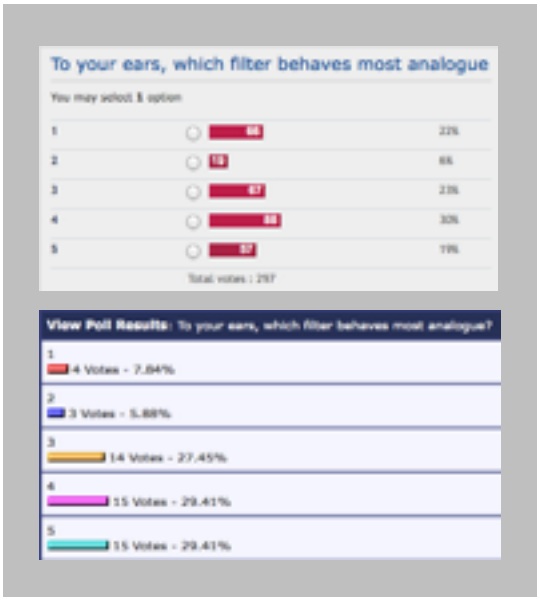
What makes No. 3 the best?

This is for everyone who followed our RePro-Alpha poll on various forums, most prominently here:

on KVR: http://bit.ly/1pWKQfM
on Gearslutz: http://bit.ly/26KEFx0

In short, we released a plug-in which would calculate the same filter model using 5 different mathematical methods, and we asked which one people thought behaved “the most analogue”. More importantly, we chose to post them on public forums (see above) so we could follow discussions between users.



first things first

Here’s what the RePro-Alpha filter algorithms 1-5 actually do:

1. Pivotal approximation of non-linearities (cheap on CPU)
2. Unit Delay + Trapezoidal Integration aka “Predictor Corrector” (okay-ish)
3. Vectorial Newton-Raphson (expensive? maybe not…)
4. Pivotal (like 1) with Heun-like correction step (okay)
5. Unit Delay + Forward Euler Integration (like 2 but no corrective step, very cheap)

the challenge

All these methods seek to solve the same mathematical problem inherent in digital models of analogue synthesizer filters. While the equations describing relationships between voltages and currents in a filter circuit are relatively simple, there is no direct solution for realtime interactions within the circuit i.e. one can’t solve them by simply rearranging the terms.

We’re talking about “implicit” equations here, whose solution depends on the result itself. This sounds more esoteric than it actually is, as implicit equations can solve many everyday scenarios – such as avoiding excess baggage fees at the airport: OK, more stuff will fit into that huge suitcase, but its empty weight eats significantly into the airline’s 23 kilogram limit. We need to calculate the optimum size so we can pack as much stuff as possible while remaining below the weight limit.

When we released Diva we described “Zero Delay Feedback” (ZDF), the iterative process and its advantage over more conventional digital filter design. Diva used significantly more CPU than most synths at the time, and we needed to explain why. ZDF soon became a buzzword everybody and his auntie wanted to use in their advertising. Some claimed equal results using much less CPU, but the context of mimicking analogue behaviour was conveniently omitted, misrepresenting how we used the term “Zero Delay Feedback”. Of course this sparked tedious discussions…

Having defended my position too often since then, I decided to start a blog and set matters straight, once and for all. So whenever you need to know what e.g. “pivotal” or “trapezoidal integration” mean, please refer to https://urs.silvrback.com!

the setup

RePro-Alpha comes with 5 different algorithms, all emulating the Curtis 3320-based filter in the vintage Sequential Cicuits Pro-One. We analysed our Pro One, derived basic properties such as frequency bandwidth, modulation depth, harmonic distortion, DC-offsets, cutoff bleeding, self oscillation levels and spectrum, then put the results into some neat mathematical equations.

The 5 different algorithms only differ in the methods used to solve those equations, with one exception that also uses a different integration method to mimic the behaviour of capacitors.

All algorithms operate at the same sampling frequency: 8x oversampled at a host sample rate of 44.1kHz or 48kHz, 4x oversampled at 88.1/96 and 2x oversampled if the host sample rate is 176kHz or faster. Such a high sample rate is necessary for this filter.

As digital filters can behave erratically (“explode”), there is a post-filter sanity check which observes audio levels. To guard against extreme noise bursts at critical settings, the filter is reinitialized whenever an output sample exceeds +30dB.

