FFMPEG Tricks

Better screen capture with kmsgrab

Those of you that have tried to use x11grab are probably aware that it is terrible at high resolutions. To be fair, it's not really FFMPEGs fault either to capture an aging X11 window (yes I remain hopeful for Wayland). The user experience isn't great though, and not only does it drop frames, it tends to lag the entire X server causing everything on screen to stutter as well. By the way, this isn't limited to FFMPEG, full XSHM screen capture on OBS has the same issue (and probably uses similar underlying logic.

There are a few alternatives:

- However Xcomposite works on OBS doesn't seem to be prone to this issue
- OpenGL capture also seems to work fine at higher resolutions

Anyways, here, I'm going to describe kmsgrab. Its another capture source for ffmpeg that works much better then x11grab (with some limitations). It still drops some frames but overall seems better then x11grab. It is quirky sometimes though and isn't as flexible. Also, it doesn't capture the cursor (not sure why tbh, I guess something about compositing?)

Information on the input is available here: https://ffmpeg.org/ffmpeg-devices.html#kmsgrab

To use it, you first need to enable CAP_SYS_ADMIN as follows

```
sudo setcap cap_sys_admin+ep /path/to/ffmpeg
```

Credit to http://lists.ffmpeg.org/pipermail/ffmpeg-user/2018-June/040328.html that helped me figure that part out.

Then, to capture, you can use the following

```
ffmpeg -device /dev/dri/card0 -f kmsgrab -i - -vf
'hwmap=derive_device=vaapi,scale_vaapi=w=1920:h=1080:format=nv12' -c:v
h264_vaapi -b:v 2500k -maxrate 4000k -y output-file.mkv
```

This also captures with VAAPI to use hardware encoding. You need vaapi setup to use it, otherwise, use libx264. See https://wiki.archlinux.org/title/Hardware_video_acceleration and https://trac.ffmpeg.org/wiki/Hardware/VAAPI on using VAAPI.

Adding -b:v and -maxrate will use VBR (in theory). Use the same value for both to get CBR instead (useful if you're going to stream straight to Twitch/Owncast). VAAPI also supports -qp for constant quantization parameter, but not CRF which would be superior since it can do more analysis to set the qp. Also, don't forget to add audio capture with ALSA/Pulse/Jack.

Note, you may need to change /dev/dri/card0 to a different DRI if you have 2 GPUs (eg: integrated and

dedicated). I also found that kmsgrab didn't work on my desktop with an RX 480. No idea why.

As always, I recommend never recording to mp4 files (or anything else w/ metadata at the end) since you will corrupt the file if the recording crashes. Use mkv (or ts) instead.

Convert still image to a video

You can convert a still picture into a video of your specified length. You just need to loop the input, and provide -t to set the time.

In the below, change the -t for how long the resulting video should be and mark -loop 1 for the input. All other parameters are placeholders and don't really matter.

I found it helpful to specify pix_fmt since otherwise it tends to use yuv444p (which is good for better colours) but fails with some codecs that don't support it.

```
ffmpeg -loop 1 -i input.png -c:v libx264 -crf 18 -r 30 -t 10 -pix_fmt yuv420p
output.mkv
```

Scale video using black bars

By default FFMPEG scales video to a given resolution but keeps the original aspect ratio, so you can end up with a 1920×1080 video that plays in 4:3. This may not be what you want.

There are a few solutions. First, you could not set the width or height with -1, and this way, FFMPEG will scale it keeping the same aspect ratio (eg: -vf scale=1920:-1) and match the missing width/height for what it needs to be for that aspect ratio. This is the recommended way.

On the other hand, you can force it to be the specified resolution, and stretch/compress the video.

You could also crop in the video to match the specified resolution and aspect ratio.

Finally, you can resize it like the first solution and add black bars to fill your desired resolution. This is useful if you are going to concatenate videos for example to edit something together, and want the size/aspect ratio consistent. This way, if you have a few 4:3 clips is a majority 16:9 video, all will be well.

The following does this, where RW is the width, and RH is the height you want. This will resize the video to fit in it's original version and then pad it. You can change the colour of the padding.

```
ffmpeg -i input -c:v libvpx-vp9 -crf 30 -vf
"scale=${RW}:${RH}:force_original_aspect_ratio=decrease,pad=${RW}:${RH}:-1:-1:
color=black" output.webm
```

Add countdown timer to video

I'm not sure who has a use for it, but I've used it to act as a countdown in a slideshow video.

To do this, first, you need to get the length of the video, then draw the text onto the video frame.

