Obiwannabe

Use the source...

Sponsored by the number 42

Tea

Technically tea is not an elemental force, but it is a strong brownian motion producer and as will become clear this, and other seemingly unrelated things are in fact all deeply connected in the rich tapestry of the universe.

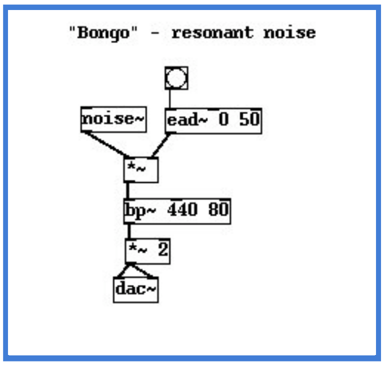
Before moving onto the next primal force it's time to visit the concept of formants and then look at noise and spectral distribution some more. These things we need to understand how better to use filters to create bodies of material and objects which can change shape and size. In a complex sound the spectrum changes through time. Sometimes, as with additive synthesis we can express the sound as a purely generative method. By this I mean that the total signal can be expressed as a function of time, say through a bank of oscillators each providing a contributory signal. But there is another abstraction in synthesis we haven't looked at yet, time dependent functions of time dependent functions. In classical sound theory we often call these processors, a familiar example is reverb. But there are many other ways of chaining together functions that operate on an existing signal to produce new sounds. One of these is the humble filter. We have briefly examined modes, the paths sound waves travel as they bounce around an object when building the additive bell. Let's look at one way we can approximate this using filters.

Hitch-hikers guide to resonance, filters, impulses and formats

A bandpass filter is like a pendulum, and if you are familiar with electronics like a LC circuit made from an inductor and a capacitor, the basis of the first filters. It is a system in which energy can freely exchange between two states, neither of which can remain stable indefinitely. In a pendulum an input of energy, in the form of a push becomes kinetic, moving energy. For a while the pendulum swings but slowly it gives up its kinetic energy storing it as potential energy. At the high point of it's swing there is no kinetic energy at all, for a brief instant the pendulum hovers, frozen in space filled only with the potential to fall again. Since there is nothing to stop it the pendulum falls again losing potential energy and regaining movement. Backwards and forwards it goes swapping the energy it holds between two forever unstable states.

In an electronic resonant circuit kinetic and potential energy are replaced by electromagnetic and electrostatic charge. The coil stores energy as a magnetic field, but it cannot hold it forever. Eventually the field collapses and current flows through the wires into a capacitor which then holds onto it for a while as a cloud of electrons on its plates, but as you guessed this is not stable, after a moment the electrons start moving again the opposite way creating a current through the coil and causing a magnetic field. This continues until all the energy is lost by wastage, or damping, in the case of the pendulum air resistance, and for the electronic circuit resistance in the wires. Damping occurs in natural physical objects as vibrations are converted to heat in the material. In all cases certain parameters such as the length of the pendulum string, or the capacitance of an electrical component determine the rate at which energy flows back and forth, giving the system a particular frequency at which it tends to move. Unsurprisingly the equations that describe all of these processes share a lot in common. When a signal or vibration encounters a resonant system it gives up some of its energy. If the frequency of the signal matches, or is very close to, the resonant frequency of the system something interesting happens. It gives up more and more energy to the resonator, the resonant body absorbs energy like a sponge. Imagine pushing a pendulum or a kids swing perfectly in time with its natural rythmn, each time you add only a small push but the swinging body quickly attains a large amplitude. We say that the signal is sympathetic with the resonance of the system.

Here is a simple circuit to make a sound a bit like a bongo drum being hit. It's just a quick snap of white noise gong into a filter with a high resonance. Notice there is no oscillator in the patch. So where does the pitch come from? It comes from the filter resonating or "ringing" at it's main pitch. This is basically the circuit used in old analog drum machines like the Roland 808. It works because noise contains every frequency, so some bit of the noise matches the resonance of the filter and sets it ringing. Notice that sometimes when you hit the bang button the sound is a bit different, not as strong or pure, that's because noise contains frequencies at random, sometimes there's more of those that match the filter than others.

