FFmpeg Cheat Sheet for 360º video

Brought to you by Headjack

FFmpeg is one of the most powerful tools for video transcoding and manipulation, but it's fairly complex and confusing to use. That's why I decided to create this cheat sheet which shows some of the most often used commands.

Let's start with some basics:

- `ffmpeg` calls the FFmpeg application in the command line window, could also be the full path to the FFmpeg binary or .exe file
- `-i` is follwed by the path to the input video
- `-c:v` sets the video codec you want to use
  Options include `libx264` for H.264, `libx265` for H.265/HEVC, `libvpx-vp9` for VP9, and `copy` if you want to preserve the video codec of the input video
- `-b:v` sets the video bitrate, use a number followed by `M` to set value in Mbit/s, or `K` to set value in Kbit/s
- `-c:a` sets the audio codec you want to use Options include `aac` for use in combination with H.264 and H.265/HEVC, `libvorbis` for VP9, and `copy` if you want to preserve the audio codec of the input video
- `-b:a` sets the audio bitrate of the output video
- `-vf` sets so called video filters, which allow you to apply transformations on a video like `scale` for changing the resolution and `setdar` for setting an aspect ratio
- `-r` sets the frame rate of the output video
- `-pix_fmt` sets the pixel format of the output video, required for some input files and so recommended to always use and set to `yuv420p` for playback
- `-map` allows you to specify streams inside a file
- `-ss` seeks to the given timestamp in the format `HH:MM:SS`
- `-t` sets the time or duration of the output

Get video info

```
ffmpeg -i input.mp4
```

Transcode video

The simplest example to transcode an input video to *H.264*:

```
ffmpeg -i input.mp4 -c:v libx264 output.mp4
```

However, a more reasonable example, which includes setting an audio codec, setting the pixel format and both a video and audio bitrate, would be:

```
ffmpeg -i input.mp4 -c:v libx264 -b:v 30M -pix_fmt yuv420p -c:a aac -b:a 192K output.mp4
```

To tanscode to *H.265/HEVC* instead, all we do is change `libx264` to `libx265`:

```
ffmpeg -i input.mp4 -c:v libx265 -b:v 15M -pix_fmt yuv420p -c:a aac -b:a 192K output.mp4
```

iOS 11 and OSX 11 now support HEVC playback, but you have to make sure you use FFmpeg 3.4 or higher, and then add `-tag:v hvc1` to your encode, or else you won't be able to play the video on your Apple device.

For *VP9* we have to change both the video *and* the audio codec, as well as the file extension of the ouput video. We also added `-threads 16` to make sure FFmpeg uses multi-threaded rendering to speed things up significantly:

```
ffmpeg -i input.mp4 -threads 16 -c:v libvpx-vp9 -b:v 15M -pix_fmt yuv420p -c:a libvorbis -
b:a 192K output.webm
```

You may have noticed we also halved the video bitrate from `30M` for H.264 to `15M` for H.265/HEVC and VP9. This is because the latter ones are advanced codecs which output the same visual quality video at about half the bitrate of H.264. Sweet huh! They do take way longer to encode though and are not as widely supported as H.264 yet.

Hardware accelerated encoding

We just saw how to encode to *H.264* using the `libx264` codec, but the latest Zeranoe FFmpeg builds for Windows now support hardware accelerated encoding on machines with Nvidia GPUs (even older ones), which *significantly* speeds up the encoding process. You use this powerful feature by changing the `libx264` codec to `h264_nvenc`:

```
ffmpeg -i input.mp4 -c:v h264_nvenc output.mp4
```

To use hardware acceleration for *H.265/HEVC*, use `hevc_nvenc` instead:

```
ffmpeg -i input.mp4 -c:v hevc_nvenc output.mp4
```

If you get any error messages, either your FFmpeg version or your GPU does not support hardware acceleration, or you are using an unsupported `-pix_fmt`. There is unfortunately no hardware acceleration support in FFmpeg for the *VP9* codec.

We noticed one strange artefact when using `h264_nvenc` and `hevc_nvenc` in combination with scaling. For example, when we scaled a 4096x4096 video down to 3840x2160 pixels, the *height* of the output video showed correctly as 21**60** pixels, but the *stored_height* was 21**76** pixels for some reason, which causes issues when trying to play it back on Android 360º video players.

Resize video to UHD@30fps

At the moment, the most common playback resolution for 360º video is the UHD resolution of `3840x2160` at `30` frames per second. The commands we have to add for this are:

```
-vf scale=3840x2160,setdar=16:9 -r 30
```

Which results in something like this:

```
ffmpeg -i input.mp4 -vf scale=3840x2160,setdar=16:9 -r 30 -c:v libx265 -b:v 15M -pix_fmt
yuv420p -c:a aac -b:a 192K output.mp4
```

Add, remove, extract or replace audio

Add an audio stream to a video without re-encoding:

```
ffmpeg -i input.mp4 -i audio.aac -c copy output.mp4
```

However, in most cases you will have to re-encode the audio to fit your video container:

```
ffmpeg -i input.mp4 -i audio.wav -c:v copy -c:a aac output.mp4
```

Remove an audio stream from the input video using the `-an` command:

```
ffmpeg -i input.mp4 -c:v copy -an output.mp4
```

Extract an audio stream from the input video using the `-vn` command:

```
ffmpeg -i input.mp4 -vn -c:a copy output.aac
```

Replace an audio stream in a video using the `-map` command:

```
ffmpeg -i input.mp4 -i audio.wav -map 0:0 -map 1:0 -c:v copy -c:a aac output.mp4
```

You could add the `-shortest` command to force the output video to take the length of the shortest input file if the input audio file and the input video file are not exactly the same length

Sequence to video

Many high-end video pipelines work with DPX, EXR or TIFF sequences. To transform these sequences into video files, the easiest way is to specify the first file in the sequence as the input and then use `-framerate` to set the input frame rate and `-r` to set the output frame rate:

```
ffmpeg -i input_0001.dpx -framerate 59.94 -c:v libx264 -b:v 30M -r 29.97 -an output.mp4
```

Stereo to mono

We can use video filters to cut the bottom half of a stereoscopic top-bottom video to turn it into a monoscopic video:

```
ffmpeg -i input.mp4 -vf crop=h=in_h/2:y=0 -c:a copy output.mp4
```

Cut a piece out of a video

Use `-ss` to set the start time in the video and `-t` to set the duration of the segment you want to cut

```
ffmpeg -ss 00:01:32 -i input.mp4 -c:v copy -c:a copy -t 00:00:10 output.mp4
```

The above command seeks to 1.32 minutes in the video, and then outputs the next 10 seconds. As you can see, `-ss` is placed *before* the `-i` command, which results in way faster (but slightly less accurate) seeking.

Concatenate two videos

Concatenation is not possible with all video formats, but it works fine for MP4 files for example. There are a couple of ways to concatenate video files, but I will only describe the way that worked for me here, which requires you to create a `txt` file with the paths to the files you want to concatenate.

Only if the files you want to concatenate have the *exact same* encoding settings can you concatenate without re-encoding:

```
ffmpeg -f concat -i files.txt -c copy output.mp4
```

In the `files.txt` file, place urls to the files you want to concatenate:

```
file '/path/to/video1.mp4'
file '/path/to/video2.mp4'
file '/path/to/video3.mp4'
```

You can add `-safe 0` if you are using absolute paths. If you miss some frames after concatenation, keep in mind that the concatenation happens on I-frames, so if you don't cut at *exactly* the right frame, FFmpeg will discard all frames up to the nearest I-frame before concatenating.