

Nageru: Taking free software video mixing into 2016

Steinar H. Gunderson

FOSDEM, January 30th 2016

Hi! I'm Steinar. I wrote this thing. These are my speaker's notes; they're in no way word-for-word what I'm going to say at the conference, but the slides probably won't make a lot of sense without them.

投げる【なげる】 (v1,vt) (1) to throw; to cast away

Let's get the naming out of the way first. The name “Nageru” is a loose pun on the Japanese verb “nageru”; I liked its “cast” meaning, since Nageru is for broadcasting video. (Japanese speakers can despair now.)

投げる【なげる】 (v1,vt) (1) to throw; to cast away
(2) to face defeat; to give up;

Later, I learned that it has a second meaning. This wasn't my intention.

Primary goals:

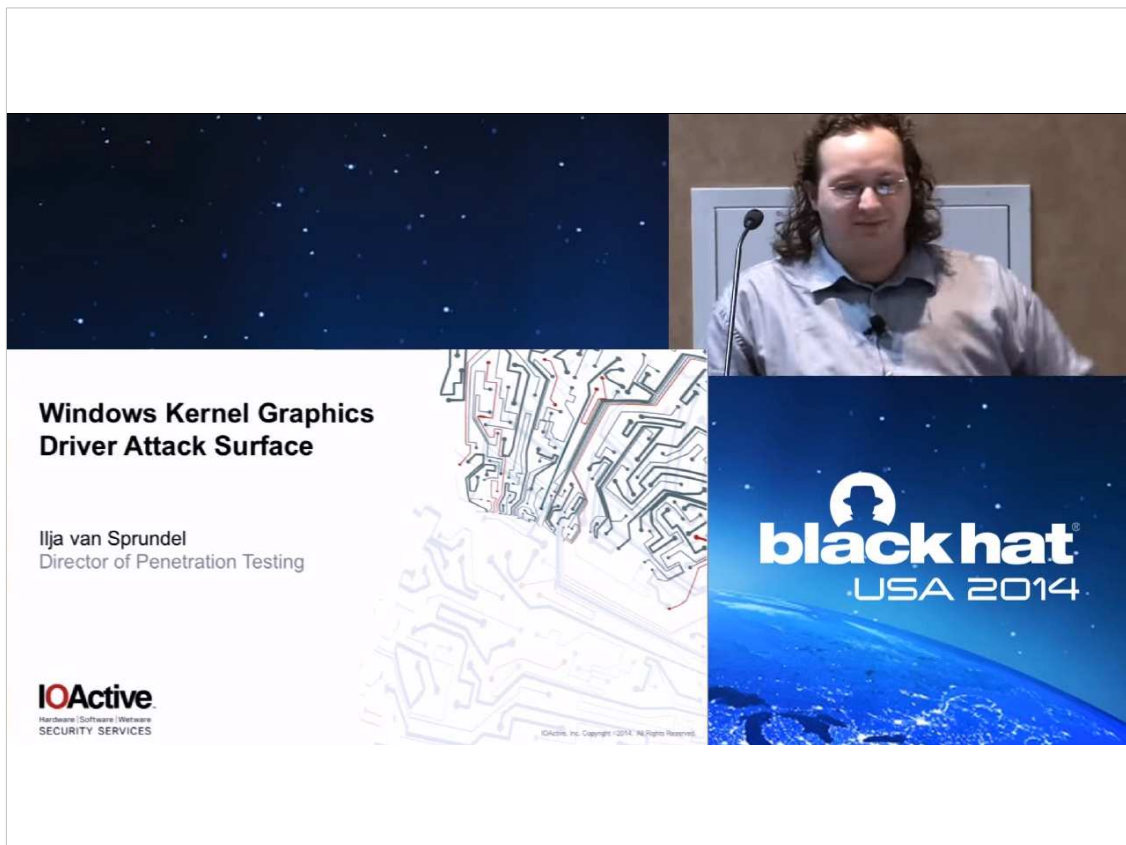
- High-quality
- High-performance
- Usable audio tools

Nice to have:

- Suitable for Debian main
- HTML5 overlay graphics
(and/or integration with CasparCG)
- A pony

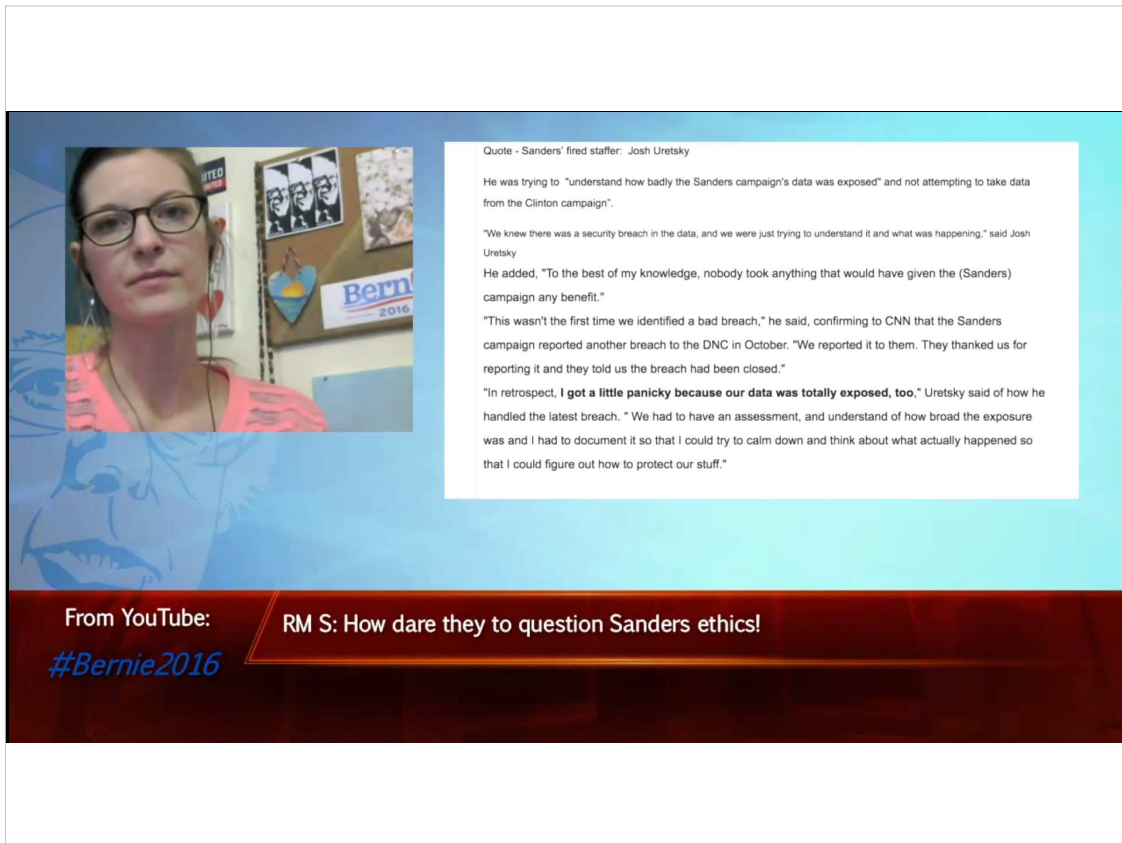
I promise you, this is the only bullet point slide you'll see in this presentation. But you can't really see why software ends up the way it is without understanding the goals that went into building it. So I want to get the list of goals out there as soon as possible.

Of course, others will start out with a different set of goals, and end up writing software with different tradeoffs. Personally I'm happy I managed to hit all of my primary goals and one of the three nice-to-haves (the first one; otherwise I would probably not be allowed to present at FOSDEM in the first place).



So, before I went out and built something, I tried to figure out what the quality bar is in the outside world, because frankly, the world has been doing television for (much) more than fifty years now, and perhaps the free software world should start by trying to learn something from that. So I tried to find video from conferences that were specifically not about free software. I scanned through conference video for hours on end, actually.

This is Black Hat USA 2014, obviously. It's pretty simple; it has a speaker camera, a (larger) slide view next to it, and some background graphics. And it's in 720p.



Quote - Sanders' fired staffer: Josh Uretsky

He was trying to "understand how badly the Sanders campaign's data was exposed" and not attempting to take data from the Clinton campaign".