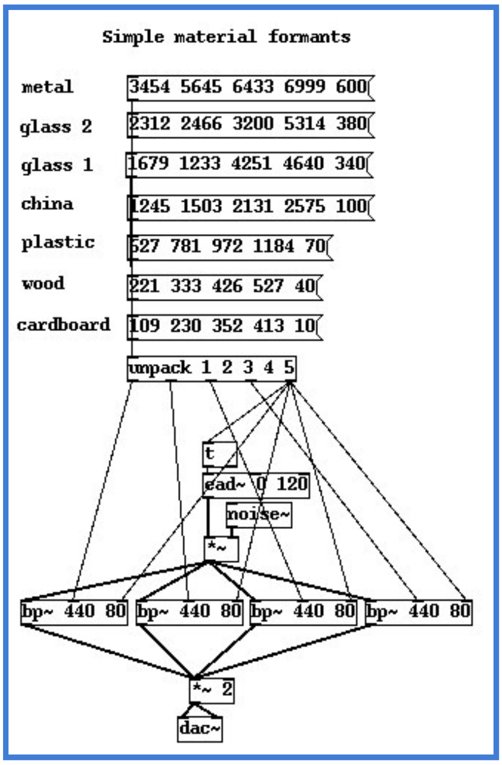


Puredata file .pd

The modes of resonance in a real physical body is like a big network of bandpass filters all connected together. You can picture the structure of a piece of wood, or a metals crystalline lattice like balls connected together by elastic rods, rather like those models you see in chemistry to demonstrate a molecular structure. Each pair of nodes and each path made up of many nodes can be reduced to a system bandpass filters which mimic its resonant behaviour. We find that most real systems have not one resonant mode but many. But regardless how complex the system may be it's possible to use laws of reduction, like Kirchovs rules for reducing electrical networks or DeMorgans rules for reducing logical systems, to reduce a complex body with millions of tiny resonant points connected together and approximate it by just a handful of bandpass filters. The minimum set of these, which kind of describes the tone of a body when struck, is called a formant filter, and the characteristic tone of a body is its formant. In DSP we talk of hitting a circuit with an idealised impulse response, a single short sharp pulse of energy to excite a body or circuit into vibrating in its natural modes. The sound we hear when we do this is not the sound of the impulse, an ideal impulse is too short to hear, but instead we hear the resonances of the body as impulse energy bounces backwards and forwards. Plenty of designs for synthesising a sound break the process into a signal source called the exciter, and a body resonance, called the formant. Of course it's not always so simple. Sometimes the output of one resonant system is the exciter to the next, and sometimes elaborate feeback loops where the output of a resonator affects the excitation signal occur so that complex behaviour emerges.

What does all this theory mean in practice? Well it means that for certain materials like wood, or metal, or glass we can create a set of filters which mimic the behaviour of that material. If we feed another signal into the filter the sound we hear is as if the signal had been played through an object made of the material. That's great news for sound designers, it means we can start from a real world description of an object and begin to model it digitally only by considering what shape, size and material parts of it is made from. The classic example given in music technology literature is the violin or guitar. A steel or nylon string vibrating on its own doesn't have so much character, it's quite a pure and dead sound. Much of the sound of a violin comes from the body formant, the wood resonating with the string. In sound design we want to model all kinds of objects, some far more complex than a violin.

The most common practical form is a parallel line of filters. Sometimes bands overlap in frequency, but generally a formant is designed to pick out quite specific frequencies and so each filter has a high resonance and picks out a narrow band. Modifying the above patch a little gives us this parallel formant filter that roughly creates the sounds of a few materials. If you hit each message it is unpacked to set the filter frequencies and resonances and then fire the noise envelope. Roughly speaking more dense and elastic materials have a higher resonance and frequency.



