Movie Compression with ffmpeg for media review.

FFmpeg is frequently used by different studios for encoding their media, however the documentation for ffmpeg is often poor, or cryptic so its often harder than it should be to come up with a good starting point. We are aiming to come up with recommendations for different scenarios as well as document what the different flags are doing with the aim to make this easier to get to a good baseline.

Overview

We are looking for recommendations for the following:

- Best color preservation for output to:
    - Web, OSX, IOS and Windows.
    - Common applications: e.g. RV, Nuke.
    - Rec709 and sRGB displays to start with, but eventually, P3, rec2020 and HDR displays.
    - Web browser - Firefox reviewing mp4. - use firefox plugin.
    - RV
- Codec recommendations for:
    - Proxy H264 playback (e.g. web streaming), should be setup for web streaming.
    - Animation/Modelling/Layout movie playback. - somewhat lower quality playback, but should always provide smooth motion.
    - Lookdev/lighting/compositing movie playback - should have excellent color fidelity and minimal encoding artifacts
        - Should any filmlook be baked in, or should we assume that is always applied during viewing.
        - How much should we be able to adjust color and have the image hold up? (Or rely on exr's for that?).
    - Export to editorial.
    - High-resolution or frame rate - e.g. 4k, 8k, 60fps, 120fps.
    - Stereo or VR.
- Q: Which container should we be considering: mov, mp4, mxf.

Where ffmpeg arguments, it would be great to document why we are using them, rather than ending up with a recipe.

Presentation Links

Overall workflow

For this paper, we are assuming that we are encoding from a file-sequence of frames into a movie (rather than re-encoding a movie), but we are also assuming almost all of the colorspace work would be done outside of ffmpeg, with tools using the OCIO library. Examples could include nuke and oiio. Once we get to ffmpeg, the goal being that the pixel data we get in, should be as close as possible to the data we get out of ffmpeg. However, there are still quite a few areas that a ffmpeg user could go wrong, which we break down below.

TODO: Find examples of overall workflow.

Codec Selection.

Name	Target Usage	Source	ffmpeg flags	Description	Size
libx264 -pix_fmt yuv420p	Proxy playback, for web review, and non-color critical workflows, e.g. animation, modeling, etc.			This should be a lightweight compression, capable of supporting HD with a reasonable bit-rate, hopefully supporting a wide range of web browsers. Final review, e.g. lighting	
libx264 -pix_fmt yuv444p10le	Final review, e.g. lighting	https://trac.ffmpeg.org/wiki/colorspace	-c:v libx264 -preset placebo -qp 0 -x264-params "keyint=15:no-deblock=1" -pix_fmt yuv444p10le -sws_flags spline+accurate_rnd+full_chroma_int -vf "colorspace=bt709:iall=bt601-6-625:fast=1" -color_range 1 -colorspace 1 -color_primaries 1 -color_trc 1		
libx264rgb	Final review, e.g. lighting			This is a variant of the above, its essentially using x264, but not converting to YCrCb.	
libx265			-c:v libx264		
Prores 4444	For delivery to editorial		-c:v prores_ks -profile:v 4444 -qscale:v 1 -pix_fmt yuv444p10le -sws_flags spline+accurate_rnd+full_chroma_int -vf "colorspace=bt709:iall=bt601-6-625:fast=1" -color_range 1 -colorspace 1 -color_primaries 1 -color_trc 1		
-profile:v 4444 is equivalent to -profile:v 4					
shotgun_diy_encode		https://support.shotgunsoftware.com/hc/en-us/articles/219030418-Do-it-yourself-DIY-transcoding',	-vcodec libx264 -pix_fmt yuv420p -g 30 -vprofile high -bf 0 -crf 2		
DnxHD	For delivery to editorial				
Prores 422 HQ	For delivery to editorial	Some FFMpeg commands I need to remember for converting footage for video editing. http://bit.ly/vidsnippets · GitHub	-pix_fmt yuv422p10le -c:v prores_ks -profile:v 3 -vendor ap10 -sws_flags spline+accurate_rnd+full_chroma_int -vf "colorspace=bt709:iall=bt601-6-625:fast=1" -color_range 1 -colorspace 1 -color_primaries 1 -color_trc 1		

Note the -vendor ap10 part below is only needed if working with Final Cut, but it does no harm otherwise.