Something we haven’t mentioned yet: The filter frequency in RePro-Alpha was capped at 25kHz. Using modulation, our CEM 3320 chip can be pushed up to 68kHz when self-oscillating, but the typical maximum is 40-45kHz. The accurate algorithm (3) can process the full desired range without excessive aliasing or artifacts, unlike any of the CPU-saving versions (which would either run into the sanity check or alias like crazy).

We wanted to make the test “blind”: Multiple algorithms are computed in parallel, making it impossible to determine the complexity of each filter model by checking CPU consumption. However, running all 5 algorithms in parallel seemed excessive, so they were divided into two groups, 1-2-5 and 3-4. When any filter is selected, all filters in that group are computed per sample.

about solutions 1 and 4

1 and 4 are both based on temporarily replacing the non-linear functions (curves) by a linear factor (straight lines). This turns the implicit non-linear equation system into a linear one which can then be rearranged and solved explicitly. While this solution preserves the delayless feedback path, the linearisation of the non-linear term is based on the previous sample step. One could say that while the linear part of the filter is a zero delay feedback solution, the non-linear aspect is indeed slightly delayed.

To minimise the effect of the sample delay, algorithm 4 computes the filter twice with a sample lookahead, and averages these results for the final solution. The idea is that if the first pass overshoots, the second pass will undershoot and vice versa. Taking the middle will be close to the correct solution (see Heun’s method).

The main problem with this method is that the non-linearities in our model are not as gentle as classical tanh() soft-clipping stages. Instead, they have a large linear area and a sharper corner, much like a diode clipper. This leads to a problem in which the linearisation error varies a lot over the range of values. If values are close to a “corner”, even a small change has a big impact on accuracy. The correction step helps significantly, but does not perform as well as it would for more moderate waveshaping functions.

As a result, 1 and 4 are very sensitive to the filter input level (drive) and resonance. At higher levels, the overall output gain is higher than it should be, the cutoff frequency is a tad off, and in some situations the sanity check kicks in, resetting the filter. These artifacts are often audible with model 1, but they also occur with 4 – not a problem of the filter algorithm per se, but of the special situation found in this model, and the necessity to protect the output.

An interesting observation: Some people wondered whether those artifacts were “analogue imperfections”. My tongue-in-cheek reply in the form of a question: Why jump through hoops to achieve the accuracy of analogue filter circuits if the analogue circuits were less perfect than existing digital models?

A majority of people found model 4 the most appealing. I think the higher perceived loudness played a role in this choice. A side effect of the exaggerated gain is more distortion, so 1 and 4 are not only a few dB louder, but are also driven harder. Unfortunately nothing like the sound of a Pro-One, even though the old Prophet synths have been said to sound a tad “fizzy”. Perhaps some who voted for 1 and 4 equated the extra drive with the familiar fizz of the old Prophets.

Also, as the output gain is larger during extreme settings, a fast filter envelope could create a “popping” attack. This sound is reminiscent of compression. Which may be another reason people prefer it, but then it’s not anything that an analogue filter does. It can be recreated easily and made better sounding with an actual compressor rigged up behind the synth.

Could we have done better? A third or fourth pass might improve the solution and reduce the aliasing and gain issue, but that could exceed the CPU hunger of the vectorial Newton method.

What we’ve learned from this is that the corrective step is a vast improvement over the plain pivotal method. It still has difficulties with “corners” in the waveshapers, but it may be the preferred solution for classical filters exhibiting tanh()-like smooth, silky distortion.

about solutions 2 and 5

Algorithms 2 and 5 deal with the problem outlined above in a different way. Instead of removing the non-linear parts from the equations, they remove the necessity to solve them implicitly. Instead of delaying the impact of the non-linear waveshapers, they keep the waveshapers as-is and delay the feedback paths by one sample.

These are often said to sound particularly smooth. However, there is no hiding the fact that artificial delays are introduced in every feedback path. A 4-pole ladder/cascade filter has 5 such feedback paths, one in each filter stage and one surrounding the whole filter. This causes a somewhat damped response to signal changes.

During discussions about the sense or otherwise of zero delay feedback filters, some experts pointed out that the unit delay becomes less apparent as the sampling rate increases. I often heard “why not increase the sample rate instead of that iterative solver?”. The answer is obvious: One would have to oversample many, many times more than people are prepared to pay for, CPU-wise.

