Playing with Pd **By Jim Aikin**

Apr 1, 2004 12:00 PM

It wasn't too many years ago that musicians with more passion than money thought nothing of building their own music hardware. Electronic Musician founding editor Craig Anderton, who is handier than most with a soldering iron, once built an entire mixer for his home studio.

Now that the world is inundated with massively engineered gadgets such as cell phones and shirt-pocket computers, it's getting tough to build your own hardware. But creating your own music software is easier than you might think, and more affordable, too. In fact, you can do it for free.

Music software for DIY aficionados comes in several flavors. You can write your own sequencer or synth in a conventional programming language, but there are easier options. This month I'll take a close look at one of the more powerful and (relatively) user-friendly programming environments for music: Miller Puckette's Pd. You don't have to know anything about computer programming to use Pd, but some patience and the ability to think logically wouldn't hurt.

Given that so many wonderful music software tools are available with the wave of a credit card, why would you want to roll your own? For one thing, you may want to do esoteric things that no commercial software will do. With the aid of a program like Pd, you can generate and process complex streams of MIDI data for a live performance or build your own software-based synthesizer and effects.

Beyond that, solving technical problems in software design can be a fascinating challenge. You get a sense of accomplishment from creating something that works exactly the way you envisioned. Then there's the money issue. Your personal Pd software synth may not sound quite as massive as Native Instruments Absynth or VirSyn TERA, but if you've spent your last dime on a fast computer and a decent sound card, tools such as Pd and Csound (see “Csound Comes of Age” in the July 2002 EM) are an unbelievable deal.

In this column, I'll tell you how to get Pd, and I'll provide a few insights into what it's like to work with it. (A full Pd tutorial is beyond the scope of this column.) To give you some ideas about what you might want to do with Pd, I've created two MIDI processing utilities and an unusual step sequencer. You can download them from the EM Web site. Once you've installed Pd, you'll be able to use them and customize them to meet your needs.

WHAT IS PD?

Pd is a freeware graphical programming environment for real-time MIDI and audio manipulation. (Extensions are also available for video processing.) To create a program in Pd, you connect little boxes to one another with the mouse using graphic “patch cords.” Once you've created or loaded the patch, you can interact with it by sending MIDI messages to your computer and by using onscreen sliders and buttons.

If you've ever seen or used Cycling '74's Max, Pd will look eerily familia because to a considerable extent, Pd is Max. Puckette created Max at IRCAM in the mid-1980s. At present, Max is a commercial application (for Mac and Windows), and Puckette has no direct involvement with it. However, he's actively developing Pd, which, in addition to Mac and Windows, also supports Linux and Irix operating systems.

One of the main differences between Pd and Max is that Pd's functionality is simpler than Max's, and its simple user interface, although adequate, is quite bare-bones. A third variety of the same software, jMax, is also available as a free download from IRCAM (www.ircam.fr/equipes/temps-reel/jmax/en/index.php3).

The fact that Pd is free has a number of implications. If you're a C programmer you can customize the source code from the ground up or write new Pd objects that will integrate with the programming environment. Also, there's no tech support. There is, however, an active mailing list (http://iem.at/mailinglists/pd-list) to which you can subscribe.

Technical questions that are posted to the list are usually answered within a day or two by more experienced Pd users. For lots of useful miscellany on Pd (including downloadable goodies designed by various people), the place to go is www.pure-data.org. Puckette, who now teaches at the University of California at San Diego, is not affiliated with pure-data.org, but he does monitor the mailing list.

GETTING AND RUNNING PD

All of the versions of Pd can be downloaded from http://crca.ucsd.edu/~msp/software.html. No special installation procedures are needed. The Windows version, which is what I use, arrives as a Zip file. After it's unzipped, you should place the folder named pd in the root directory of your C drive. (Mac users can get an OS X — style “package” created by Adam Lindsay at http://homepage.mac.com/atl/sw. It simplifies the installation process.)

In Windows, Pd runs from a command-line interface. You can't run it by double-clicking on the program icon. Instead, you open an MS-DOS prompt window and navigate to the directory containing the program by typing this:

```
cd c:\pd\bin
```

The prompt will then say: c:\pd\bin>. That shows you're in the pd\bin directory, and depending on what MIDI and audio devices are available in your system, you then start the program with the needed command-line arguments. I have an ASIO sound card, so I start the program like this:

```
c:\pd\bin>pd -asio -midiindev 1
```

This tells Pd to use the ASIO driver and the first MIDI device listed in the Windows control panel.

Keeping the MS-DOS window open while Pd is running is both necessary and useful. For diagnostic purposes, Pd includes a handy "print" object whose output appears as text in the MS-DOS window.

Pd's main window hardly qualifies as a window at all, though it does have clipping indicators and a couple of checkboxes (see Fig. 1). When you open an existing file or load a new one, you'll be working in a separate window that looks more like Fig. 2 or Fig. 3. New objects can be added to this window from a menu, or you can type Control-key commands, which is faster.

Pd comes with a set of tutorial files to get you started. If you're new to the whole idea of graphic programming, you should definitely spend a few days working your way through the tutorials. Pd also has interactive Help files for most of its objects. Right-clicking on an object brings up a context menu from which you can select Help. That opens a new Pd window with information and an actual functioning Pd patch that demonstrates how the object works. Some of the discussions are a bit sketchy, but even so, the Help files are a great resource. You can also read an HTML manual in your Web browser.