-profile:v 3 is equivalent to -profile:v hq

h264

Key flags (see https://trac.ffmpeg.org/wiki/Encode/H.264)

- -crf 23 - This is the constant rate factor, controlling the default quality (see: https://slhck.info/video/2017/02/24/crf-guide.html) where -crf 0 is uncompressed. By default this is set to 23, which is probably good enough for our needs.
- -qp 23 - Quantization Parameter - it is recommended that you do not use this, in preference to -crf above (see: https://slhck.info/video/2017/03/01/rate-control.html)
- -preset slower - https://trac.ffmpeg.org/wiki/Encode/H.264#FAQ
- -tune film - Optionally use the tune option to change settings based on specific inputs - https://trac.ffmpeg.org/wiki/Encode/H.264#FAQ - see also: https://superuser.com/questions/564402/explanation-of-x264-tune I suspect that we would want to use one of:
    1. -tune film good for live action content.
    2. -tune animation good for animated content with areas of flat colors.
    3. -tune grain good for live action content where you want to preserve the grain as much as possible.
- -qscale:v 1 - Generic quality scale flag: https://www.ffmpeg.org/ffmpeg.html#toc-Main-options - not sure if its needed?

-preset slow -crf 18 level 4 profile high.

RV - 10bit YUV444.

Better to stick with yuv –

FFMPEG.

TODO:

- Suggestions for max-bitrate?
- Suggestions for preset - ? slow
- Suggestions for tune

h265/HVEC

Support: https://caniuse.com/hevc (or https://www.chromium.org/audio-video) currently no support for h265 on chrome or chromium based browsers.

links:

- https://codecalamity.com/encoding-uhd-4k-hdr10-videos-with-ffmpeg/
- https://codecalamity.com/encoding-settings-for-hdr-4k-videos-using-10-bit-x265/
- https://support.frame.io/en/articles/4305241-creating-hdr-files-for-frame-io
- https://stackoverflow.com/questions/69251960/how-can-i-encode-rgb-images-into-hdr10-videos-in-ffmpeg-command-line
- https://brandur.org/fragments/ffmpeg-h265

HDR

- https://developer.apple.com/av-foundation/High-Dynamic-Range-Metadata-for-Apple-Devices.pdf
- https://www.avsforum.com/threads/open-source-video-testing-calibration-patterns.2944378/
- https://github.com/test-full-band/tfb-video/releases
- https://trev16.hatenablog.com/entry/2021/07/23/145725 – good site for HDR test sites.

ProRes

There are four Prores encoders, Prores, Prores_ks, Prores_aw and now with ffmpeg 5 VideoToolBox Prores, which is a hardware based OSX M1 encoder /decoder.

From https://trac.ffmpeg.org/wiki/Encode/VFX the recommendation is to use Prores_ks with -profile:v 3 and the qscale of 11

Options that can be used include:

- -profile:v values can be one of.
    - `proxy` (0)
    - `lt` (1)
    - `standard` (2)
    - `hq` (3)
    - `4444` (4)
    - `4444xq` (5)
- -qscale:v between values of 9 - 13 give a good result, 0 being best.
- -vendor apl0 - tricks the codec into believing its from an Apple codec.

Using this with the usual color space flags, seems to work well with the exception of ffmpeg itself, which needs the flags:*-vf scale=in_color_matrix=bt709:out_color_matrix=bt709* added to the command to ensure the right input colorspace is recognised, e.g.:

```
ffmpeg.exe -i INPUTFILE.mov -compression_level 10 -pred mixed -pix_fmt rgba64be -sws_flags
spline+accurate_rnd+full_chroma_int -vframes 1 -vf scale=in_color_matrix=bt709:out_color_matrix=bt709 OUTPUTFILE.
png
```

However, other encoders seem to be recognised correctly, so there is clearly some metadata missing. I did try using the prores_metadata filter to try adding some additional parameters, but it didnt seem to help.

```
ffmpeg.exe -i ./chip-chart-yuvconvert\basicnclc.mov -c copy -bsf:v prores_metadata=color_primaries=bt709:
color_trc=bt709:colorspace=bt709 chip-chart-yuvconvert\basicnclcmetadata.mov
```

TODO:

- Figure out the missing metadata.
- Wedge qscale values
- Do some colorspace tests with different qscale values to see where color breaks down.

VMAF

I did explore using VMAF - Video Multi-Method Assessment Fusion as a way to quantify the compression, the notes for setting this up are below, however I think we are going with a fairly high compression factor , so I think this is probably not really going to help us much.

https://github.com/Netflix/vmaf

https://jina-liu.medium.com/a-practical-guide-for-vmaf-481b4d420d9c

https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652

https://ottverse.com/vmaf-ffmpeg-ubuntu-compilation-installation-usage-guide/ - building VMAF on ubuntu.