"We knew there was a security breach in the data, and we were just trying to understand it and what was happening," said Josh Uretsky

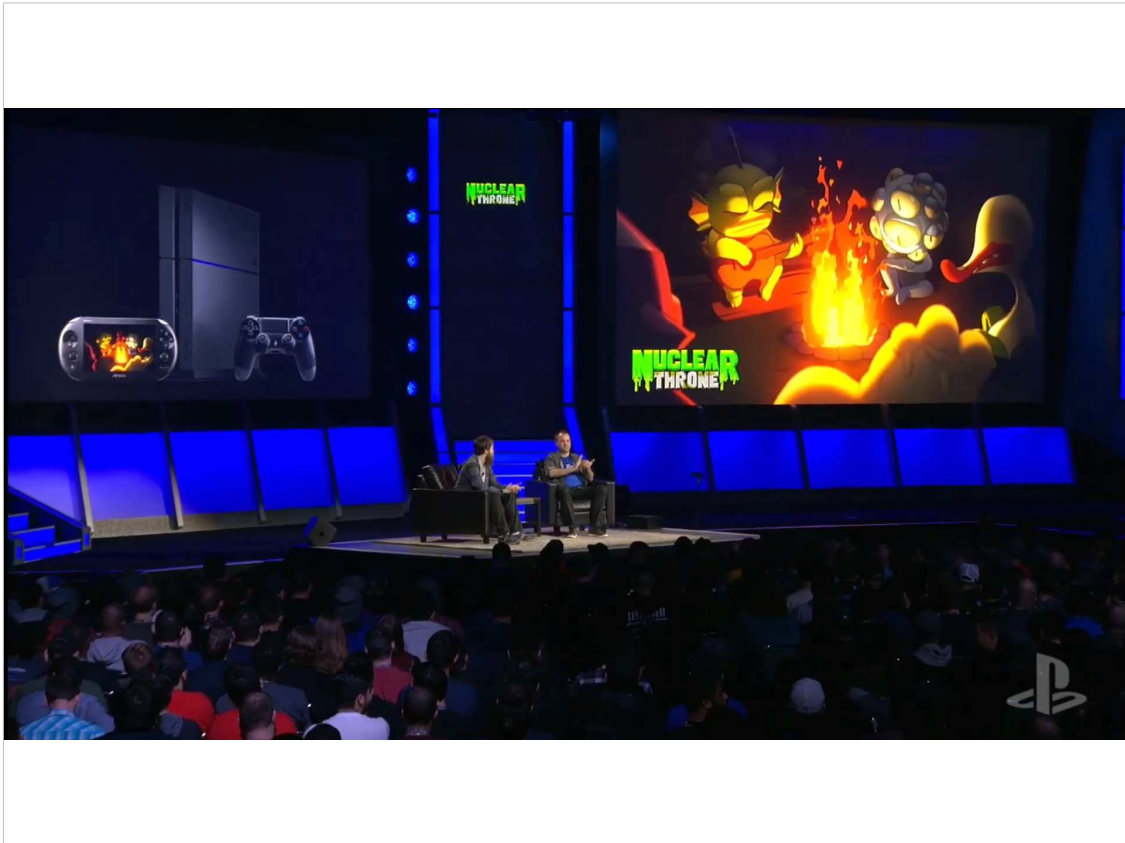
He added, "To the best of my knowledge, nobody took anything that would have given the (Sanders) campaign any benefit."

"This wasn't the first time we identified a bad breach," he said, confirming to CNN that the Sanders campaign reported another breach to the DNC in October. "We reported it to them. They thanked us for reporting it and they told us the breach had been closed."

"In retrospect, I got a little panicky because our data was totally exposed, too," Uretsky said of how he handled the latest breach. "We had to have an assessment, and understand of how broad the exposure was and I had to document it so that I could try to calm down and think about what actually happened so that I could figure out how to protect our stuff."

From YouTube: **RM S: How dare they to question Sanders ethics!**
#Bernie2016

For something completely different, this is from the coverage of a 2015 Bernie Sanders conference. I had no idea these things existed. But it's a camera (seemingly a webcam), some slides, an overlay, and a background. Again, it's in 720p.



If you have lots of money, you drop the digital compositing and just use large screens as backdrops instead. This is PlayStation Experience 2015; this kind of setup is probably a bit beyond what I am aiming for. It's 1080p.



Finally, I looked at broadcast TV to see what the real pros do. There's not a lot of broadcast TV that is comparable to conference video, but this is a Norwegian chess broadcast from TV 2, which at least has a similar pacing. (This is an SD capture, but it's sent in 720p.)

Obviously, if you watch the PDF, you won't see the video, but there's a lot going on here, much of it centering around overlay graphics. My main focus, though, is the transitions. The little boxes swish and swosh around in synchrony; there's obviously nobody doing this manually. It's all scripted to give a consistent, fluid look.

As far as I know, this is done in a commercial product called VizRT. VizRT has a WYSIWYG editor where you can create amazing things, but this is probably too much work for Nageru.

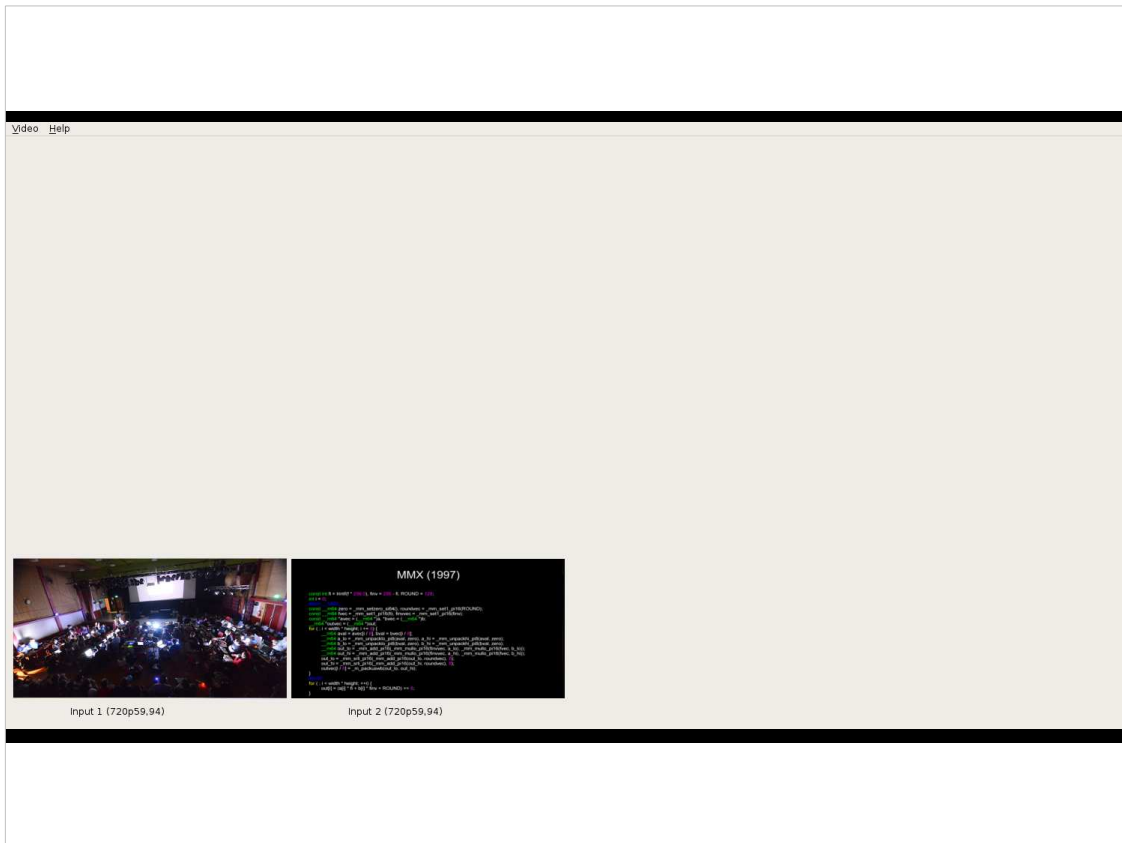


This is a black slide because it makes video navigation in LibreOffice easier. :-)



So enough about the outside world, let's actually talk about Nageru. This is the UI before we start adding anything to it; it's kind of bleak, so let's start placing widgets.