I reckon that 32x oversampling at 44.1kHz will make the non-ZDF version of this filter behave almost as well as a 4x oversampled version with an iterative solver, but the solver will use only 2-3 iterations on average – which would compute a lot faster. For a rough estimation, create a patch that causes algorithms 1 and 5 to fall apart, then lower the modulation rate by 2 or 3 octaves. It will still sound wrong, but the sound might not disappear.

So, independent of the integration method (trapezoidal or Euler), filters with a unit delay in the feedback path somehow “lowpass” any changes in any signal. Whether we’re talking about fast modulation, sudden changes or dynamic states at high resonance, they all fall apart in extreme situations.

Some specialists note that real electronic components are not entirely delay-free, and thus a unit delay in feedback paths might just lead to more correct behaviour than zero delay. While it’s true that these components take a short while to settle, that time is typically only a nanosecond or two. Unless we process at hundreds of megahertz, zero delay feedback is much closer to the real behaviour than a unit delay of several microseconds.

about solution 3

This is the algorithm that’s always well-behaved. As I said earlier, it can easily deal with cutoff frequencies approaching half the sampling rate, where all the others fall apart – they either go silent or they explode. Or both!

Looking at the discussions in the forum threads, this mode is the least controversial. People disagree about whether algorithms 1 or 4 are “strong” or “harsh”, while descriptions of 2 and 5 range from “smooth” to “lifeless”.

Mode 3 is unspectacular, and this may be its strength. Does it really take that much more CPU? Yes, but no…