Color Preservation

Testing Methodology

Converting SMPTE color bars to the compressed movie, using ffmpeg to expand and then compare with OIIO. NOTE, for compression schemes that are not 444 we may need to mask the transitions.

Testing loading the compressed movie in to RV, Firefox, VLC, Avid, resolve, , to compare the resulting color transformation - not sure if there is a procedural way to run this?

For the tests below we are assuming that other tools are being used (e.g. oiiotool) to convert the rendered frames into an intermediate file (e.g. PNG) in the target color-space.

Q: Currently focusing just on color matching in vs. out, but should also do EXR ACEScg in to resulting movie. Feels like we should also bless full pipeline, e.g.: Reference "Dailies script" https://github.com/jedypod/generate-dailies

Test Sources

SMPTE test chart: https://commons.wikimedia.org/wiki/File:SMPTE_Color_Bars_16x9.svg

Download image sequence from: https://senkorasic.com/testmedia/ -

Explore netflix: https://opencontent.netflix.com/

Test Results:

taurich.org/encodingTests/results.html

Links

- An excellent starting point for this is: https://trac.ffmpeg.org/wiki/colorspace
- https://github.com/RxLaboratory/DuME/blob/master/src/FFmpeg_COLORS.md
- https://medium.com/invideo-io/talking-about-colorspaces-and-ffmpeg-f6d0b037cc2f
- https://docs.nvidia.com/video-technologies/video-codec-sdk/ffmpeg-with-nvidia-gpu/
- https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1886-0-201103-I!!PDF-E.pdf - the BT1886 spec, esentially gamma 2.4 rec709 primaries.

Notes

The big issue here is that by default if you start converting images to another format, and ffmpeg cannot determine the colorspace it will default to bt601. So many of the flags below are to:

A: Tell ffmpeg that the source media is in fact bt709

B: Add the metadata to the output, so that other future conversions also know how to convert it back.

C: Do as clean a conversion from RGB to YUV as possible.

RGB/YCrCb Colorspace Conversion

As a rule of thumb, we would like ffmpeg to do as little as possible in terms of color space conversion. i.e. what comes in goes out. The problem is that most of the codecs are doing some sort of RGB to YUV conversion (technically YCrCb). The notable exception is x264rgb (see above). For more information, see: https://trac.ffmpeg.org/wiki/colorspace

For examples comparing these see: https://richardssam.github.io/ffmpeg-tests/tests/chip-chart-yuvconvert/compare.html

colormatrix filter

```
ffmpeg -y -i ../sourceimages/chip-chart-1080-noicc.png -sws_flags spline+accurate_rnd+full_chroma_int -vf
"colormatrix=bt470bg:bt709" -c:v libx264 -preset placebo -qp 0 -x264-params "keyint=15:no-deblock=1" -pix_fmt
yuv444p10le -qscale:v 1 -color_range 1 -colorspace 1 -color_primaries 1 -color_trc 1 ./chip-chart-yuvconvert
/spline444colormatrix2.mp4
```

This is the most basic colorspace filtering. bt470bg is essentially part of the bt601 spec. See: https://www.ffmpeg.org/ffmpeg-filters.html#colormatrix

colorspace filter

```
ffmpeg -y -i ../sourceimages/chip-chart-1080-noicc.png -sws_flags spline+accurate_rnd+full_chroma_int -vf
"colorspace=bt709:iall=bt601-6-625:fast=1" -c:v libx264 -preset placebo -qp 0 -x264-params "keyint=15:no-
deblock=1" -pix_fmt yuv444p10le -qscale:v 1 -color_range 1 -colorspace 1 -color_primaries 1 -color_trc 1 ./chip-
chart-yuvconvert/spline444colorspace.mp4
```

Using colorspace filter, better quality filter, SIMD so faster too, can support 10-bit too. The second part -vf "colorspace=bt709:iall=bt601-6-625:fast=1" encodes for the output being bt709, rather than the default bt601 matrix. iall=bt601-6-625 says to treat all the input (colorspace, primaries and transfer function) with the bt601-6-625 label). fast=1 skips gamma/primary conversion in a mathematically correct way. See: https://ffmpeg.org/ffmpeg-filters.html#colorspace

libswscale filter

```
ffmpeg -y -i ../sourceimages/chip-chart-1080-noicc.png -sws_flags
spline+accurate_rnd+full_chroma_int+full_chroma_inp -vf "scale=in_range=full:in_color_matrix=bt709:out_range=tv:
out_color_matrix=bt709" -c:v libx264 -preset placebo -qp 0 -x264-params "keyint=15:no-deblock=1" -pix_fmt
yuv444p10le -qscale:v 1 -color_range 1 -colorspace 1 -color_primaries 1 -color_trc 1 ./chip-chart-yuvconvert
/spline444out_color_matrix.mp4
```

Using the libswscale library. Seems similar to colorspace, but with image resizing, and levels built in. https://www.ffmpeg.org/ffmpeg-filters.html#scale-1

This is the recommended filter.