(By the way, this is a mockup; the live images are just pasted over a test input. But if you use the software in real life, you'll see it's almost identical to what's shown here.)

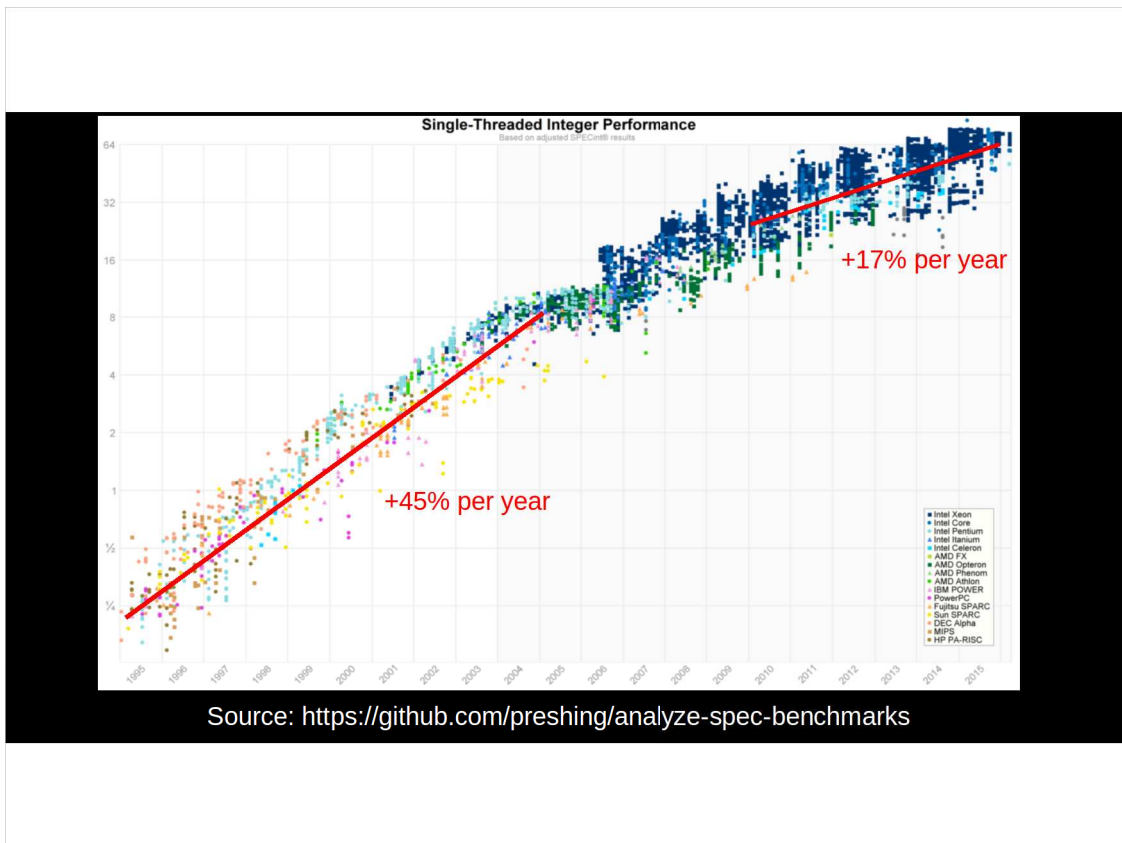


First of all, we need inputs. I've chosen to do 720p60 as the target format, for two reasons: First of all, 720p is the minimum that is really useful as of 2016 (cf. all the conference video we saw in that resolution), and 60 fps is just so much nicer than half-framerate (30 fps). Second, I wanted to make sure I could actually handle full frame rate, ie. not have too high per-frame overhead. It is only ~12% percent fewer pixels/second than 1080p30, so if you can do 720p60, you're very likely to be able to do 1080p30, too. It's just generally a harder target.



This is the Blackmagic Intensity Shuttle. It's cheap (\$199) and does almost everything you'd want; it has HDMI input with passthrough, 8-channel audio inputs, composite and S-video if you want, and an USB3 connector. There's also a more expensive SDI version if you prefer SDI devices. And finally, it's wildly popular and is still current, so getting more will be easy for years to come. The only sad thing, really, is that their promised 1080p60 support never came; it's max 1080i60, 1080p30 or 720p60.

Unfortunately, unlike most of Blackmagic's lineup, there's no Linux driver for it, only for the Thunderbolt version. I solved this by writing my own driver, `bmusb`. It runs from userspace via `libusb`. The protocol isn't hard, really; you just set some registers and then the card starts throwing mostly raw Y'CbCr frames at you (~80 MB/sec).



Input, of course, is no good without processing.

Here's a secret: Moore's law as we used to know it is effectively dead. CPUs don't get much faster anymore; for a while, we compensated by increasing the core count, but we don't do that anymore either (except for server chips). It used to be that we could write slow code and then just wait, but no more.

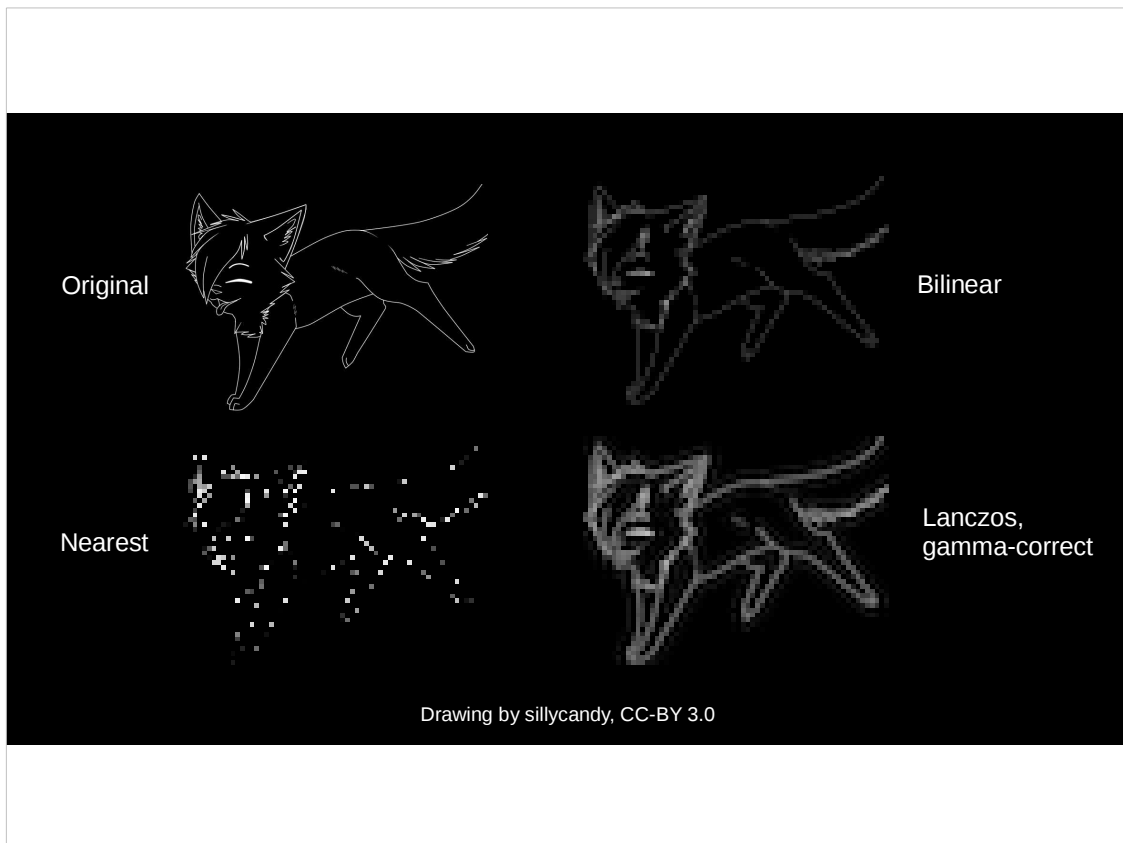
However, *GPUs* still get faster all the time! It follows that we should do as much processing as humanly possible on the GPU. That way, if we want 1080p, 4K or more inputs, we just need to buy a faster GPU, which is actually quite available these days. The ideal is that the CPU should never touch a single pixel; Nageru isn't quite there, but it's actually pretty close.