PROGRAMMING IN PD

There are a few tricks to handling audio in Pd, so it's best to start with MIDI. Pd provides a number of objects for handling MIDI, the two most basic being notein and noteout. To use them, insert a new object box from the Put menu, and type "notein" (without the quotation marks) in the box. Follow the same procedure to make a noteout, and connect them as shown on the left side of Fig. 2.

The little black indicators at the top and bottom of each box are inlets and outlets. All objects in Pd receive signals at inlets (on the top of the box) and transmit signals from their outlets (on the bottom). A connection is made by clicking on an outlet and dragging with the mouse to the inlet of

another box. Each type of object does something different, so each has a different combination of inlets and outlets (see Fig. 4). The inlets expect to receive certain types of signals, and if an inlet receives a signal that it can't handle, you'll get an error message.

The connection between notein and noteout in Fig. 2 does nothing except pass on to the output all of the MIDI Note On messages received at the input. It's a MIDI Thru connector for notes. The connection between bendin and bendout is a little more interesting. This patch subtracts 8,191 from the data coming out of the left outlet of bendin, and then passes on the result to the corresponding inlet of bendout.

Why do that? Because this simple patch fixes a bug in Pd. The bendin object is sending the MIDI channel of the incoming Pitch Bend message from its right outlet, and the value of the Pitch Bend message from its left outlet. MIDI Pitch Bend is a 14-bit message, which means the data can have any of 16,384 possible values. But as it happens, bendin transmits this data in the range between 0 and 16,383, while bendout expects to receive it in the range between -8,192 and +8,191. By subtracting 8,191, this patch puts the Pitch Bend data in the range that bendout needs. (Note: like most computer programs, Pd expects large numbers to be entered with no commas.)

Just about everything you do in programming Pd will be more or less like that: figuring out what range of values the receiving object needs to see, and then shaping the data stream so that it looks right when it reaches its destination. In addition to simple arithmetic operations, you'll use objects like clip (which limits data to the range you specify), spigot (which opens to allow data through and shuts to cut data off), moses (which "parts a data stream," sending values above a threshold to one outlet and values below the threshold to another outlet), and value (which stores a numeric value for later use).

Web Clips 1, 2, and 3, which you can download from the EM Web site, will give you a taste of what you can do. I don't have space here to go through each patch in detail, but I've included comments in the files themselves.

The xposer (transposer) patch (see Fig. 3) can turn each incoming MIDI note into a four-note chord. For each note in the chord, you can choose a MIDI channel and scale the Velocity up or down. Each note can also be delayed, turning the chord into an arpeggio. The delay time is set in 16th notes at the current tempo, and some simple chord presets have been provided. The ctrlprocess patch is for scaling, offsetting, limiting, and inverting MIDI Control Change messages. The stepseq patch can generate sequences up to 64 16th notes long, and you can interact with the sequence in numerous ways while it plays.

CLEAN CUPS, MOVE DOWN!

The key to Pd programming is understanding the order in which the messages are transmitted. A personal computer is extremely fast, but it does only one thing at a time. A Pd algorithm can work in an unexpected way, or fail to work at all, if you make a bad assumption about the order in which events take place.

Objects in Pd send "simultaneous" messages from their outlets in a seemingly counterintuitive way, servicing the outlets in right-to-left order. But what happens when several patch cords are attached to a single outlet? In Max, the objects attached to a single outlet receive their messages in right-to-left order based on the screen position of the receiving object. That has a good side and a bad side. The good side is that you can always see the order in which messages are passed by looking at a patch on the screen. The bad side is that you can "break" a patch if you tidy up the screen and in the process change the right-to-left orientation of two objects that are attached to the same outlet.

In Pd, this situation is turned on its head. When several patch cords are attached to a single outlet, the Pd object sends its messages in the order in which the patch cords were created. As a result, you can safely shuffle objects around on the screen (you'll be doing that a lot) without breaking the patch. But if a patch is malfunctioning, you can't necessarily tell what's wrong by looking at it, because there's no visual indication of the order in which the messages are being transmitted. Deleting the patch cords and reattaching them may be the only way to be sure of what's going on. In fact, I've found that when subpatches are copied and pasted in Pd, the order of the connections can change. After copying, I sometimes have to delete patch cords and reattach them.

AUDIO IN PD

When it comes to synthesis and signal processing, Pd is very good at some things and not quite so good at others. It will do sample playback and FM synthesis, and the waveshaping tools let you easily produce a variety of tone colors. Pd's filters, however, don't compare with what you can get in commercial software. A basic ADSR envelope generator is part of the software library that comes with Pd, but creating a multisegment envelope with Velocity modulation inputs is a project that can easily take an evening if you're new to Pd programming.

Audio objects in Pd (more than two dozen are provided) have tildes (the ~ character) at the end of their names. The osc~ object, for example, is an oscillator. The mtof~ object translates MIDI note numbers into frequency values suitable for playing equal-tempered scales, the readsf~ object plays a sound file from the computer's hard drive, and the dac~ object sends

audio to the sound card.

Audio data normally has a range from -1 to +1. If you accidentally multiply an audio data stream by a large value, Pd will not protect your speakers or your ears from the results. Nor does it have a DSP usage meter; in my experiments, however, Pd seems fairly efficient (though the screen redraws are slow).

PURE DIY

Currently at version 0.37, Pd is a work in progress, not a finished program. But while it may not do everything you might envision (at least not easily), it has enough tools to keep a do-it-yourselfer busy for a long, long time. Best of all, you don't have to worry about burning yourself with a soldering iron!

Jim Aikin writes about music technology for a variety of publications and Web sites. His most recent book is Software Synthesizers (Backbeat Books, 2003). You can visit him on the Web at www.musicwords.net.