PeRColate

A collection of synthesis, signal processing, and video objects
(with source-code toolkit)
for Max/MSP/Nato

v. 1.0b3

Dan Trueman R. Luke DuBois
dan@music.columbia.edu luke@music.columbia.edu
Computer Music Center
Columbia University
September, 2001

Introduction

PeRColate is an open-source distribution of a variety of synthesis and signal processing algorithms for Max, MSP and Nato. It began around a (partial) port of the Synthesis Toolkit (STK) by Perry Cook (Princeton) and Gary Scavone (Stanford CCRMA). Like the STK, it provides a fairly easy to use library of synthesis and signal processing functions (in C) that can be wired together to create conventional and unusual instruments. Also like the STK, it includes a variety of precompiled synthesis objects, including physical modeling, modal, and PhISM class instruments; the code for these instruments can serve as foundations for creating new instruments (one example, the *blotar~*, is included) and can be used to teach elementary and advanced synthesis techniques. Given its STK heritage and educational function, PeRColate is largely un-optimized. Since its first release, PeRColate has come to include many more objects not from the STK; some are from RTcmix and others are our own evil creations, designed to crash your computer, but only after making some kind of interesting sound. In addition, a library of PeRColate Nato video processing objects has been created, to munge up your video along with your audio.

Distribution and Support

PeRColate is freely distributed and largely unsupported (please read the software license in the README that comes with the distribution for specifics). We welcome bug reports, but make no promises to fix them. Many, many bugs from the original version have been fixed. Phew. Source-code includes some Codewarrior 6.0 projects, though it is it likely that some of the access paths will have to be updated to match your directory structure. Many of the synthesis algorithms are protected by various patents, mostly by Yamaha

and Stanford. For those who are interested, there is an STK mailing list for discussion of various aspects of the STK:

http://www-ccrma.stanford.edu/CCRMA/Software/STK/maillist.html

PeRColate and STK are quite similar, but there are many differences in implementation, so don't expect much help from the list for specific implementation issues (or anything else, for that matter).

The current version of the PeRColate distribution, as well as revision notes (when they happen) and this document, are available at:

http://music.columbia.edu/PeRColate

Installation

PeRColate should unstuff into a folder with colored subfolders. The two folders labeled 'Hot' (which is normally a shade of red) are the only two you need to keep if you don't care about the source code that comes with the distribution. The folder labeled 'Cool' (light blue) is the master source code folder. The CodeWarrior projects we've included in the source code folder should compile provided you reset the search paths to include the relevant files from the Max, MSP, and/or Nato development kits.

The two 'Hot' folders, *PeRColate_objects*, and *PeRColate_help*, should be installed in your externals folder and your max-help folder, respectively.

Inside the distribution folder is a Max patch called *PeRColate overview*, which is a master index of sorts for the objects included in the current collection. Opening it up is a great place to start. If you're running Max version 4, you can place it in the :patches:extras folder of your Max distribution for easy access.

MSP users who don't own a Nato license (and Nato uses who don't own an MSP license) will not be able to use those objects which are written for systems they don't have installed. Attempting to use those objects (or open their help files) will result in harmless errors being printed in the Max window.

Object Classes

PeRColate is a collection of 63 objects, and the number seems to be getting larger by the day. To make your (and our) lives easier, we've divided the objects into eight categories, defined below.

Physical Models
Physical modeling is a class of sound synthesis based on the actual physics of musical instruments. All of the models here are waveguide models, which use delay lines and reflection coefficients to simulate the waveguide properties of flute bores and strings. Included in this set is a new instrument, the *blotar~*, which is a hybrid of the flute, electric guitar (see Sullivan), and mandolin models. Besides being insanely fun, the *blotar~* illustrates how the source-code functions of PeRColate can be used to create new

instruments; compare the *flute~* and *blotar~* code, and you will see. Also, the *bowedbar~* model is an interesting combination of waveguide and modal synthesis techniques (see Essl and Cook).

Modal Synthesis
This class of synthesis is inspired by various struck-bar instruments. All three instruments (*marimba~*, *vibraphone~*, and *agogo~*) feature 4 strong resonances, modeled here by biquad filters.

PhISM
Physically Inspired Sonic Modeling. These are totally cool. Lots of little things banging into each other. Shakers, water drops, bamboo wind chimes (See Cook, 1996, 97). The course code for these is a bit messy, but so are the sounds, so it's appropriate.