From: Steinar H. Gunderson
Cc: Markus Rechberger
Subject: [PATCH] Add support for usbfs zerocopy.

From: Steinar H. Gunderson
Subject: [PATCH] [libusb] Add support for persistent device memory

So let's start by removing one such touch. I picked up a patch by Markus Rechberger that had been floating around for a few years and got it into shape for mainline inclusion (currently scheduled for 4.6). It allows userspace to allocate a persistent chunk of kernel memory and have the USB host controller just DMA directly into that block. I also wrote libusb support for that interface (currently under review).

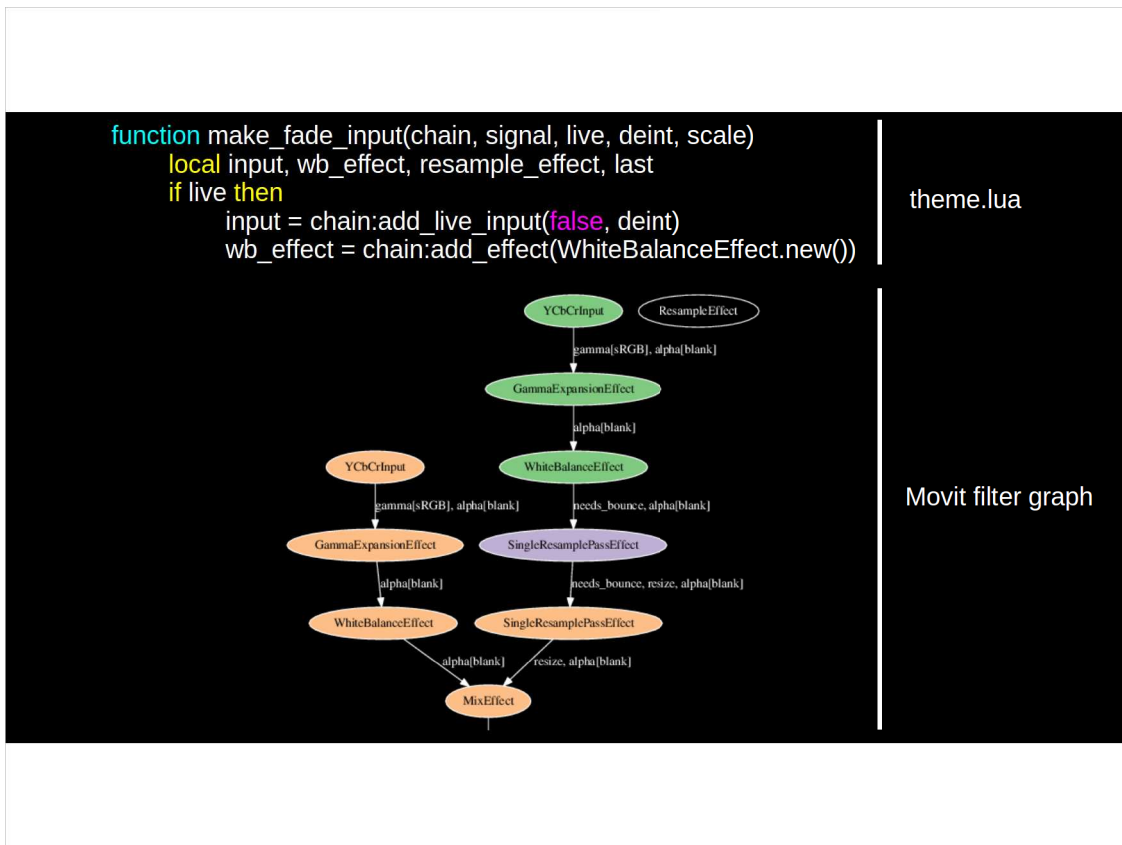
Not only does this save CPU usage, but it also fixes an issue where fragmentation of kernel memory could cause requests to fail after running for too long. It's quite cool, actually.



For the GPU processing, Nageru uses Movit (<https://movit.sesse.net/>), my own GPU filter library. I presented it on FOSDEM two years ago; see that presentation for more details.

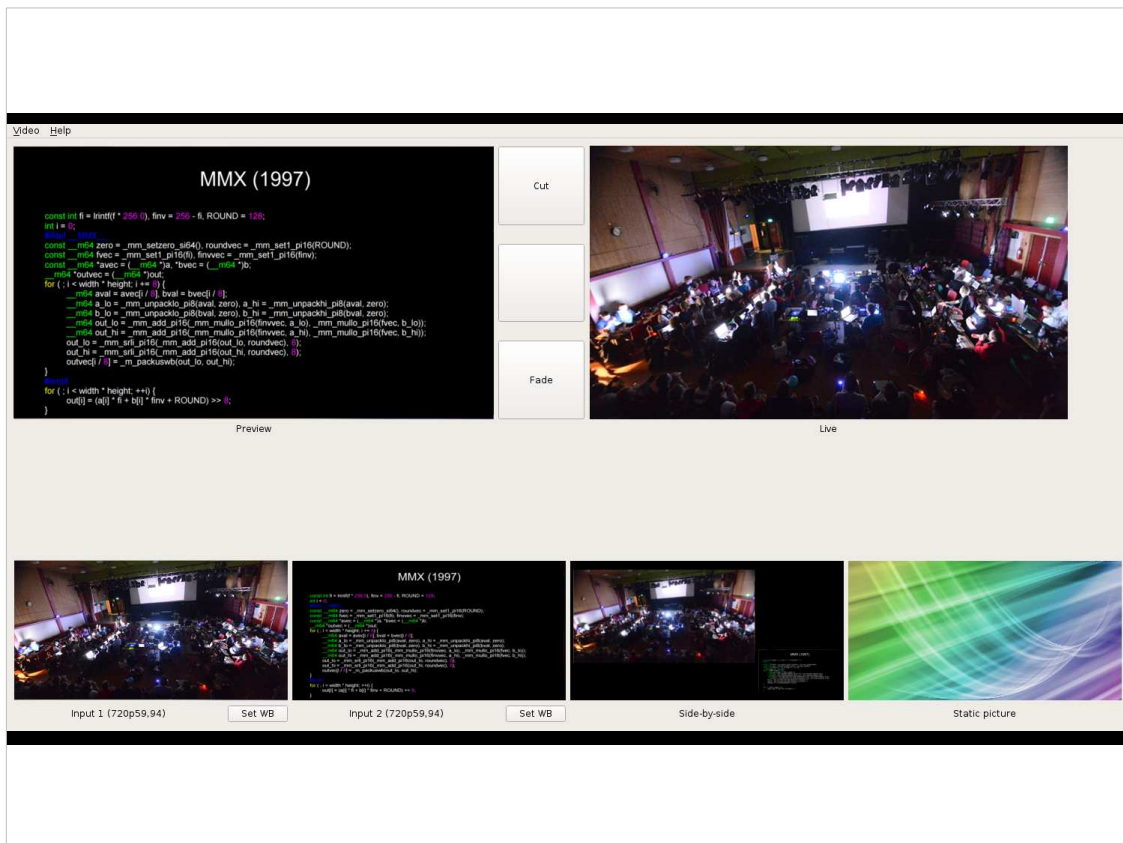
Movit contains most of the building blocks needed for high-quality pixel processing: Gamma-correct processing, nice scaling filters (none of that ugly bilinear or nearest-neighbor), white balance that actually turns the color you chose into white, not completely random handling of colorspace... that sort of stuff.

This is a 640x480 drawing scaled down to 64x48 using three different algorithms. You can guess which one Movit uses.