I recommend using this in a shell or bash script. First, find the duration as follows, where $f is the input video filename, and duration is the output duration variable.

```
duration=$(ffprobe -loglevel error -show_entries format=duration -of
default=nw=1:nk=1 "$f");
```

Next, you can use ffmpeg to add the countdown text

```
ffmpeg -i "$f" -c:v libtheora -q:v 7 -vf
"drawtext=fontfile=/usr/share/fonts/truetype/dejavu/DejaVuSansMono-
Bold.ttf:text='%{eif\:$duration-
t\:d}':fontcolor=yellow:fontsize=24:x=50:y=50:box=1:boxcolor=black@0.5:boxbord
erw=10,format=yuv444p" output.ogv
```

Here, you need to provide the path to the font you want to use (I recommend something monospaced so the timer stays still).

Then, the text parameter takes the duration variable from earlier, and subtracts the current time of the video in seconds. This results in a countdown timer of how many seconds are left. Font color is the colour, and fontsize is the size of text. The x and y parameters specify where on screen it should appear. You can use w or h to represent the width and height of the video, so 50,50 is top left, w-tw-50,50 is top right, so on. This also adds a box behind the text with a 10px border, with a 50% black opacity. Makes it easier to read, but you can mess with it or remove it entirely.

As always, you can daisy chain video filters with , if you want multiple text boxes, or resize and countdown, etc.

Extract closed captions to subtitle file

Some MP4 and ts files have closed captions embedded in the old ATSC standard. Some players can't handle embedded closed captions, and it's more convenient to have it as a separate file.

You can extract the CC streams and save it as a separate file as follows:

```
ffmpeg -f lavfi -i "movie='input'[out0+subcc]" -map s "output.ssa";
```

where input and output are the filenames to use. Subrip SRT files can be used for the output instead of SubStationAlpha SSA files. SSA seems to actually decode the proper colours and location of the closed captions better, and maintains that information although it is less compatible with some players.

You can easily make this loop across a folder with a for loop.

```
for i in *.mp4;
 do name=`echo $i | cut -d'.' -f1`;
 ffmpeg -f lavfi -i "movie='$i'[out0+subcc]" -map s "${name}.ssa";
done
```

Interpolate video to higher framerate

Ffmpeg has a filter called 'minterpolate' that can interpolate videos to a higher framerate. You get get not-terrible results by playing with the parameters, it's better then the 60 or 120hz interpolation done by TV's that just fades/blends the frames.

Here is a command that will take the input, interpolate it to 60 fps, and encode it with libx264 at crf 18 (visually lossless)

```
 ffmpeg -i "input.mkv" -c:a copy -c:v libx264 -preset medium -crf 18 -vf
"minterpolate=fps=60:mi_mode=mci:mc_mode=aobmc:me_mode=bidir:vsbmc=1"
"output.mkv"
```

Interpolation is very single threaded, and very slow. Most modern CPUs have multiple cores so it would benefit performance to run multiple threads. We can do this by splitting the video into 20 or 30 second chunks (depending on video length) and running multiple instances of ffmpeg in parallel.

This can be done using my video splitting method listed below, then using the following GNU parallel command to run 16 instances of FFMPEG (for my 16 core processors - be careful since it uses a lot of RAM).

```
 parallel -j 16 -q -a files ffmpeg -i {} -c:a copy -c:v libx264 -preset medium
-crf 18 -vf
"minterpolate=fps=60:mi_mode=mci:mc_mode=aobmc:me_mode=bidir:vsbmc=1" -
max_muxing_queue_size 1024 -y "{.}.done.mkv"
```

where -j is the number of threads to run, and the “files” file contains a list of the parts, one file per line. You can generate the list of files by running a find command

```
find ./ -name '*.mkv' >> files
```

Finally, you can merge all the files to merge using ffmpeg. (see below)

Split a video file to multiple parts

If you need to either upload a video in multiple parts, or run a command in parallel (see above), it can be useful to split a video of audio file.

This can be done with ffmpeg using the 'segment' feature.

```
ffmpeg -i input.mkv -f segment -segment_time 30 -c:v copy -c:a copy -
reset_timestamps 1 -map 0 output-%2d.mkv
```

This will losslessly split the video into 30 second chunks. Since we are just copying the audio and video without re-encoding, we don't need to worry about reducing the quality, or gapless playback when you merge back. (You really shouldn't split and re-encode audio, it will be noticable with noise or pop when going between segments when you merge it back).

In this command, -segment_time is the length to split at in seconds. -reset_timestamps 1 is needed so that the resulting video starts at 0:00 with the proper timecode information.

The main benefit of using segment instead of splitting directly, is that segment is smart when deciding where to split. Even it you request chunks at 30 seconds, the actual chunk lengths will vary. This is because it will make sure that the split falls on a keyframe, so there is no corruption when you merge and play the video back.

Merge multiple video/audio files into one

This is useful when you need to combine multiple videos into one, either from parts above, or more likely, to make a pseudo-playlist or slideshow of videos. This method also works for audio files such as if you're going to burn a continous track on a CD, or combine interviews for a podcast or whatever.