Puredata file .pd

Noise distribution

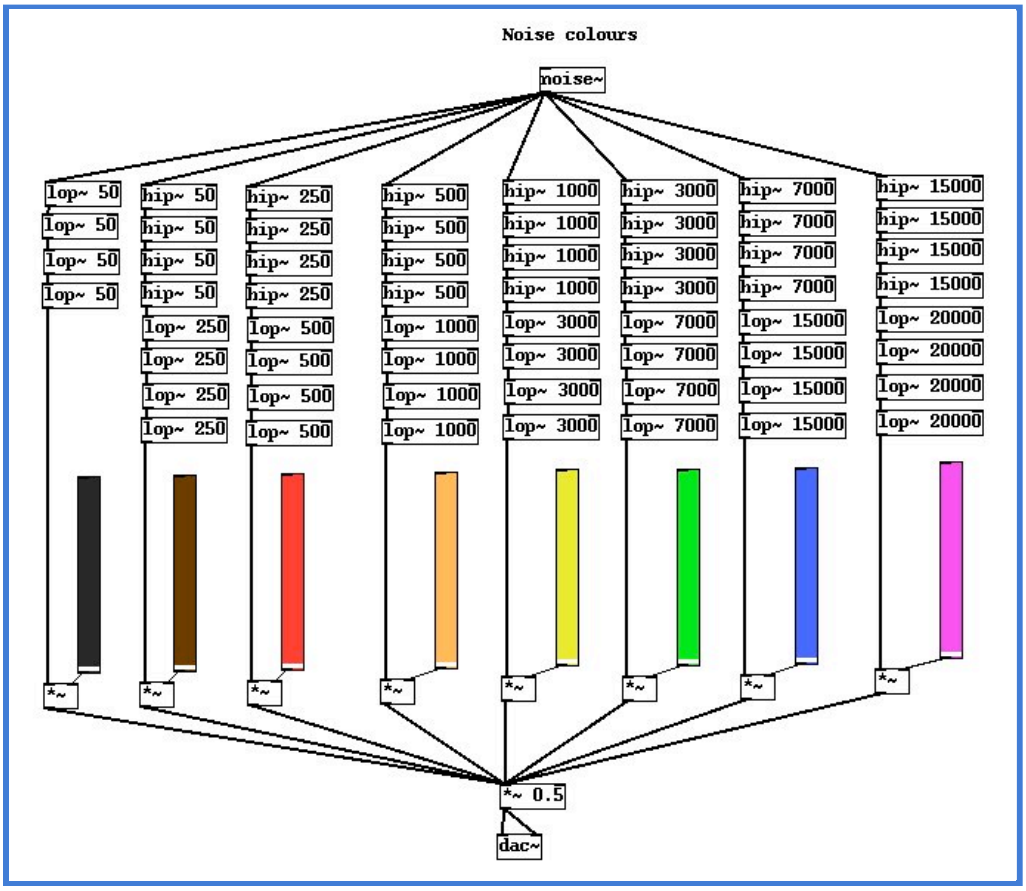
Noise comes in many flavours of which white noise is just one. If you are familiar with statistics then noise should make sense in terms of distributions. White noise contains evenly distributed frequencies, that's to say there's as much chance of finding a frequency of 1000Hz in any 1 second slice of white noise as there is of finding 12000Hz or 3Hz. The likelyhood of finding a certain frequency in a bit of noise is often talked about as the energy in a band. Pink noise contains the same likelyhood of finding a frequency in an octave interval as in the octave above it, so for example the chance of finding all the frequencies betwen 500Hz and 1000Hz is the same as that of finding those betwen 1000Hz and 2000Hz which is the same as that of finding all the frequencies between 2kHz and 4kHz, and so on. We say each octave contains the same energy. The spectra of noise has a lot to say about the underlying mechanism, the thing causing it.

There are no hard and fast rules about mapping colour names to noise, but the analogy to light is generally observed, so for example blue noise would be higher in frequency than yellow noise which would be higher in frequency to red noise. If we take this analogy in a sensible fashion we get roughly the following useful handles for noise bands, black and brown are added from the common cable and resistor colour coding scheme as values below red. These 8 ranges work out nicely to roughly cover the 10 main octaves of the nominal hearing range. The values place the centre of the scale roughly where the middle peak of the optical scale also happens to be (since our eyes are most sensitive to a colour which is green).