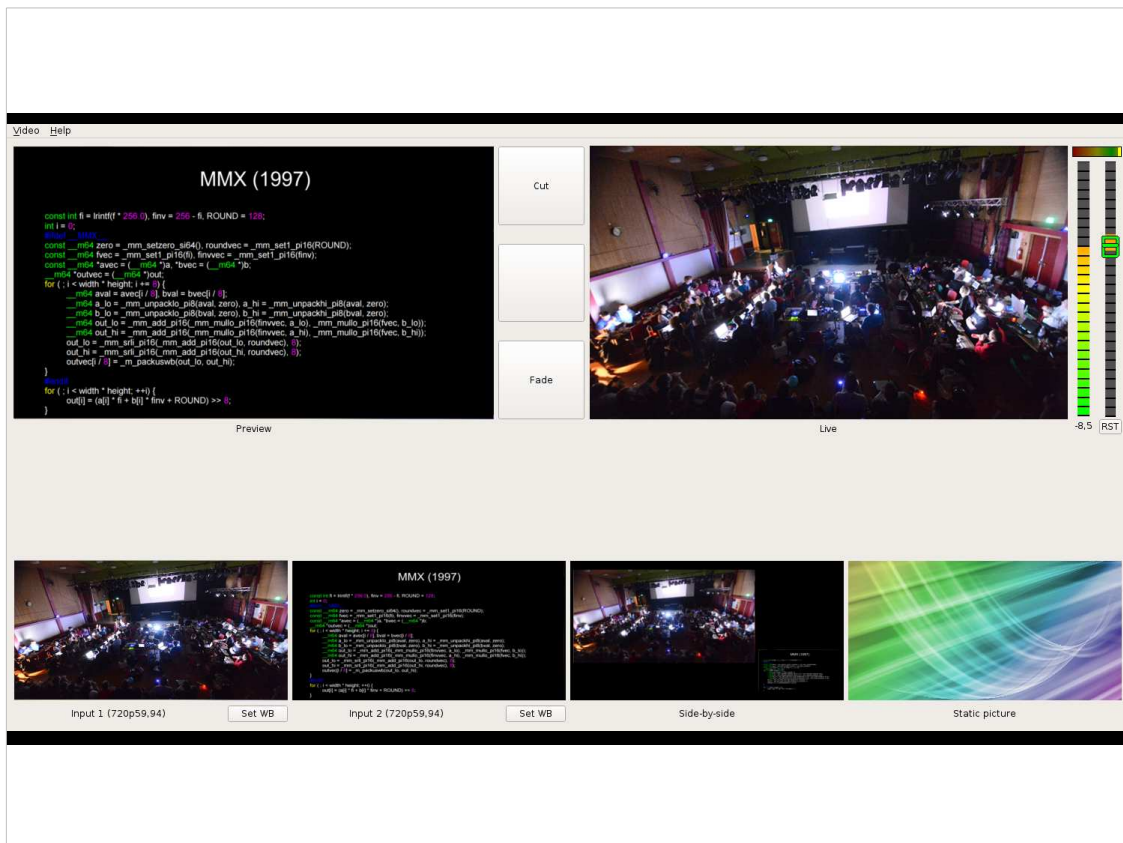
The visual look and feel of the production is governed by the *theme*, which is written in Lua and intended to be user-replacable. In a sense, this is Nageru's replacement for that fancy VizRT editor. You can see a small excerpt of it here (the included example is almost 800 lines of Lua code).

The theme's primary responsibility, for each and every frame, is to decide what Movit filter graph to use and what parameters to give to it. The one shown here comes from the middle of a fade, where one input needs to be scaled from another resolution to 720p. Movit decides which blocks can be fused together (marked with the same color) to increase speed, and compiles everything into a set of GLSL shaders that run on the GPU. Once the fade is done, probably a simpler filter graph will be used instead.



This is how it looks in the UI. Nageru uses a standard workflow which for whatever reason has been dubbed “mixer/effects” (M/E). The basic idea is that you first bring up something on the left (preview) display, then use some transition to get it onto the live one (right) as soon as you're happy. This minimizes the risk of putting the wrong thing on screen. The theme determines which transitions you can use between different sources, and handles really all the business logic.

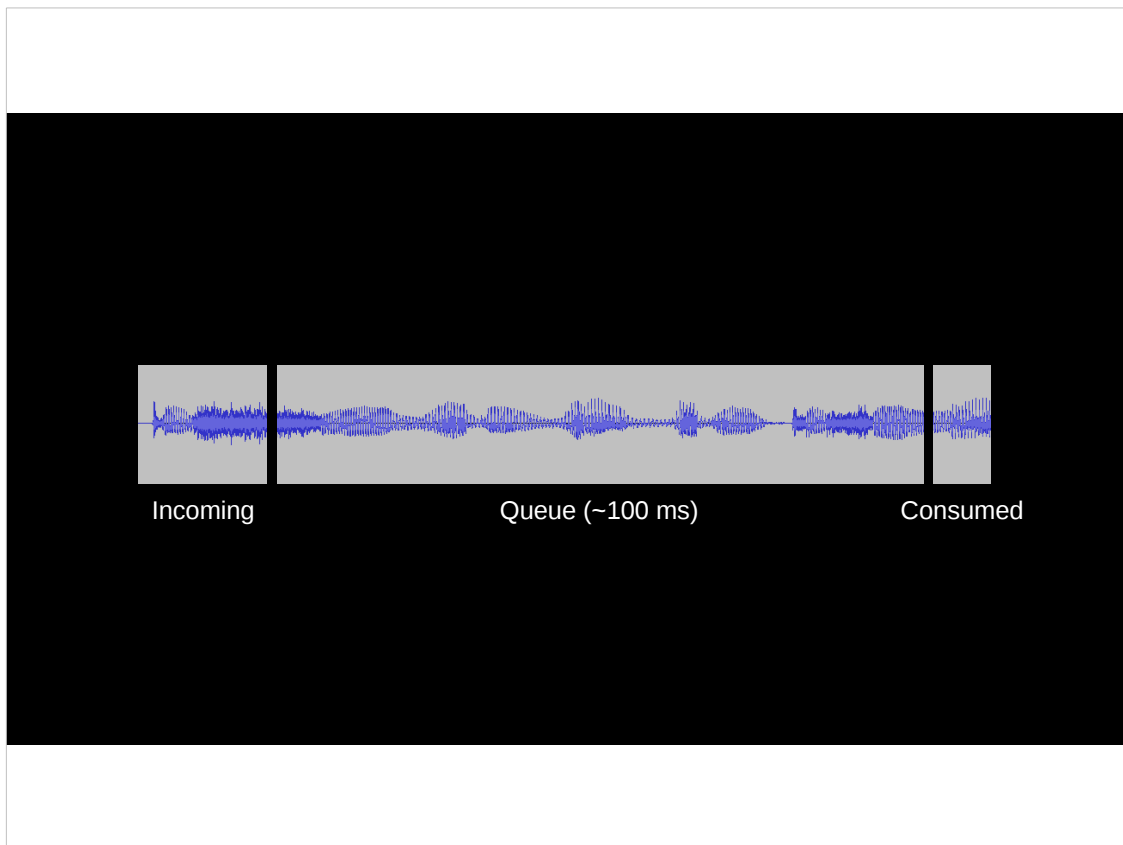
Each of these six displays here (the theme determines how many) is actually its own Movit chain which is rendered directly onto the screen; this eliminates expensive copies, and also let us render the previews in lower resolution/quality (in particular, cheaper resizing algorithms) to save on GPU power.



Everybody talks about video, but really, conference talks are all about the audio. Beyond the obvious cue-out, Nageru aims to give the operator real, usable tools to monitor and process audio, with a focus on speech. I listened to a lot of filibusters during the development on this part; it's great to have a test signal with someone babbling for hours.