First, get the list of files in a text file. It needs to be in the format of “file filename” in each line.

```
find *.mkv | sed 's:\ :\\\ :g'| sed 's/^/file /' > files
```

Here is a command that will make a text file called files of all mkv files in a folder. You can do this for any extension, and the sed command will append the file structure needed.

Next, run ffmpeg to merge the video or audio segments.

```
ffmpeg -f concat -safe 0 -i files -c copy output.mkv
```

This will take in all the files in the list generated above, copy the codecs without re-encoding into the output.

You may or may not want to re-encode when merging. If your inputs have different codecs, re-encode the input. If you're merging from parts you generated by splitting the video manually (see above example), then you can just copy the codecs.

Compress all FLAC in subfolders to OPUS

Compress all FLAC audio files in subfolders to OPUS at 128 Kbit/s (near-transparent), while maintaining the metadata tags and album art.

```
find ./ -name '*.flac' -exec bash -c 'ffmpeg -i "$0" -f flac - | opusenc - --
bitrate 128 --vbr --comp 10  "${0/flac/opus}" && rm -v "$0"' {} \;
```

We use find to select all the flac files, and run the bash files. The first runs ffmpeg and pipes the output to opusenc. The reason is if you use ffmpeg to save to opus files, the album art is lost, whereas when piped in to opusenc, it is encoded properly.

Visualize music into notes

Sometimes I get curious and want to break down a song to it's core melodies, FFMPEG has a cqt filter (like fft), that can convert an audio file into a visual based on the notes. I also realized you can do this in real-time with decent latency using ALSA.

```
ffmpeg -f alsa -ac 2 -i pulse -filter_complex
"[0:a]showcqt=s=1280x720:fps=30:[out]" -map "[out]" -f sdl -
```

- This will try capturing a microphone by default, use 'pavucontrol' to change the captured input to loopback for your speaker.
- I tried to use '-f pulseaudio' instead, but it just ends up being very stuttery and with much higher latency, whereas the alsa version is so much smoother and lower latency.
- You can change the resolution and fps are needed, I sometimes daisy-chain multiple effects.
- Output as SDL has much lower latency from my testing. I used to pipe it into mpv with no buffer, but this is even faster

Older revision of this command I used to use as reference:

```
ffmpeg -f pulse -ac 2 -i default -filter_complex
"[0:a]showcqt=s=1280x720:fps=30:[out]" -map "[out]" -r 30 -c:v rawvideo -f
matroska - | mpv --no-cache --untimed --no-demuxer-thread -
```

This stutters a bit more, but might be preferred in some cases. It also uses pulseaudio directly. I switched to using rawvideo instead of other codecs, since rawvideo has a much lower latency, as there is no need for compression and decompression, which doesn't matter since we're piping it for playback anyway.

Stream desktop to Icecast

If you need to stream to many computers at once, Icecast is pretty useful. FFMPEG can directly stream to an Icecast server (alternatively, you can use similar settings in OBS). I capture the x11 window, then

encode using VAAPI for reduced CPU usage.

```
ffmpeg -vaapi_device /dev/dri/renderD128 -video_size 1920x1080 -framerate 30 -
f x11grab -i :0.0+0,0  -vf 'hwupload,scale_vaapi=w=1920:h=1080:format=nv12' -
c:v h264_vaapi -f mpegts -content_type 'video/mp2t'
"icecast://source:password@example.com:8000/name.ts"
```

- The coordinates after the -i is width, height, left offset, top offset.
- The video filters are to scale to 1080p, while maintaining the vaapi encoder.
- I use mpegts to stream the video, though other file formats that allow streaming such as matroska also work. If you're streaming theora/vorbis, use ogg, for audio, use opus if you're encoding opus, so on.
- I set the content type to video/mp2t, which makes the video stream properly detect. It also makes the m3u file download rather then play in the browser.

Force a fixed frame rate

Some devices seem to record both the screen and camera with variable frame rates. *COUGH* APPLE *COUGH*.

FFmpeg handles some frame rates differently. If you do -r, it depends on container. For MP4, it WILL make it a fixed rate by dropping or duplicating. For MKV, it won't. It will ONLY DROP frames if the input frame rate is above the specified one. If the output is higher it WON'T duplicate frames.

For force a frame rate with dropping/duplication, use

```
-vf fps=fps=ntsc
```

eg:

```
parallel --ungroup -j 4 -q -a files ffmpeg -i "{}" -c:v libx264 -crf 26 -vf
fps=fps=ntsc  -c:a libvorbis -q:a 4 -ac 1 -y -threads 4 "{.}.mkv"
```

where the ntsc can be replaced with a fraction. The parallel bit is just to run this on every file in the text file files. Good for mass-converting. 4×4 threads = 16, which is the #CPU cores I have.