Movie

Extract frames

- Extract keyframes
  - **-hide_banner** we are using this parameter to hide ffmpeg compilation information
  - **-vsync vfr**: This is a parameter that tells the filter to use a variable bitrate video synchronization. If we do not use this parameter ffmpeg will fail to find only the keyframes and shoud extract other frames that can be not processed correctly.

```
ffmpeg -i VIDEO0068.mp4 -vf "select=eq(pict_type\,I)" -vsync vfr thumb%04d.bmp -hide_banner
```

- Extract one frame per second

```
ffmpeg -i video.webm -vf fps=1 thumb%04d.jpg -hide_banner
```

- Extract one frame per 10 seconds

```
ffmpeg -i video.webm -vf fps=1/10 thumb%04d.jpg -hide_banner
```

- Extract with different quality / to bmp

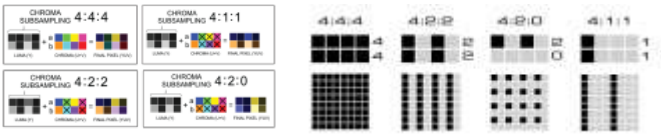
```
ffmpeg -i video.webm -vf fps=1 -quality 90 thumb%04d.jpg -hide_banner
```

```
ffmpeg -i video.webm -vf fps=1 thumb%04d.bmp -hide_banner
```

- Extract only one frame

```
ffmpeg -i video.webm -ss 00:00:07.000 -vframes 1 thumb.jpg
```

Chroma subsampling

https://en.wikipedia.org/wiki/Chroma_subsampling 

- j:a:b - j - horizontal sampling reference (usually 4). **a** number of chroma samples (Cr, Cb) in the first row of **j** pixels. **b** number of changes of chroma samples between firs and second row of **j** pixels.
- 4:4:4 - (H265, H264) - no chroma subsampling - for every 4 pixels of luma, there are 4 pixels of colour
- 4:2:2 - 50Mbit - (H264) for every 4 pixels of luma, there are 2 pixels of colour. Keying can produce bad edges.
- 4:2:0 - 35MBit - (DVD, DV, JPEG, MPEG1, Blueray) - only alternatign lines are sampled for chroma. Chroma vertical resolution is halved.

Stabilisation

To remove Little shaking in 1-pass

```
ffmpeg -threads 8 -i $1 -vf deshake -c:a copy -c:v utvideo yuv420p $1-deshake.mkv
```

NOTE: No multithreading.

To completely process shaky video in 2-pass|

FFMpeg already contains plugins **vidstabdetect** and **vidstabtransform** http://public.hronopik.de/vid.stab/download.php. But latest version (supports multithreading) can be downloaded as statically linked binary from: https://www.johnvansickle.com/ffmpeg/

```
ffmpeg -i $1 -vf vidstabdetect=shakiness=10:accuracy=15:result="tracefile.trf":show=1 -c:a copy -c:v utvideo ${PIX_FMT} ${OFILE}-vidstab1.mkv
ffmpeg -i $1 -vf vidstabtransform=input="tracefile.trf" -c:a copy -c:v utvideo ${PIX_FMT} ${OFILE}-vidstab2.mkv
```

NOTE: output from pass1 is also written as video to see motion vectors.

Lossless

Use UT Video codec as better alternative to HUFYUV: https://en.wikipedia.org/wiki/Ut_Video_Codec_Suite

- Download for Windows: https://www.videohelp.com/software/Ut-Video-Codec-Suite
- Supported by ffmpeg

```
ffmpeg -i $1 -vf deshake -c:a copy -c:v utvideo -pix_fmt yuv422p $1-deshake.mkv
```

Comparison of file size:

pixfmt	huffyuv	utvideo
444p		4104M
422p	4077M	3204M
420p		2556M

The same video encoded using **utvideo** @422p is 78% of **huffyuv**.