Colour	Range	Sounds like
Violet	15-30KHz	Gas escaping, angry snake
Blue	7-15KHz	Light breeze, aerosol, hihats
Green	3kHz-7KHz	Bacon frying, acid on concrete
Yellow	1kHz-3KHz	Rain, distant waterfall
Orange	500Hz-1KHz	River, city ambience
Red	250Hz-500Hz	Airplane cabin, train/car interior
Brown	50Hz-250Hz	Thunder, distant plane/rocket, large objets moving

Black	0-50Hz	Explosion rumble, near infrasonic, sub bass.

Other kinds of noise parameters are taken from mathematical functions common in statistics and signal analysis. Gaussian noise, binomial noise, power law noise are examples. These describe how the shape of the noise spectrum fits over the full range. When designing control level features of a sound, like the relaxation sequence for our bubbles or the distribution of raindrops these patterns of noise can have a profound effect if we choose the right one. Patterns that come up often when designing natural sounds are Gaussian (or bell curve) and exponentially distributed noise. See the references for more info. Some kinds of noise take their names from well known processes, natural or otherwise, which produce that kind of noise. Flicker noise, scatter noise, Miller noise (from Millers transistor theory) or line noise (crackling) are examples. Brown noise should not be confused with Brownian noise which is the random motion of molecules due to thermal energy. It was discovered by the British botanist Robert Brown in 1827 while looking at pollen particles through a microscope. Brownian motion explains why tea is hot, but not why it is brown, nor why Earl Grey a variety of black tea is neither grey nor black.



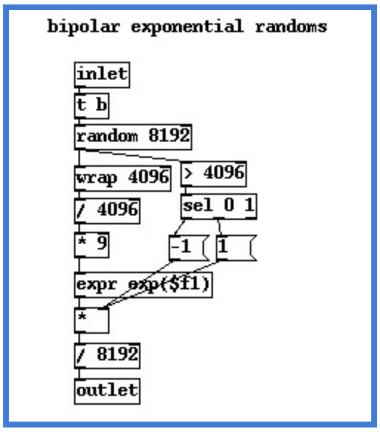
Puredata file .pd

Anyway enough nonsense. Armed with this knowledge we are ready to take another go at emulating running water. This exercise is optional, it's a little more advanced, but I highly recommend taking the trouble to prepare this highly refreshing cup of tea.

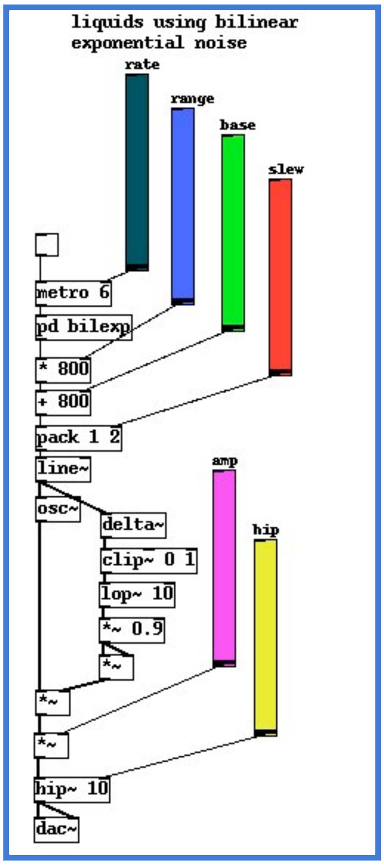
Our previous attempt was quite crude, but we did take note of the many parameters affecting the sound of running water, rate of flow and depth being two of them. One we have implicitly considered is the mode of excitation. Both the previous exercises have involved water sounds, but the bubbles were very different from the attempt at continuous flow. This time we are going to consider poured liquids, which falls somewhere between the bubbling sound and the flowing sound.