MaxGens
These are ports of some of the makegen() commands from Real-Time Cmix (Garton and Topper, 1997). They are Max objects that simply evaluate some basic synthesis and signal processing functions given a list input of parameters, and should look vaguely familiar to users of synthesis languages descended from Music 4 (such as Cmix and Csound). They are designed to be used either with the *table* / *multislider* / *coll* objects or with *peek~*, and generate harmonic wavetable functions (*Gen9* / *Gen10*), chebyshev polynomial equations (*Gen17*), random number distributions *(Gen20)*, breakpoint functions (*Gen7*, *Gen9*, and *Gen24*), and hamming / hanning windows (*Gen25*). You can check out the RTcmix link (below, in the references) for more information.

SID
Synthesis Isn't Dead (it just sounds that way?). This is a collection of Luke's idiosyncratic objects for making some neat sounds with extremely simple signal-processing techniques (see the source code to see just how simple). As a note, almost all of these objects work with signals only (I haven't implemented multiple dsp perform methods to allow for floats in the inlets yet), so most of your control interfaces will at some point have to turn into signals in order for these objects to work. A brief run-down on the current objects:

- *absmax~* takes two signals and outputs the one which is farthest from zero.
- *absmin~* takes two signals and outputs the one which is nearest to zero.
- *chase~* is a three-way signal comparator, checking two signals against a third and outputing the closest and farthest as output signals.
- *escal~* is a rounding object for signals, which depending on how you configure it will turn your floating point signals into integer signals, which is useful for opening gates, creating step sequencers, etc.
- *flip~* is based on one of the ugens in James McCartney's SuperCollider language (McCartney, 1996), which wraps a carrier signal around a modulator signal if the absolute value of the carrier exceeds the absolute value of the modulator. Its useful for creating analog-ish distortion effects.
- *jitter~* randomly varies an input signal by a small amount. This object is very useful for exciting the PhISM instruments, some of which need to have their parameters being varied regularly in order to make sound (see the help files for the PhISM objects for details).
- *klutz~* reverses the order of samples in each vector, making some very unpleasant (but potentially useful) sounds.

- *random~* is an audio-rate random number generator with a variable positive range. You can use this object as a substitute for *noise~* in some situations, or with *escal~* to generate random signal-rate integers.
- *terrain~* is a wavetable scanner much like the *wave~* object, except that it works in fixed 'frames' which are set in samples, and it has a second inlet that controls the wave-terrain position as the object 'scans' from the first frame to the last.
- *waffle~* is like *gate~* with a variable threshold for which outlet the main signal is outputted (see the help file for details). You can use *waffle~* to do a lot of silly things, from audio-rate panning to fft/ifft spectral crossovers.
- *weave~* is a subharmonic oscillator that outputs pwm based on a count of zero-crossings of the input signal. If used with very clean signals it can make for a very low-latency pitch-tracking synthesizer.

Note: some of the SID objects have been effectively replaced by objects in MSP2. We've removed the %~ object, for example. The help files for the SID objects will note where there is an MSP2 object which does more-or-less the same thing.

Random DSP

These are a collection of signal processing objects that Dan uses in performance. It includes:

- the *munger~*: a granulizing delay-line. Takes a signal in and spits out a stereo signal of little (or big) grains, transposed, backwards/forwards, enveloped (sorta), with nifty pitch sieves to play pretty chords. The maximum number of voices is limited only by your CPU (up to some very high hardwired limit), and can be changed smoothly while running. A maximum delay-length is set with the argument, and the working delay-length can be swept smoothly. Check out the help file, which actually explains what most of the parameters mean. Good thing, because I keep forgetting.
- the *scrubber*: a scrubbing delay-line. Really simple, and almost stupid. Takes a signal in, which you can scrub forward and backwards. To avoid clicking, *scrub~* uses a silly scheme which divides the delay line into three buffers. Basically, *scrub~* records into one buffer, while scrubbing through another; when the record buffer is full, the play head jumps to the record buffer (with some amount of overlap), and the record head moves to a third buffer. Each time the buffers switch, *scrub~* uses a ramp to record into the new buffer to avoid clicks. Users can now sweep the delay length, the amount overlap when switching buffers, and the ramp length when recording. These new parameters make *scrub~* a *much* more interesting object than in the original release. Try changing the delay-length while scrubbing with extremely high rates. Silly fun.
- *gQ~*: a really really nice filter (actually, a biquad with fancy coefficients) for doing equalization. Unlike most EQ filters, gQ completely decouples the primary parameters (center frequency, bandwidth, gain), making it easy to handle. This is from Dan's SGI application of the same name (see link below). *gQ~* now accepts an argument that establishes a number of filters in series, all of which are independently controllable. Check out the little subpatch in the *gQ~* help file which makes cool use of the great new Max4 LCD object.
- *dcblock~*: yep, that's what it does. It's good to put these before things like the munger and scrub, to avoid ugly clicking, unless you like clicking.
- more coming soon.....