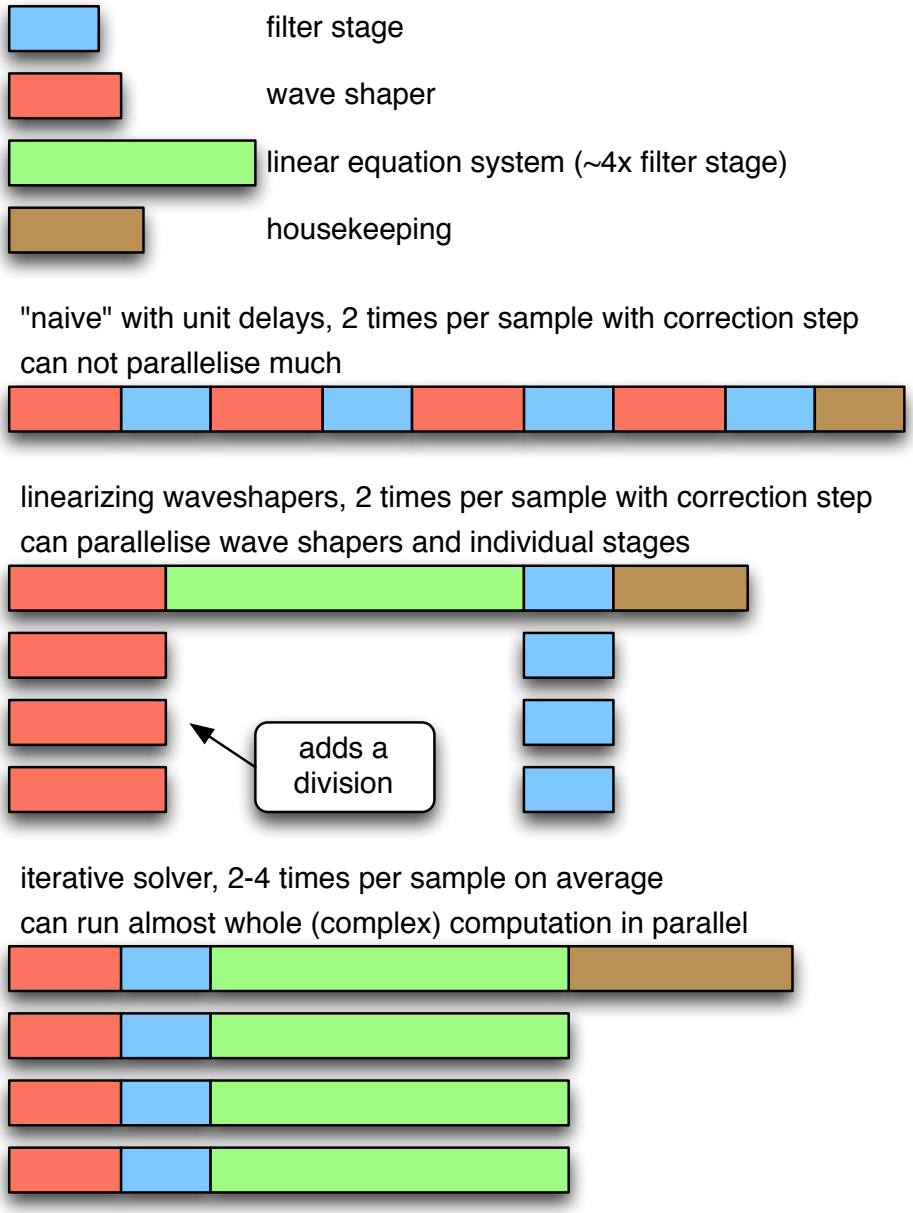
about CPU usage in general

The original motivation for this test was to check whether we can get away with less CPU intensive methods of filter calculation than we used in Diva (for instance). Admittedly, I announced this test a little prematurely, at a time where solution 3 was already shaping up nicely and I knew exactly how to approach the other modes.

However, when comparing the CPU usage of the various algorithms at 8x oversampling, the difference in computation time is not so dramatic. The actual oversampling factor may be the single most important contributor to CPU hunger. If such a high sampling rate is necessary to reduce aliasing, nothing can bring CPU down to the level of software that ran well in 2001.

Surprisingly, algorithm 5 isn’t so much faster because of the way it’s computed. Despite the delays in feedback, the expensive waveshaper calculation has to be done for each filter stage individually. In modes 1, 3 and 4 they can be computed in parallel. We could try to analyse this graphically…

Let’s say that the time to solve the whole filter linearly takes x cycles (green), and computing a filter stage takes y (blue) and solving a non-linearity takes z (red). Not forgetting a bit of housekeeping around the different filter modes (brown). We arrive at the following relation for a single run in each mode:



The interesting observation is that those filter algorithms which impose artificial delays to compute the non-linearities suffer directly from strong dependencies, dependencies which make it difficult to parallelise the computation.

While the other algorithms are more complex, they also lend themselves better to parallel computing (SSE optimisation). Although the complexity is higher, a single iteration is not necessarily slower. On the other hand, multiple iterations may be necessary to achieve good enough results.

the verdict

The main lesson we learned was that “differences exist”, but when all algorithms are implemented with a generally high quality, these differences manifest themselves mostly under extreme conditions. That said, even the most CPU-intensive and accurate algorithm does not use excessive CPU under “tame” conditions where these differences are barely audible. It only uses more CPU (say, twice as much) when the result truly counts i.e. when differences are clearly audible.

Otherwise there's not much difference in CPU load - it's pretty high even with supposedly CPU-saving methods.

The most efficient way to lower CPU consumption may be to reduce the internal sample rate, which should work fine in moderate settings. Again, this speaks in favour of algorithm 3 – the only one whose cutoff frequency can go way past 1/8 sample rate without side effects. So I guess we'll stick to mode 3 and offer extra "quality" modes based on the internal sample rate, if necessary. Furthermore, even at 8x oversampling (at 44.1/48kHz), only algorithm 3 can be opened far enough to not attenuate frequencies in the audible range (it's still capped in RePro-Alpha, though). As a 4-pole filter attenuates the signal by 12dB at the cutoff frequency, only a filter that can be set to 40+ kHz can be "fully open".

As for people's preferences or subjective findings, I think we have a tie between the 3 basic methods. The "smoother sound" of 2 or 5 may be preferrable in some situations, and so is the exaggerated/compressed sound of 1 or 4. Some people find the artifacts of the sanity check interesting, but maybe these would be better modeled in a different synth.

Concluding, it is interesting that while "yes there are differences", these differences were discussed very controversely by many people out there. Furthermore, we don't have the impression that CPU-saving algorithms actually save that much CPU once a certain quality standard is set. So what's driving our decision is not necessarily a preference or "bearable tradeoff" but the "more forgiving" nature of the accurate model: It's simply more versatile, and its ability to run at lower sample rates makes it the better trade-off for quality than using any of the other algorithms.

We shall see!

Thank you all for participating,

- Urs Heckmann, May 2016, www.u-he.com