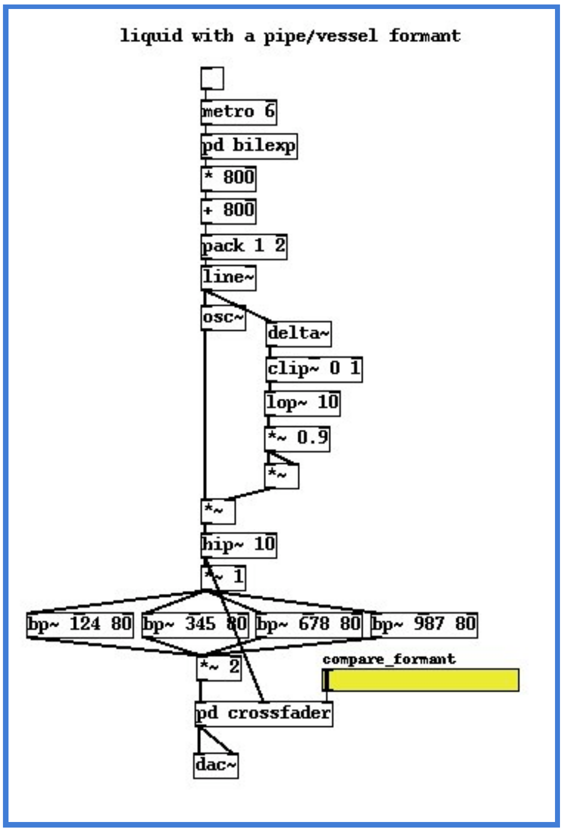
If you review the last sound example you'll notice the problem with too many high frequencies in comparison to low ones. A crude attempt to fix it with a bit of filtering made some difference, but there's still something not right about that running water noise. The problem is the distribution of noise in the source we used to create the sinewave frequencies. It doesn't match up with what happens in reality. In the real world sounds will observe certain physical laws, and one of these is Planck's energy rule. Waves which are higher in frequency contain more energy than lower frequency ones of equal amplitude. A photon of blue light contains more energy than a red one, and a burst of 20kHz contains more than an otherwise identical burst of 100Hz. A good way to look at this is to imagine that every wavecycle costs the universe a certain amount, and since a burst at 20kHz contains 200 times more cycles than a 100Hz tone it is more expensive in terms of energy. If the energy in a sound was distributed evenly then we would expect to get fewer high ones for our money (energy) than low ones, and that's what happens in a dynamic process like flowing water. Also loud high frequencies have a startling effect on the brain, they are a signal of potential danger because they indicate an exchange of a lot of energy, consequently they jump out and are much more noticeable than low frequencies, so the anomaly of a few more high frequencies than there should be isn't easy to ignore, even when this imbalance is relatively small in numerical terms.

To fix this we will start with another distribution of noise this time. We want exponentially distributed noise. Special external atoms exist to make a wide variety of noise distributions for PureData, but as we wish to avoid relying on components that might not be in every version we are going to make our own exponentially distributed noise. Fortunately it's an easy task. Let's take a the bog standard orthodox random number generator that gives us evenly spaced numbers. Now we will feed this through the exponential function and voila! Well not quite. See the PureData diagram below for reference. Since we only get integers and the exponential function grows so quickly we only get about 20 values that are in range and it disappears off to infinity. Ultimately we want floating point values in a normalised range, between zero and one, and e^9 is about 8103. The trick is to generate a fairly large integer number and divide it by another big one to get a small float. We choose values like 4096 and 8192 and 16384 a lot in digital work because when we come to divide these numbers there is a very efficient way to do it on a binary computer, that's why I chose those particular values so don't wory about them, you could use any number to scale by. Another thing is that we want random numbers exponentially distributed around a certain base, in both directions. To do this we could use another random generator with the values of 1 and -1 to give us a random sign. A more elegant way to get a bipolar signal is to make twice as many random numbers as we need and map half of them onto negative values. We could do that by subtracting half the range, but the reason for the slightly more complex method shown is that it's really useful to have a sign signal, we'll be needing that later.



Okay, let's put the new random generator into a familiar system and hear how it sounds. Notice the big improvement already?