Luke's Random Stuff
These objects are bits an pieces from Luke's collection of useful things. They include signal rate objects to convert MIDI note values to and from frequency (*mtof~*/*ftom~*), an abstraction to prevent dsp chain feedback loops (*pause~*), and a little object that prints things into the Max window (*post*).

PeRColate Nato
These objects are a collection of image processors for the Nato extensions to Max. They include:

- *242.cga*: does channel-by-channel bit quantization on an image.
- *242.colorspace*: implements Rafael Santos' toolkit for converting to and from alternate colorspaces.
- *242.constrain*: constrains the rgb ranges of an image within (or outside of) user-configurable boundaries.
- *242.cutout*: a simple 2-source image mask.
- *242.eclipse*: Luke's infamous meta-imaging algorithm.
- *242.eclipse02*: a 2-source version of *242.eclipse*.
- *242.eclipse03*: a variation on the *242.eclipse* theme, with threshold tinting added.
- *242.fromage*: does simple linear wipes between two images.
- *242.imgmatrix*: an image matrix and router object optimized for use with the *matrixctrl* Max user-interface external.
- *242.keyscreen*: a three-source patchable chroma keyer.
- *242.modgain*: a channel gain control with wraparound.
- *242.rene*: a three-source gradated keyer written for Rene Beekman.
- *242.rgbavg*: an idiosyncratic pixel averaging filter.
- *242.rgbavg02*: a 2-source version of *242.rgbavg*.
- *242.rgbseek*: searches an input image for a specific color, allowing you to trigger Max events.
- *242.traffic*: a tristimulus color tinting implementation that allows you to tint an image using a 3x3 coefficient grid.

References

G. Essl and P. Cook, "Banded Waveguides: Towards Physical Modeling of Bowed Bar Percussion Instruments," Proc. of the International Computer Music Conference, Beijing, October, 1999

P. R. Cook, "Physically Informed Sonic Modeling (PhISM): Percussive Synthesis," Proc. of the International Computer Music Conference, Hong Kong, Sept. 1996.

P. R. Cook, "A Meta-Wind-Instrument Physical Model, and a Meta-Controller for Real Time Performance Control," International Computer Music Conference, San Jose, Oct., 1992.

P. R. Cook, "Physically Inspired Sonic Modeling (PhISM): Synthesis of Percussive Sounds," Computer Music Journal, Volume 21, Number 3, September 1997.

P. Cook and G. Scavone, “The Synthesis ToolKit (STK), Version 2.1,” Proc. of the International Computer Music Conference, Beijing, October, 1999.

R. L. DuBois, “RTcmix Online Documentation.” http://www.music.columbia.edu/cmix

B. Garton and D. Topper, “RTcmix – Using CMIX in Real Time,” Proc. of the International Computer Music Conference, Thessaloniki, September, 1997.

J. McCartney, “SuperCollider: a new real-time synthesis language,” Proc. of the International Computer Music Conference, Hong Kong, September, 1996.

G. Scavone and P. R. Cook “Real-time Computer Modeling of Woodwind Instruments,” Proceedings of the International Symposium on Musica Acoustics, Acoustical Society of America, Woodbury, NY, 1998

C. R. Sullivan, “Extending the Karplus-Strong Algorithm to Synthesize Electric Guitar Timbres with Distortion and Feedback,” Computer Music Journal, Volume 14, Number 3, Fall 1990.

D. Trueman, “gQ,” http://www.music.princeton.edu/~dan/gQpage/gQ.html

Object Index

clar~	clarinet physical model.
dcblock~	dc-bias remover.
escal~	signal rounder.
flip~	signal wraparound / inverter.
flute~	flute physical model.
ftom~	frequency to MIDI conversion.
gen5	exponential breakpoint function generator.
gen7	linear breakpoint function generator.
gen9	computes a sinusoidal wavetable.
gen10	computes a harmonic wavetable.
gen17	solves chebyshev polynomials.
gen20	random function generator.
gen24	scalable breakpoint function generator.
gen25	hamming/hanning function generator.
gQ~	stereo filter.
guiro~	guiro model.
jitter~	signal randomizer.
klutz~	abuses signal vector ordering.
mandolin~	mandolin physical model.
marimba~	modal struck marimba model.
metashaker~	interface to the seven main PhISM models.
mtof~	MIDI to frequency conversion.
munger~	granulating delay-line.
pause~	prevents dsp chain feedback loops.
plucked~	plucked string model.
post	prints anything in the Max window.
random~	audio-rate random number generator.
scrub~	delay-line scrubber.
sekere~	sekere model.
shaker~	maraca shaker model.
sleigh~	sleighbells model.
tamb~	tambourine model.
terrain~	simple linear wave-terrain model.
vibraphone~	vibraphone model (modal synthesis).
waffle~	signal crossover.
weave~	sub-harmonic oscillator.
wuter~	water droplet model.