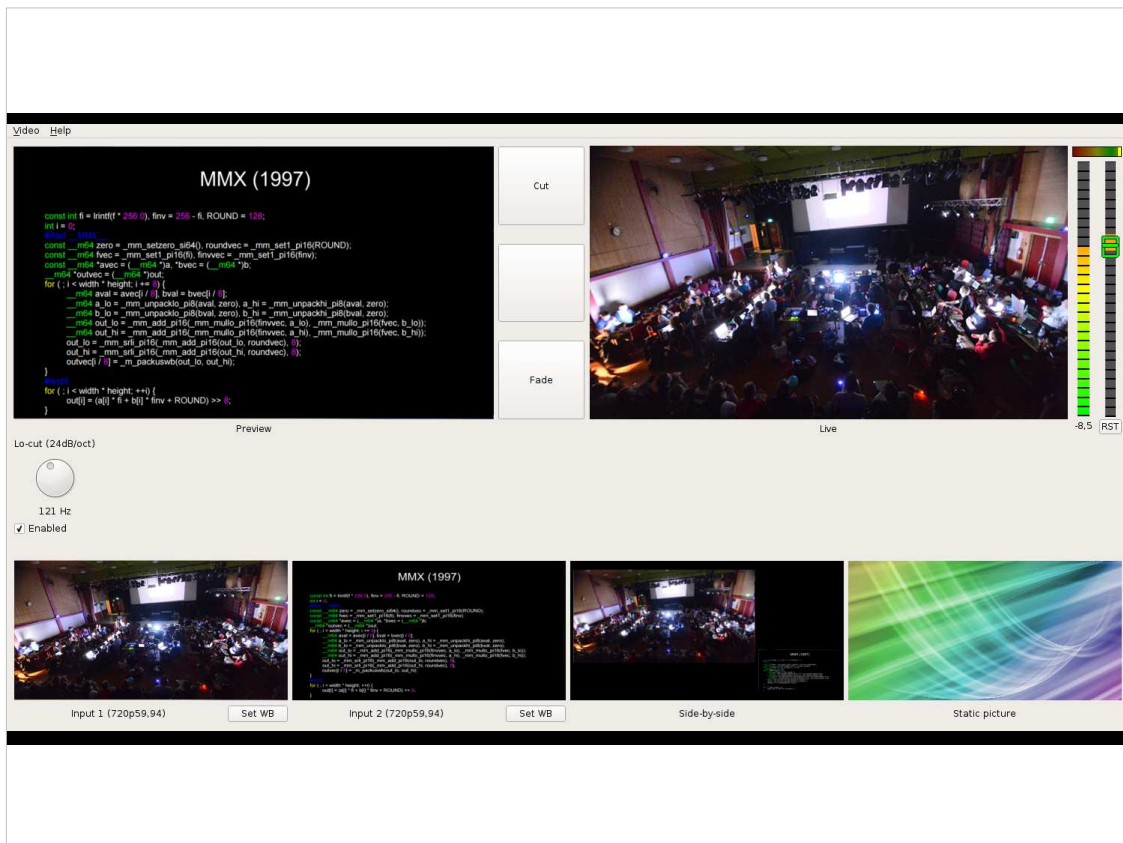
Level meters are easy to get wrong, but the EBU R128 standard tells us how to make ones that actually correspond to perceived loudness. To the left is the short-term level; to the right is the long-term range. If it stays within the little marked area (+/- 1 LU), you're at the right level, and you'll be at an acceptable loudness on most people's systems.

There's also a peak meter, and a left/right correlation meter, so that you can see you didn't get e.g. a dropout on one of the channels.



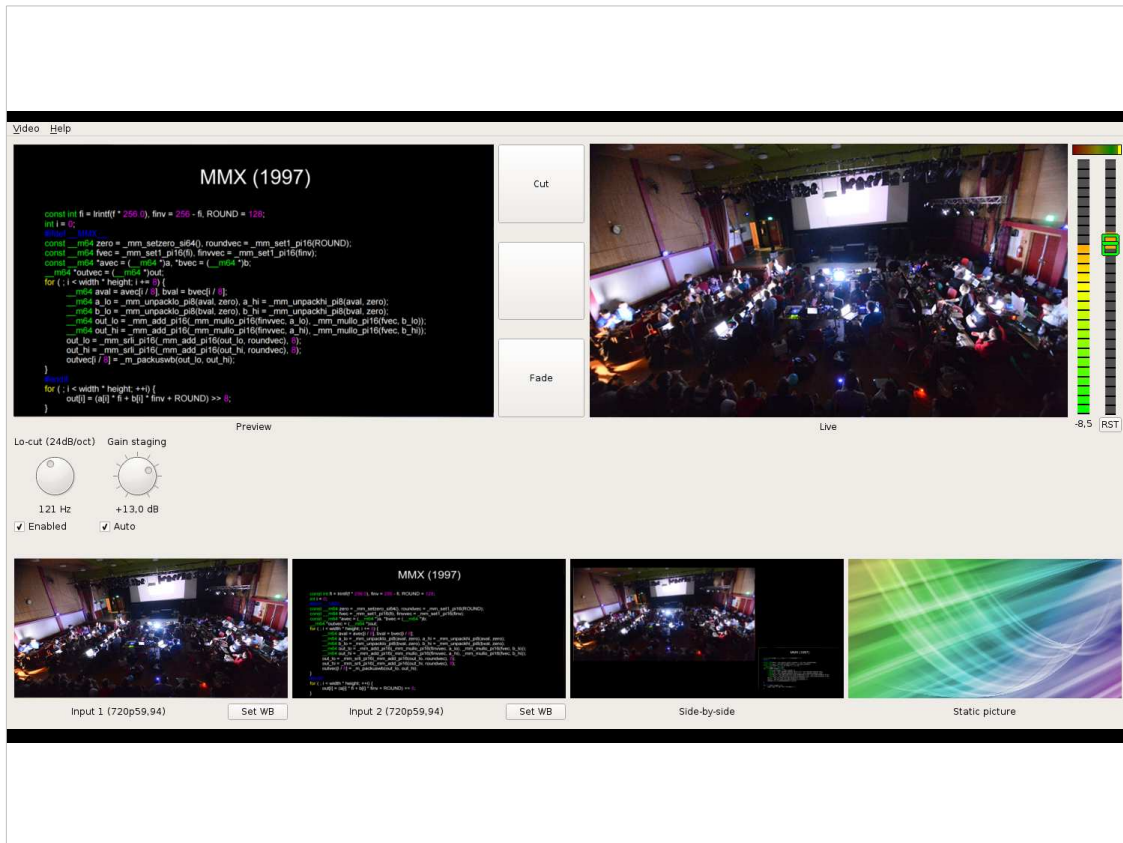
A/V sync is a much harder problem than people intuitively assume. Nageru chooses (pretty arbitrarily—might be configurable in the future) the first video card as the master clock, and every audio stream is adjusted to match that. We look at the rate of data coming into the queue and the rate of consuming it (dictated by the video clock). If they don't match, we stretch the audio ever so slightly. With a good control algorithm and a high-quality resampler (both due to Fons Adriansen) this is imperceptible.

The fact that we slave to the video clock means we rather implicitly get full VFR (variable frame rate) support. If you change the primary source from 60 to 50 fps, your stream will change, too. It mostly just works, even if you drop frames.



People have been doing audio for even longer than video, so the obvious thing to do when starting on Nageru's audio processing capability was to go and ask an audio engineer. He suggested a simple pipeline which I implemented more or less verbatim, based on simple existing components I already had lying around.

The first step in the pipeline is a highpass filter. The exact cutoff frequency is a bit of a matter of taste (and also depends on the speaker), but the main point is that it gets rid of low-frequency hum and a lot of the background noise that is not related to the speaker's voice.



Then, we have what's essentially a very slow compressor (500ms attack, 20 seconds release). This is set so that the overall sound level is about right; or actually, a bit over the correct point.

Video Help

MMX (1997)

```

const int fi = intff( * 256 / 0), friv = 256 - fi, ROUND = 128;
int i = 0;
// ...
const __m128 zero = _mm_setzero_si64(), roundvec = _mm_set_pi16(ROUND);
const __m128 fvec = _mm_set_pi16(friv), frivec = _mm_set_pi16(friv);
const __m128 fvec2 = (_mm512) fvec * fvec;
__m128 *outvec = (__m128 *)out;
for (i = width * height; i > 0; i--) {
    __m128 a_val = avect[ i ], b_val = bvec[ i ];
    __m128 a_lo = _mm_unpacklo_pi8(a_val, zero), a_hi = _mm_unpackhi_pi8(a_val, zero);
    __m128 b_lo = _mm_unpacklo_pi8(b_val, zero), b_hi = _mm_unpackhi_pi8(b_val, zero);
    __m128 out_lo = _mm_add_pi16(_mm_mullo_pi16(frivec, a_lo), _mm_mullo_pi16(fvec, b_lo));
    __m128 out_hi = _mm_add_pi16(_mm_mullo_pi16(frivec, a_hi), _mm_mullo_pi16(fvec, b_hi));
    out_lo = _mm_srli_pi16(_mm_add_pi16(out_lo, roundvec), 1);
    out_hi = _mm_srli_pi16(_mm_add_pi16(out_hi, roundvec), 1);
    outvec[ i ] = _mm_packsswb(out_lo, out_hi);
}
// ...
for (i = width * height; i > 0; i--) {
    out[i] = (a[i] * fi + b[i] * friv + ROUND) >> 8;
}

