Extreme compression of Video with VP9 and AV1 (webm) using ffmpeg

Dr. Arne Babenhauserheide

<2020-12-06 So>

Emacs Conf 2020 just started creating optimized videos of their recordings, so I gathered the recommendations I worked out over the past years to compress video with ffmpeg.

Contents

The typical approach is to provide low-resolution video for those with slower connections. As an alternative that I prefer, it is possible to use extreme compression settings in vp9 to get high-resolution video with low bandwidth.

That will take more encoding time, so it's a mid-term solution, but it can reduce bandwidth a lot.

Parameters for VP9

Here's my advice for ffmpeg-parameters that work pretty well. Adjust `EXT` to the extension of the filetype to transcode, and adjust `Q` for different quality settings (56 is easily enough for talks, and text works pretty well, for high-movement scenes you might need to go down to 42, but do test first!).

These parameters were worked out by people in the Freenet Project where bandwidth is even more scarce than with self-hosting in the clearnet or on mobile platforms.

```
Q=56; EXT="webm"
```

```
time for i in *.${EXT}; do
    nice ffmpeg -y -i "$i" -c:v libvpx-vp9 -b:v 0 -crf $Q \
 -aq-mode 2 -an \
 -tile-columns 0 -tile-rows 0 \
 -frame-parallel 0 -cpu-used 8 \
 -auto-alt-ref 1 -lag-in-frames 25 -g 999 \
 -pass 1 -f webm -threads 8 \
 "$(basename "$i" .${EXT})".temp;
    nice ffmpeg -y -i "$i" -c:v libvpx-vp9 -b:v 0 -crf $Q \
 -aq-mode 2 -c:a libopus -b:a 12k \
 -tile-columns 2 -tile-rows 2 \
 -frame-parallel 0 -cpu-used -5 \
 -auto-alt-ref 1 -lag-in-frames 25 \
 -pass 2 -g 999 -threads 8 \
 "$(basename "$i" .${EXT})"-vp9-q${Q}.webm;
done
```

I've seen high-quality baked-in subtitles of Anime stay crisp at crf higher than 42. For fixed video files (not live streaming) I would always go for the crf-approach, since it produces fewer glitches and still provides reasonable constraints on bandwidth.

To make it easier for you to check the different qualities, I uploaded an experiment I did with different encoding qualities of New Horizons for Science. This is from 2017 and AFAIK encoder quality got a bit better since then, but it should still give you a good starting point to choose the crf (the parameter `Q` in my code-snippet).

Quality	Filesize	Encoded video
Q=24	87M	new-horizons-for-science-vp9_24.webm
Q=36	27M	new-horizons-for-science-vp9_36.webm
Q=42	16M	new-horizons-for-science-vp9_42.webm
Q=46	12M	new-horizons-for-science-vp9_46.webm
Q=48	9.3M	new-horizons-for-science-vp9_48.webm
Q=50	8.0M	new-horizons-for-science-vp9_50.webm
Q=52	7.0M	new-horizons-for-science-vp9_52.webm
Q=56	5.1M	new-horizons-for-science-vp9_56.webm
Q=63	2.1M	new-horizons-for-science-vp9_63.webm

For the video above I decided to go with crf=56 after this test, because 5MiB can be sent in an email without feeling big, and because it makes me

smile sheepishly to send around a high-resolution video with a bitrate of an audio-file from when I started into p2p networks :-).

Note that if you go with CRF below 56, you might want to adjust the opus bandwidth (-b:a 12k) to 16k. Going to the extreme limit of opus for complex audio is only worth it if you also go to the limits of the video, otherwise the bitrate savings of going to the limit of the audio-stream (with Opus which is already extremely efficient in normal operation) just isn't relevant.

Things which save a lot of bandwith here:

-cpu-used -5 in pass2: This slows down encoding A LOT, but it is the highest value I found that still produces sufficient quality. You can go down to -8, but the added quality isn't as spectacular as the added encoding time :-) (there were benchmarks that show the cutoff at -5, but I don't have them at hand) -cpu-used 8 in pass1: This makes the first pass fast and does not impact quality. Remember to put -5 for the second pass.

-g 999: This basically increases the time between keyframes to about 30s. This impacts fast-forward and jumping to specific timecodes, so you might want to use a lower value. -g 300 still gives a keyframe every 10s, so it should be less noticeable.