Puredata file .pd

Puredata file .pd

Flow and drips

Parameters for rate, base and range are pretty easy to understand. They set the tempo at which new random liquid movements happen, the approximate middle frequency of the droplets and bubbles and the low and high extremes of the random range. Have a play with the controls and get a feel for the variety of liquid sounds possible. A control that isn't so obvious is the slew. That's because it's codependent with our tempo rate. If you set it too high, slew > rate * 2, the effect of running water is replaced by something like distant birds singing. If you set it too low the drops become too percussive and clicks will begin. Somewhere between slew = rate/2 and slew = rate gets the best values for a sound that ranges between dripping and a constantly poured liquid. We can change the rate at which the liquid apparently flows.

The teacup

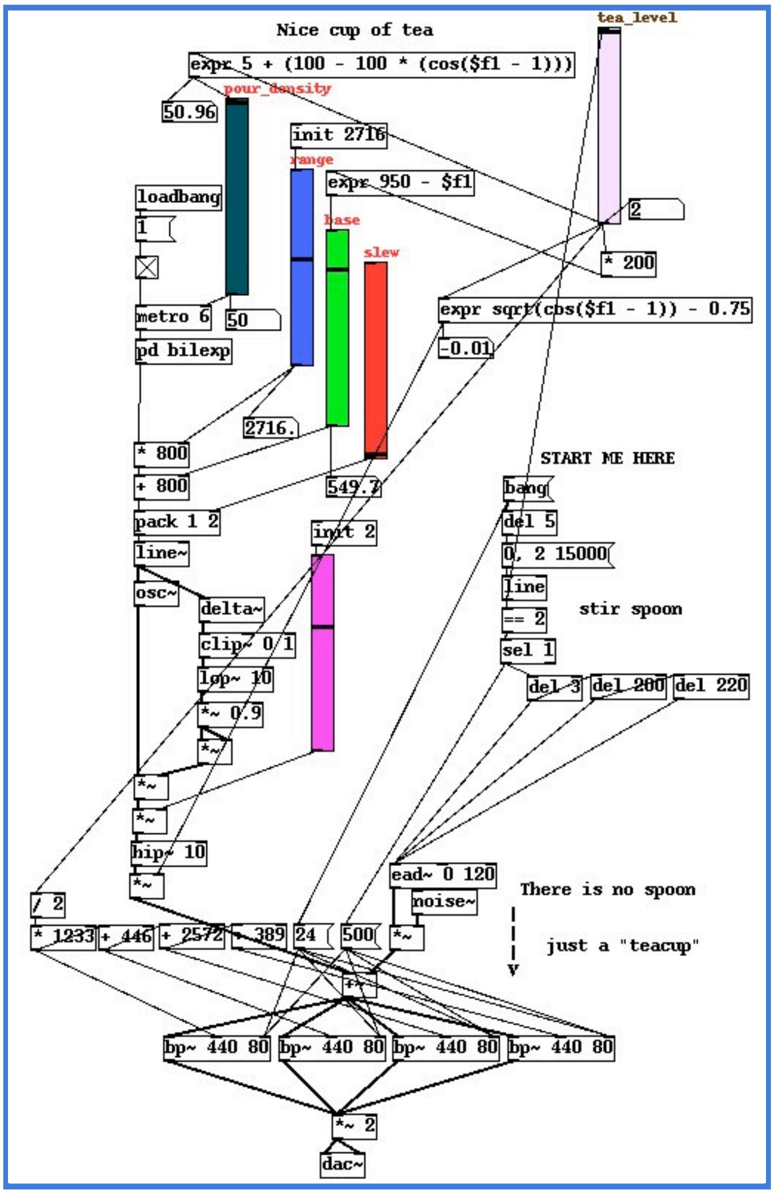
Beside our new exponentially distributed liquid above is a second diagram and file in which we pass the liquid sound through some kind of filter. This filter is an attempt to emulate a teacup, or rather the space above the liquid in a teacup. I've given this example a fixed formant and a crossfader so you can move between the unaltered liquid sound and the sound of it "inside a cup".