```

Live

Preview

Lo-cut (24dB/oct)

121 Hz

 Enabled

Gain staging

+13.0 dB

 Auto

Compr. threshold

-26.0 dB

 Enabled

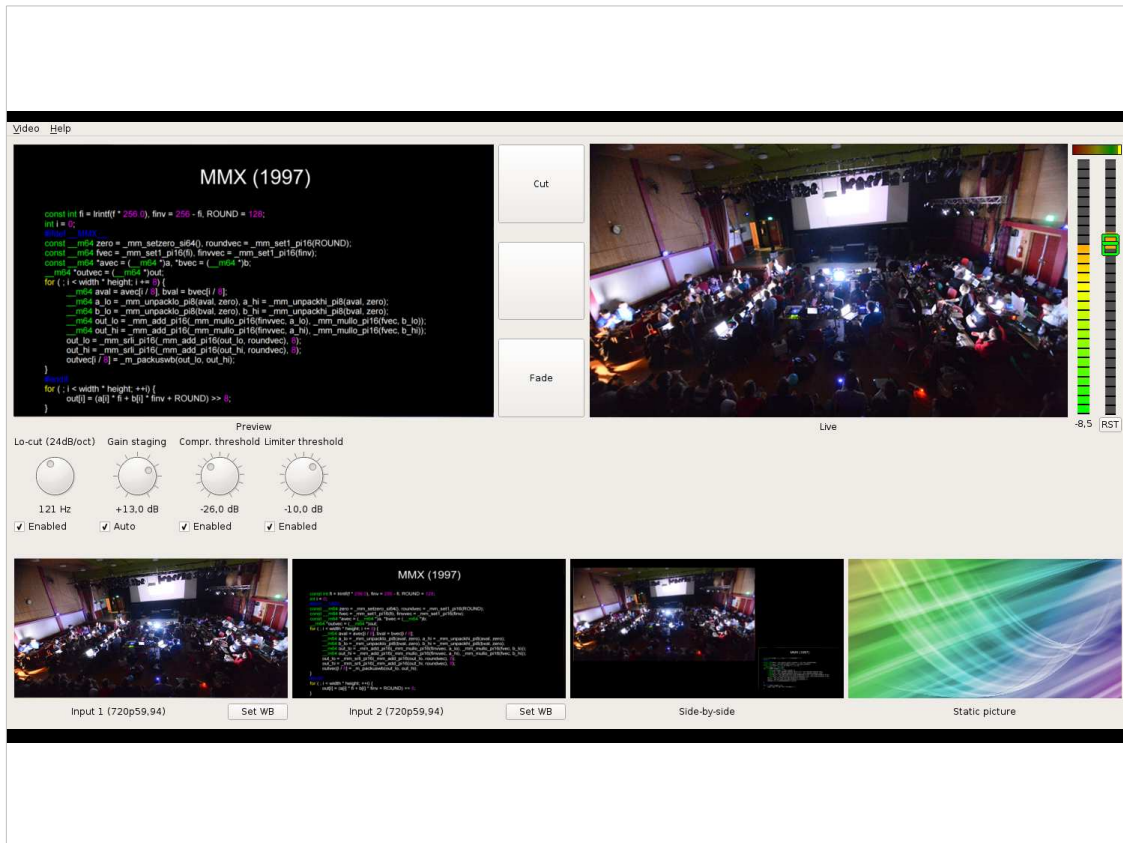
Input 1 (720p59,94) Set WB

Input 2 (720p59,94) Set WB

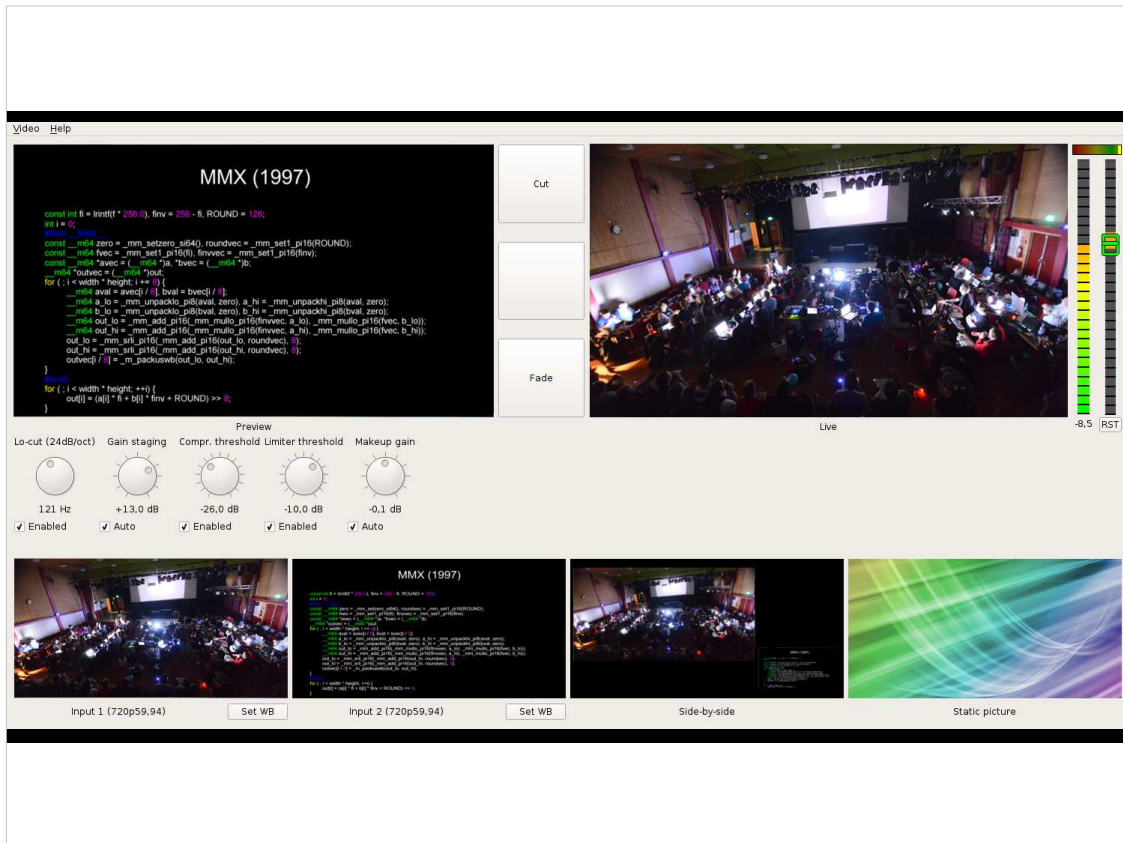
Side-by-side

Static picture

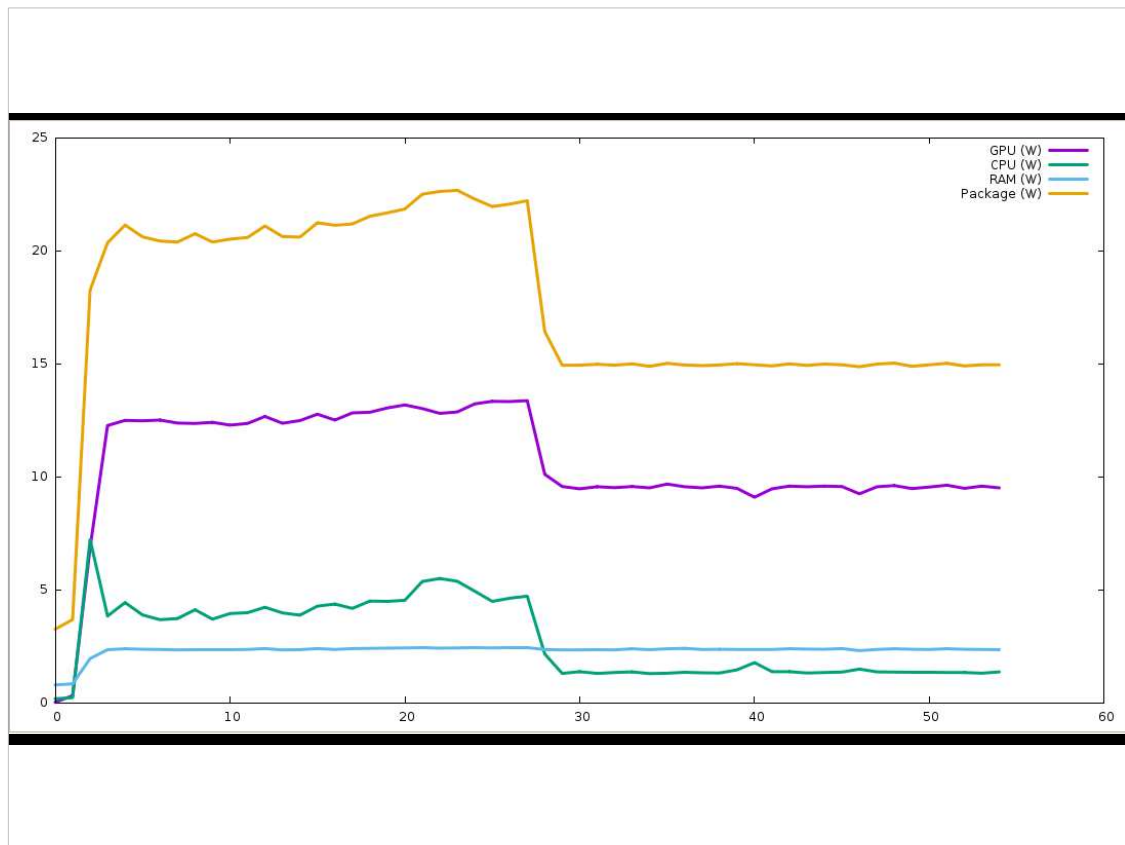
Now comes the real compressor, with typical voice settings (5 ms attack, 40 ms release). It has the effect of making the voice sound a bit tighter, more level and just overall better.



