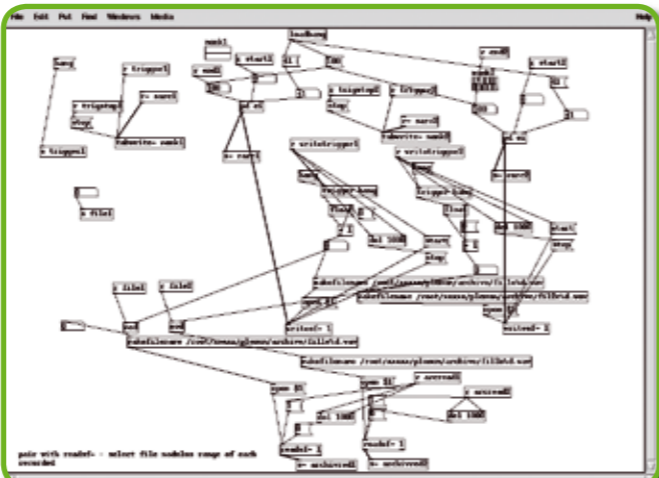
PUNISHING THE ATOMS

Yet there's still something missing from our web of well specified pluggability apps, interfaces and infrastructures; a heavyweight code-centric spider which under live coding scenarios can plug and play with all these toys. GNU Emacs, headily extensible in working use by way of Emacs Lisp code, and mapping through Inferior Lisp processes, SuperCollider interface and the like to almost all other environments, deserves a central, sprawling position here. Indeed, GNU Emacs itself presents a vast edifice of what could be termed pluggable code. Under the steady gaze of the living interpreter we can come to view pluggability in a totally different light. Pluggable, dynamic code is the very dream of the multimedia artist, pushing beyond the coarse grained Unix methodology of chunky connected tools.

Yet to some extent, and particularly under ill-titled Pure Data, the absolute fine-grained pluggability of data as code and vice versa is impossible to achieve across the less than dynamic wall of the PD code object, the closed black box presenting clean interface. We can bridge the gap between the code world of GNU Emacs and PD somewhat with OSC, PF and further extensions such as k_guile, which allows for PD objects to be constructed in Guile, but there is still a distinction

between interpreter and object. K_guile, from the creator, Kjetil S. Matheussen, of a range of k_ prefixed externals which stress connectivity to SuperCollider and other apps, presents a unique yet to some extent pinned down code-based window into PD, with the added functionality of an EVAL hinting towards live coding work.

Indeed, the venerable, primarily text-based SuperCollider (SC) multimedia coding environment presents a clean code-interface to GNU Emacs which certainly raises the pluggability stakes and makes of SC an attractive model, particularly in concert with the largely undocumented ssc, or S-Expression SuperCollider which implements a Scheme-style read syntax for the SuperCollider language. Integration with GNU Emacs is total, yet installation and start-up routines are both brittle and painful. Short documentation can be found online at the author's 1010 site.



A series of performance patches captured after the event, a collaboration with KOP, exhibit the total pluggability of the Pure Data environment

Destruction in Code

Like some Hiroshimic Easter egg, a hidden design, destruction is embedded within both the history and contemporary architecture of computation, with a nod in the direction of John von Neumann and his ubiquitous CPU design model. Von Neumann's dual role within the history of computation and the Manhattan project, charged with developing and implementing the first atomic weapons, is clear. The subsequent history of the paralleled application of simulations and game theory to the political manipulations of the Cold War is less well documented, but does make for intriguing reading. The field of auto-destructive art, as pioneered by artist Gustav Metzger, stands in direct relation to these atomic issues, acting in some way as mirror or cathartic demonstration. Metzger was both actively involved in anti-nuclear campaigns throughout the 60s and 70s, as well as

being one of the first artists to interrogate computation as a highly flexible, expressive medium. He participated in the seminal computer art show, Cybernetic Serendipity, at the ICA, London in 1968, authored pioneering articles on early computer graphics, and sketched detailed plans for auto-destructive works reliant on, and necessarily embedding, new technology. One such design proposed a wall-like sculpture comprised of 10,000 uniform elements. Over the course of ten years, each element would be ejected under the control of a computer programmed by the artist himself. Such a work was never completed, and several designs exist for similar pieces placed within public space. More recently, and on a smaller scale, Tsunami.Net, in collaboration with Team Fragnetics, presented the live self-destruction of an online Web server under a five tonne industrial crusher.

Auto-destructive art in this context places an emphasis on execution in both the common sense, and of course in the light of the executable program itself. Code impacting on the physical is made totally explicit in destruction. The code side of the equation is clarified by performance works such as those of activist, artist and core PD coder, Yves Degoyon whose rm -rf /* work is definitely a piece unintended for home recitation. Patching PD frenetically for advanced audio, Yves issues the deadly command mid-performance, allowing the audience to hear and witness, by way of projected desktop, the return of the machine to a less than usable state. Similar, though less dramatic pieces, include the degenerative. php Web work by Eugenio Tisselli, a work which corrupts a single character on the page each time the site is visited. The page is now totally black.

AUDIO CAN REVEAL COMPLEX SYSTEMS IN A PHYSICAL MANNER, SYSTEMATICS WHICH WELL SUIT THE PARADIGM OF PROGRAMMING. IN CONTRAST, VIDEO IN THIS CONTEXT IS ALL TOO READY TO LEAN ON THE CRUTCH OF MEANING, OF NARRATIVE

With OSC and JACK, thanks again to k_jack~, as middlemen, PD and SC can plug and play reasonably well together, bridging a divide which stems from PD's stricter patch model against SC's more compositional approach. It's the core of pluggability, and with such infrastructure in place it's less than relevant what favoured apps are thrown into the mix, with Common Music, Nyquist and Csound as ready contenders worthy of attention. Yet it's still very tempting to imagine a PD shorn of old-fashioned patchwork and mouse monkeying. It's a lively proposition which has proved equally attractive to the creator of Packet Forth, Thomas Schouten, who envisages coding a PD model in the artistic Common Lisp language, and is thus slowly mapping his PF to Scheme. It's worth checking out code, for both libpf and BROOD, his most recent Scheme-based project, together with embedded intelligent code musings, on a daily basis from his darcs repositories. These are heady days indeed for the modern, highly promiscuous plug and play free software artist.

SYNTH

- Reflections on Trusting Trust
<www.acm.org/classics/sep95>
- Pure Data
<pure-data.sourceforge.net>
- SuperCollider
<supercollider.sourceforge.net>
<www.cimat.berkeley.edu/OpenSoundControl>
- JACK
<jackit.sourceforge.net>
- Erich Berger
<randomseed.org>
- Aymeric Mansoux
<goto10.org/~/->
- Unix for Poets
<www.stanford.edu/class/cs224n/handouts/kwc-unix-for-poets.pdf>
- GEM
<gem.iem.at>
- PDP
<zwizwa.fartit.com/pd/pdp>
<dev/fanout
<www.linuxtoys.org/fanout/fanout.html>
- Martin Howse
<1010.co.uk>
- PF
<zwizwa.goto10.org/darcs/libpf>
- SSC
<www.slavepianos.org/rd/sw/sw-40>
- Gustav Metzger
<www.luftgangster.de/gmetzger.html>
- Yves Degoyon
<ydegoyon.free.fr>
- Degenerative
<www.motorhueso.net/degenerative/degenerative.php>