Color Metadata NCLC/NCLX

The above gets the underlying data stored correctly, but there are additional metadata flags that can be set that are interpreted by some players, these are the NCLC color tags for color primaries, transfer function and conversion matrix. This is defined as a ISO spec here (see https://www.iso.org/standard/73412.html). The numbers below are part of the definition.

NCLC stands for Non-Consistent Luminance Coding, a brief overview of its history is here. For MP4 files, its also known as NCLX. Additionally this metadata can also be represented in the h264 metadata stream in the video usability Information (VUI) block.

You can read the metadata using mp4box.js which is a visual browser of the mp4 metadata, and look at moov/trak/mdia/minf/stbl/stsd/avc1/colr

NOTE: None of the flags below affect the encoding of the source imagery, they are meant to be used to guide how the mp4 file is decoded.

The docs are pretty sparse for this, some of the better info is FFmpeg/pixfmt.h at master · FFmpeg/FFmpeg · GitHub

links:

Color Range

Uses the flag -color_range e.g. -color_range 1 or -color_range tv

Numeric value	String Values	Numeric range	Notes
0			Unspecified
1	tv mpeg	16-135	This is the default.
2	pc jpeg	0-255	

Color Space

This defines the YUV colorspace type, as defined by ISO/IEC 23091-2_2019 subclause 8.3

Use flag -colorspace, e.g. -colorspace 1 or -colorspace rec709

This is a subset of the full list of values, for more details, see FFmpeg/pixfmt.h at master · FFmpeg/FFmpeg · GitHub

Numeric Value	String values	Description
0	rgb	
1	bt709	Typically set it to this.
2		unspecified
9	bt2020nc bt2020_ncl	ITU-R BT2020 non-constant luminance system
10	bt2020c bt2020_cl	ITU-R BT2020 constant luminance system

Color Primaries

Chromaticity coordinates of the source primaries. These values match the ones defined by ISO/IEC 23091-2_2019 subclause 8.1 and ITU-T H.273.

This is defining your color gamut, so you typically want to be setting this for -color_primaries 1 unless you are working in bt2020.

This is a subset of the full list of values, for more details, see FFmpeg/pixfmt.h at master · FFmpeg/FFmpeg · GitHub

Numeric Value	String Values	Description
1	bt709	
9	bt2020	
11		DCI P3
12		P3 D65 / Display P3

Your browser does not support the HTML5 video element

Color Transfer Characteristic aka color_trc

These values match the ones defined by ISO/IEC 23091-2_2019 subclause 8.2.

This is defines the OETF, which typically is the gamma.

Numeric Value	String Values	Description
1	bt709	Note this is the camera gamma i.e. ~1.95 this is NOT bt1886
2		Image characteristics are unknown or are determined by the application.
4	gamma22	
5	gamma28	
8	linear	Linear
9	log log100	

13	iec61966_2_1	IEC 61966-2-1 or sRGB or sYCC
14	bt2020_10 bt2020_10bit	Note this is the camera gamma i.e. ~1.95
15	bt2020_12 bt2020_12bit	Note this is the camera gamma i.e. ~1.95
16	smpte2084	bt2100-1 perceptual quantization (PQ) system.
17	smpte428	SMPTE ST 428-1 - DCI ?
18	arib-std-b67	ARIB STD-B67 bg2100-1 hybrid log-gamma (HLG) system

NOTE: -color_trc 1 - is not bt1886, but is actually the camera gamma, so has a gamma of ~1.95 rather than the 2.4 that is defined by bt1886. In order to get a gamma 2.4, you will need to use a quicktime hack (see below), but this only works on OSX. However, we suspect that chrome ignores the setting (see the following tests).

The following page shows what applying different color TRC values to the same source image:

https://taurich.org/encodingTests/ICCTest/greyramp/compare.html

What you may notice is on Chrome on windows, there is a slight color shift when compared to the PNG file. The other thing that is odd is that the bt709 flag doesnt actually seem to be doing anything, it functions identically to color-trc=2 which is a 'no-op". Its possible that this was picked deliberately due to too many people incorrectly assuming it was bt1886.