Pouring

Of course the space left in the teacup depends on how much tea is in there, and that depends on how much we have poured in. How much we have poured in is a function of the rate of pouring and time. How deep the tea is varies in time too, in inverse proportion to the space left in the cup. While this circus of linkage may seem confusing it isn't really, the trick is to abstract out a parameter on which all others depend and then determine the simplest relationship of the remaining parameters to it. We are going to move the rate of flow (pour density), the volume (amplitude), the depth of the tea (base) and the space left in the cup (formant offset) all in accordance with one imaginary parameter (tea level) which varies linearly in time between empty (0) and full (2). To do so we use a neat facility in Puredata called the [expression] atom, which basically lets us insert arbitary code of a functional kind. In an expression $f1 is an input variable, and we can have as many of these as we like enumerated $f1, $f2...$fn all going into different inlets. We get just one outlet though which is a function of the inlets. In the case of the pouring rate/density it's a semi-circular curve that peaks in the middle while the liquid base frequency (depth of tea) and formant offset (space in cup) are reciporacle.

There is no spoon

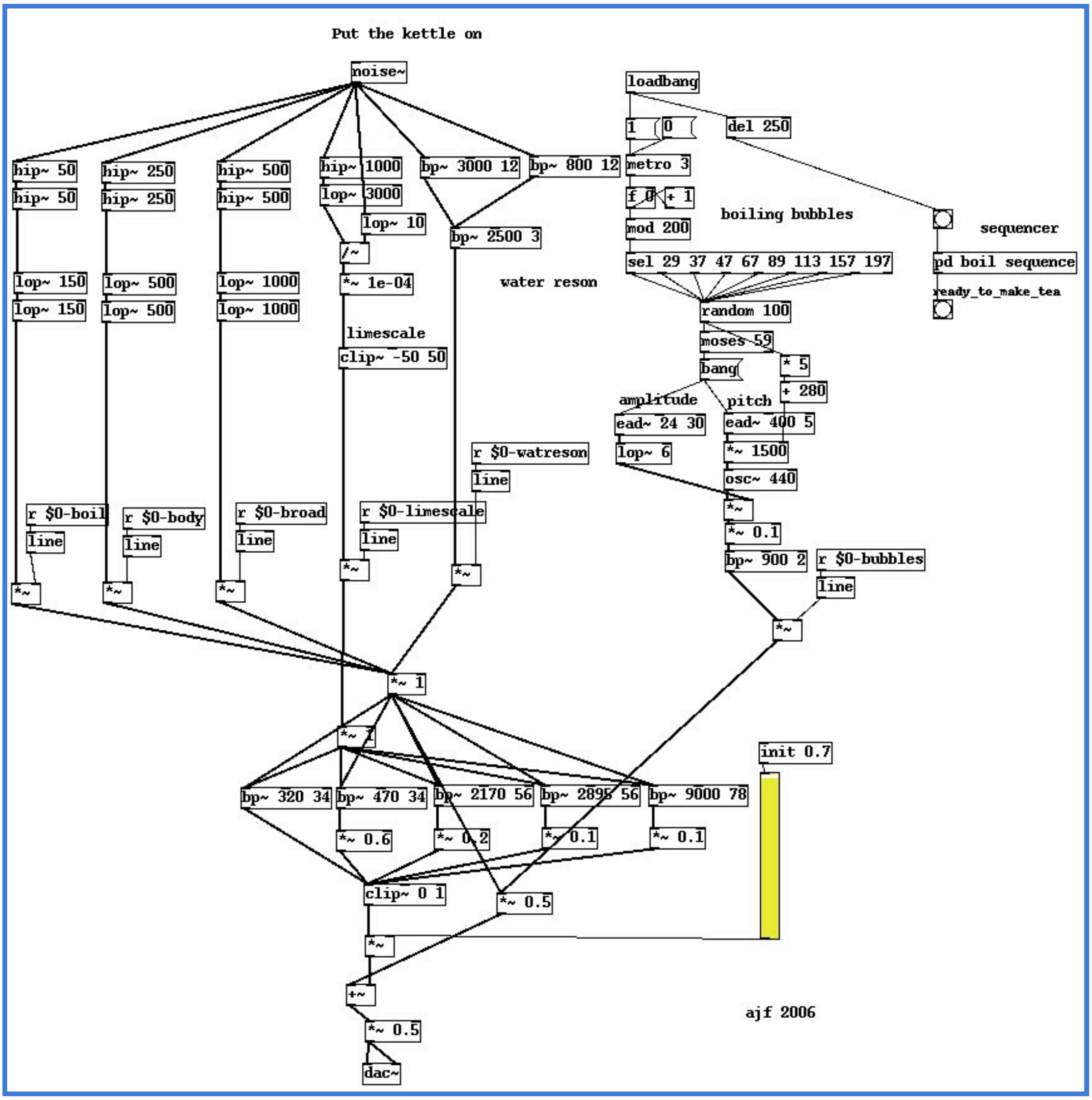
Finally we just want to show off our kung-fu. Since we've already got a teacup adding an extra effect of a spoon is quite trivial. Once the liquid in the cup is full a delay sequence is triggered which hits the cup with a few bits of white noise, that does a jolly good job of improvising a metalic impact on the side of the vessel.