-auto-alt-ref 1 -lag-in-frames 25: This allows encoding by taking data of "future" frames. It substantially improves text-overlays

More info on vp9-parameters is available froom the webmproject (specifics) `https://www.webmproject.org/docs/encoder-parameters/` and from ffmpeg `http://ffmpeg.org/ffmpeg.html#libvpx`

Streaming

To stream stuff, see Volker Tanger's Live Streaming HowTo (for the clearnet) or How to stream into Freenet for experimental decentralized, anonymous streaming.

Parameters for AV1

You might asks why I don't suggest av1 for Emacs Conf. I'm actively experimenting with that, and using **-cpu-used 1** with all tricks can almost halve the bitrate compared to the vp9 call above and still get better quality, but at the cost of taking 3 days to encode a mere 3 minutes. That's on a not

too low-specced system (almost 10 CPU-days at 4 Ghz). To re-compress the whole conference, this is currently far too expensive. Values for cpu-used above one actually give worse results than the vp9 call. With av1 I see speeds below 0.0008x — every second takes 20 minutes to encode, one minute takes roughly one day to encode.

That said, this is an ffmpeg-call with optimized AV1-parameters for minimum filesize (no guarantees for perfection: most of these parameters are experimental — following the descriptions in the docs — and I could not yet test the changes due to each parameter):

```
Q=56 && EXT="mkv" && time for i in  *.${EXT}; do
    nice ffmpeg -y -i "$i" -c:v libaom-av1 -strict -2 \
 -b:v 0 -crf $Q \
 -aq-mode 2 -an \
 -sc_threshold 0 \
 -row-mt 1  -tile-columns 2 -tile-rows 2 -threads 12  \
 -cpu-used 8 \
 -auto-alt-ref 1 -lag-in-frames 25 -g 999 \
 -pass 1 -f webm \
 "$(basename "$i" .${EXT})"-av1.temp
    nice ffmpeg -y -i "$i" -c:v libaom-av1 -strict -2 \
 -b:v 0 -crf $Q -aq-mode 2  \
 -sc_threshold 0 \
 -row-mt 1  -tile-columns 2 -tile-rows 2 -threads 8 \
 -cpu-used 1 \
 -auto-alt-ref 1 -lag-in-frames 25 -g 999 \
 -c:a libopus -b:a 12k \
 -pass 2 -threads 12 \
 "$(basename "$i" .${EXT})"-av1-q${Q}-cpu-used-1.webm
    # create a preview image to use as poster; position: 5s (-ss 5)
    mplayer "$i" -ss 5 -nosound -vo jpeg:outdir=. -frames 1
    mv 00000001.jpg "$(basename "$i" .${EXT})"-av1.jpg
done
```

This requires around 40h of encoding time for the 2.5 minutes of video, but the result is stunning:

Quality	Filesize	Encoded video
Q=56	3.3M	new-horizons-for-science-av1_56.webm
Q=58	2.9M	new-horizons-for-science-av1_58.webm
Q=60	2.4M	new-horizons-for-science-av1_60.webm
Q=62	1.9M	new-horizons-for-science-av1_62.webm
Q=63	1.7M	new-horizons-for-science-av1_63.webm

The quality 60 version is down to 1MiB per minute — 128kbit/s for HD video, and if you don't know where to look there is almost no visible breakage. If you don't see how awesome that is, you did not grow up with a computer in the filesharing age :-)

The quality 63 version even gets down to 87.5kbits/s. Watch the video to judge the quality yourself.

With these codecs we could do swarming HD-video streaming on an ancient double-ISDN line. A private DSL connection with 40Mbit/s upstream would suffice to stream pre-converted low-movement HD-video to 400 fans. This is the silent revolution of media compression algorithms whose potential has mostly gone un-noticed by the public.

At 40MiB per hour, all the recordings from the main track of a three day conference would fit on a simple DVD or an old 4GB USB stick (the cheapest you get). A plain CD could hold 20 hours of standup comedy, and the typical 32GB USB stick would hold 800 hours of low-movement video — that's more than a full month non-stop video.

From a few tests I did, high-movement video is about 5x as big, but that still leaves us with almost one week of video on a plain 32GiB USB stick — which is cheaper to buy than a typical monthly streaming subscription.

And you can send a Music Video by E-Mail.

Despite being purely digital, the encoding time vs. bandwidth clearly makes this an economy of scale. Do you still wonder why streaming platforms are currently shooting up like mushrooms? However this would also scale perfectly with decentralization: One gaming-PC would suffice to encode the creative output of a professional youtuber with high production-quality like CGP Grey or maiLab (10-30 minutes per week). Looking only at electricity, that's around 50€ per month. Investing that would allow to self-compress the video enough that it could be sent by email, avoiding the need for centralized infrastructure.