The third and final compressor in the chain is a limiter, basically a compressor with very high ratio. It's there as an emergency brake for really loud transients that got through the other compressors. Have you ever had your ears blown out by a presenter suddenly coughing into the mic? That is why you want a limiter.




All the compressors have been working on objective values, but we want the final level to be around 0 LU on the R128 scale. So we have a final makeup gain that continuously reads the R128 meters and tries to adjust the gain very slowly to end up in the right place.



So now we have lots of processing, hammering both the CPU and the GPU. What hardware do we need?

My target platform all along was my own ThinkPad X240. Its 2x2.10 GHz Haswell sounds attractive enough... until you look at this power graph. Like all modern laptops, it can only run at full power for 30 seconds or so, after which it clocks down to... 2x800 MHz. Ouch. We can't even get those +17% per year, because that's for desktop/server chips.

Even worse, the promised 25.6 GB/sec GPU bandwidth (which is *nothing* compared to a modern desktop card, which can do 300+ GB/sec) is halved due to having only one memory channel. And 12.8 GB/sec is half-duplex, shared with the CPU, and besides, a rather theoretical figure. It's actually very tight.

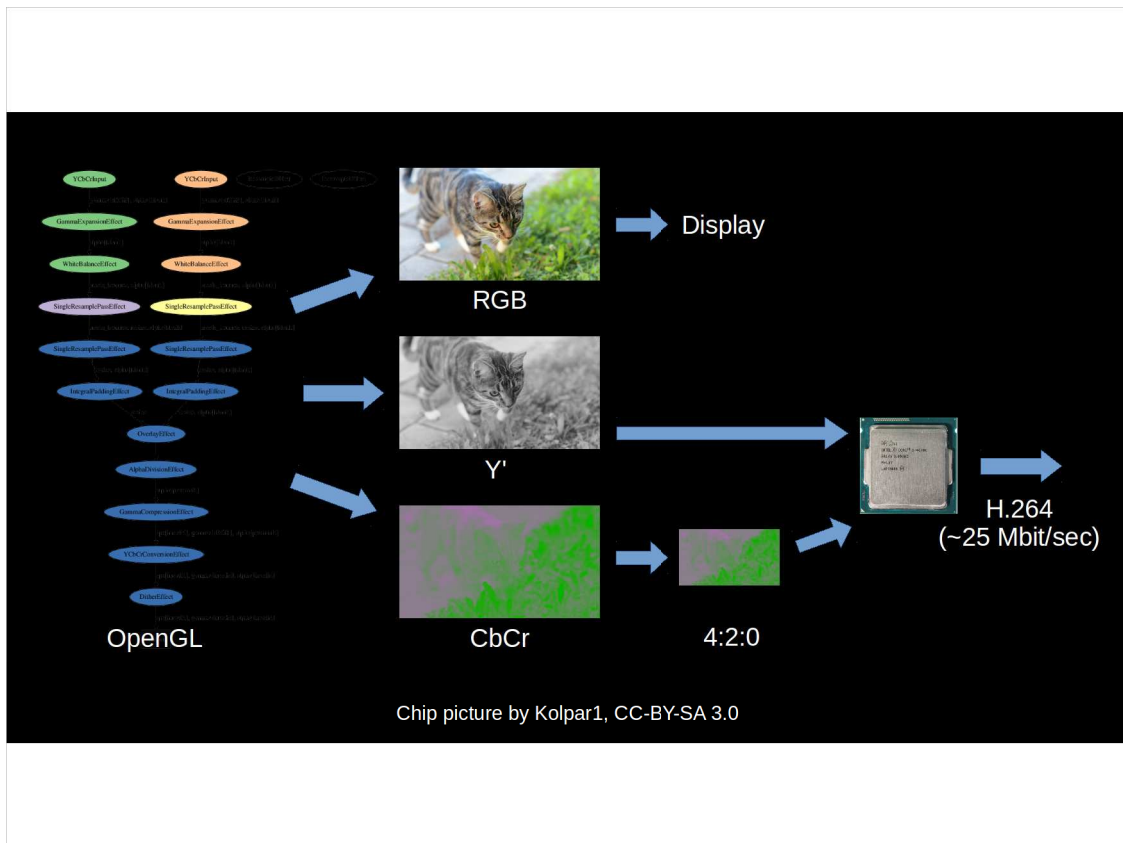


From: Steinar H. Gunderson
Subject: [PATCH] Add SSE2 support to zita-resampler

So there were a lot of optimizations done. This is one patch where I made the resampler faster, but a lot was done within Movit and Nageru itself. In particular, collapsing more passes helps save on bandwidth; in practice, we have bandwidth for only ~20 reads or writes per pixel.

Note that doing things on the GPU doesn't completely free up the CPU either. There's a fair amount of driver and kernel overhead; using modern OpenGL features such as persistent mappings helps, but there's still a lot to do.

The CPU still touches the data in one place; it parses the USB packets and copies it over into GPU memory. Mesa 11.2 adds compute shader support; it might be that it's possible to move the parsing onto the GPU, but the main cost here is actually not the parsing, it's reading the data into the L3 cache.



There's one detail we haven't talked about here yet, namely output. Uncompressed 720p60 video (4:2:0) is almost 700 Mbit/sec, or 300 GB/hour. This is possible to stream on a LAN, but very inconvenient to store. And we're actually quite out of CPU, so any codec we employed would have to be cheap.

Thankfully, modern Intel CPUs have a feature called Quick Sync Video. It's essentially a hardware H.264 encoder (the newest CPUs do VP8, too). It's not nearly as good as x264, so we use it only to get down to about 25 Mbit/sec or so; enough for storing a reference copy, and for transporting over to some other place for transcoding into whatever you might want. Beautifully enough, EGL allows the GPU to render straight into VA-API's buffers for Quick Sync to encode out of. Zero copies needed!



Live demo!

Now I will anger the demo gods by attempting easily the most involved live demo I've ever done.

Assembling two cards, a multi-camera rig (including one borrowed SDI camera I've never seen or used before FOSDEM), and HDMI and SDI cabling on stage in the course of two minutes will be fun. Of course, again there's no such thing in the PDF, but hopefully, you can watch the video if you want to see Nageru in action.

Thank you!

(Q&A if people have not run out of patience yet)

<https://nageru.sesse.net/>

So that's it! With so many slides, I'm not sure I actually get to answering any questions at all on stage, but I'll be happy to chat at any point during FOSDEM. There are a lot of people that deserve thanks, but in the interest of time, I will refrain from naming specific ones. You know who you are.

If you want to ask questions online, of course you're most welcome. I'm on IRC as Sesse, and my email address is at the Nageru home page (which will open up immediately after the talk).