MPEG2

Highest quality (qscale 4-5 is far enough)

```
-c:v mpeg2video -qscale:v 2
```

DVD

DVD's VOBS can be simply concatenated to one big file:

```
cat ./VIDEO_TS/*.VOB | ffmpeg -i - <out_name>.<out_format>
```

```
#!/bin/bash -eu

VOBS=""
while read VOB; do
    echo ${VOB}
    if [ -n "${VOBS}" ]; then
    VOBS="${VOBS}|"
    fi
    VOBS="${VOBS}${VOB}"
done < <(ls -1 *.VOB)

ffmpeg -i "concat:${VOBS}" -map 0:0 -map 0:1 -map 0:2 -c copy out.ts
```

More: https://newspaint.wordpress.com/2016/07/27/ripping-a-video-from-dvd-using-ffmpeg/

H.264 params

CRF (Constant Rate Factor), 0..51 (0-lossless)

- 17..23 very good, no blocking efect during fast movement
- 23 - default

```
-c:v libx264 -crf 25
```

```
-c:v libx264 -b:v 1024k
```

FFMpeg params

2.5Mbps bitrate with tolerance 300k

```
-b 2500k -bt 300k
```

Resize

to half of size (using video filter https://ffmpeg.org/ffmpeg-filters.html#scale

```
-vf scale=w=iw/2:h=ih/2
```

Audio recompress

1 channel audio (mono), audio quality 7 https://trac.ffmpeg.org/wiki/Encode/MP3

```
-ac 1 -c:a libmp3lame -q:a 7
```

Examples

Change container

```
ffmpeg -i input.ts -vcodec copy -sameq -acodec copy -f matroska output.ts
```

Copy video and compress audio

Create mono mp3 audio stream:

```
ffmpeg -i input_file.avi -c:v copy -ac 1 -c:a libmp3lame -q:a 7 output.avi
```

Remove audio

```
ffmpeg -i example.mkv -c copy -an example-nosound.mkv
```

Recompress

```
ffmpeg -i input.mp4 -b 1000000 output.mp4
ffmpeg -i input.mp4 -vcodec libx264 -crf 20 output.mp4

ffmpeg -i input.mp4 -c:v libx264 -crf 23 output.mp4
ffmpeg -i input.ts -c:a copy -c:v libx264 -crf 18 -preset veryfast NL.mp4
ffmpeg -i input.mkv -c:v copy -c:a
```

Limit CPU usage to 150% (2 CPUs)

```
cpulimit -z -e ffmpeg -l 150
```

Cut

Cut input from 11:20 to 1:45:50:

Example below (option order -i and -ss) will decode input from beginning frame by frame and starts processing at 11:20 and stops at 1:45:50.

```
ffmpeg -i input.mp4 -ss 11:20 -to 1:45:50 -c:a copy -c:v copy  output.mp4
```

Next example (option order -ss and then -i) will seek to position 11:20 using keyframes and start processing for 1:34:30.

```
ffmpeg -ss 11:20 -i input.mp4 -to 1:34:30 -c:a copy -c:v copy  output.mp4
```

This is faster solution.

In both examples video codes is “copy” so seeking works using I-frames, so there is no speed difference. Additionally with “copy”, it is possibilty to select time range without keyframes, so only audio will be copied and video will start from next valid key frame.

Copy chosen streams:

```
ffmpeg -i test.ts -to 5:47 -map 0:0 -map0:1 -map 0:4 -c copy cut.mkv
```

Copy chosen streams and recompress video stream:

```
ffmpeg -i test.ts -to 5:47 -map 0:0 -map0:1 -map 0:4 -c copy -c:v libx264 -
crf 25 cut.mkv
```

Copy chosen streams and resize to 50%, recompress video stream:

```
ffmpeg -i test.ts -to 5:47 -map 0:0 -map0:1 -map 0:4 -c copy -c:v libx264 -
crf 25 -vf scale=w=iw/2:h=ih/2 cut.mkv
```

Deinterlace

```
 ffmpeg -i test.ts -vf yadif -vcodec ... -acodec ... test.mp4
```

vidia

https://www.maketecheasier.com/convert-video-to-mp4-handbrake-linux/

Issues

* **deprecated pixel format used, make sure you did set range correctly**

- when using **yuv420p** pixel format