Puredata file .pd

The kettle

How is one to make tea without a kettle? Obviously the preceeding patches won't work until we add a supply of hot water. Fortunately captain chaos is our friend and saves the day with nothing more than a few bands of carefully arranged noise.



Puredata file .pd

Chaos in the kettle

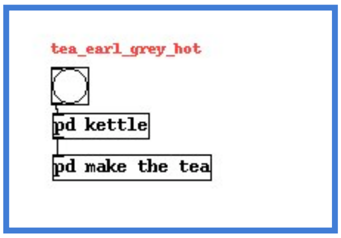
As the temperature of liquids increases the size of bubbles increases too. In boiling water massive bubbles of water vapour emerge, but long before that other dissolved gasses like CO2 and O2 come out of solution. The physical and chemical dynamics are pretty complex and probably not worth us going into too deeply here, but the net effect is a gentle crossfade between high frequency white noise and deep brown noise that roughly follows the temperature. Gasses also tend to collect around particles and rough solids in the early stages. In most countries your water probably has dissolved calcium carbonate so you'll be familiar with limescale, pieces of which cause violent "knocking" sounds when they warm up close to the heater. We acheive this with our old friend the divide operator, some clipping and a few narrow filters. For the rest of it I've just borrowed from the earlier patch a few bands of coloured noise. The entire kettle boiling is a mini sound scene in which each band fades in and out slowly. The sequencer to do this can be achieved several ways, with tables or functions or, as in this case, line segments. The details of the sequencer are unimportant although I encourage you to examine the patch thoroughly and see how it's done here.

Bringing it to the boil

You should notice a familiar pattern above. The bubbles we created earlier make another appearence. They are tuned a bit lower and made somewhat more dense but it's just the same code from before cut and pasted in. (Now you start to see the stitching on the seams. The cheapness! The cheek of it!)

Afternoon tea

Some good cakes and the newspaper are all we need now to complete a perfect afternoon. All that remains is to connect our kettle to our teapot. When the water has boiled a notification event in the guise of a single bang message informs the tea making routine to commence. This represents our first "sound scene", something comprising more than one component arranged in some causal or logical way. If you examine the Puredata file you will discover one or two other small touches, a mechanical switch noise for the kettle and some mains electricity hum for the element are little details that help complete the picture. I think you are at a stage where you can study and understand these for yourself without further explanation.



```
tea_earl_grey_hot
pd kettle
pd make the tea
```

Puredata file .pd

Section summary

We've covered a quite a lot of stuff in this section. Bubbling, boiling, dripping, pouring and running water. We've also touched on important design concepts like formants, spectral distribution and resonance. Some physical processes, and our perception of them are topics we've touched on once or twice and lastly we have integrated a couple of generators we built into a larger scale soundscene.

The next section celebrates another great British tradition. Shitty weather.

Links

Next next tutorial

Top tutorials list

*** (unproven) third moment of the Riemann zeta function**