This second test highlights that better, by giving a source image that is designed so that when the images are displayed with the TRC settings they should match with a gamma 2.2 monitor.

https://taurich.org/encodingTests/ICCTest/greyramp-rev-ps/compare.html https://www.color.org/version4html.xalter

Again, you will notice the bt709 (color-trc=1) is wildly off.

For more information on this I recommend:

- https://vimeo.com/349868875
- https://developer.apple.com/documentation/avfoundation/media_assets_and_metadata/sample-level_reading_and_writing/tagging_media_with_video_color_information
- https://www.iso.org/standard/73412.html - Note this has a link to the download of the earlier version of the doc, the latest and paywalled version is here: https://www.iso.org/standard/57794.html
- https://github.com/bbc/qtff-parameter-editor - A BBC open source app, for setting quicktime NCLC attributes.
- https://vimeo.com/349868875 - Video from baselight, reviewing setting the right NCLC tags for apple colorsync.

Web Browser Deliverables

How should we be encoding content for a web browser.

Most windows laptops and most monitors typically default to a sRGB color space, the tricky part is that sRGB is sometimes interpreted as having exactly a 2.2 gamma, and sometimes a hybrid curve (based on the spec), for more details on this, see: https://vimeo.com/442069591

Questions to be answered:

- On windows do any of the browsers read the color management settings flags for each monitor?
- Why does everybody set -color_trc 1 ? - it seems completely meaningless?
- Find the details about the firefox plugin for color management?
- What do ICC profiles for stills do on windows/linux boxes? - Are there situations where this is replicated for movies.
- Color shift on Chrome, reported: https://bugs.chromium.org/p/chromium/issues/detail?id=1262622#makechanges

Browser	Platform	Interpret NCLC flags	Color Managed	Tested	Notes
Firefox	OSX	No			
Firefox	Windows	No			
Firefox	Windows				
Safari	OSX	Yes	Yes		
Chrome	OSX	Yes	Yes		
Safari	IOS	No			

Chrome	Windows	Sometimes			Seems to occasionally stop working, it could be related to multiple screens.
Chrome	Linux				
Edge	Windows	Sometimes			Seems to occasionally stop working, it could be related to multiple screens.

Gamma 2.4

There is not a color_trc flag for gamma 2.4, the only option that exists for OSX is a cheat

using the flags "-color_trc 2 -movflags write_colr+write_gama -mov_gamma 2.4" but this only works for a full quicktime file, not a mp4 file. So the resulting file will not play correctly on windows.

-color_trc 2 – means the transfer function is unspecified.

-movflags write_colr+write_gama -mov_gamma 2.4 – allows you to specify a gamma parameter directly to the quicktime file.

e.g.

ffmpeg -y -i chip-chart-1080.png -c:v libx264 -pix_fmt yuv444p -qscale:v 1 -sws_flags spline+accurate_rnd+full_chroma_int -vf "colorspace=bt709:iall=bt601-6-625:fast=1" -color_range 1 -colorspace 1 -color_primaries 1 -color_trc 2 -movflags write_colr+write_gama -mov_gamma 2.4 test2-h264-ffmpeg-yuv444p-gamma24.mp4

Full range vs. legal range

Typically x264 (and other codecs) are following the video standard that lumance is scaled to the range 16-235. This has a history from early signaling where 236-255 were used for signaling and 0-15 to avoid any noise in the low end (some of the logic was derived from analog video)

However, that means that when we do the conversion, we can end up with 235-16 = 219 luminance values, rather than 255 (14% less levels). This is actually supported in web browsers, e.g.: chrome, firefox, safari.

The following web page demonstrates the resulting differences:

https://taurich.org/encodingTests/ICCTest/greyramp-fulltv/compare.html

There are two ways to do the conversion:

ffmpeg -y -i radialgrad.png -sws_flags spline+accurate_rnd+full_chroma_int -vf "scale=in_range=full:in_color_matrix=bt709:out_range=full:out_color_matrix=bt709" -c:v libx264 -pix_fmt yuvj420p -qscale:v 1 -color_range 2 -colorspace 1 -color_primaries 1 -color_trc 1 ./greyramp-fulltv/greyscale-fullj.mp4

or

ffmpeg -y -i radialgrad.png -sws_flags spline+accurate_rnd+full_chroma_int -vf "scale=in_range=full:in_color_matrix=bt709:out_range=full:out_color_matrix=bt709" -c:v libx264 -pix_fmt yuv420p -qscale:v 1 -color_range 2 -colorspace 1 -color_primaries 1 -color_trc 1 ./greyramp-fulltv/greyscale-full.mp4

TODO:

- Do tests of what happens when ffmpeg then converts the resulting file format, to ensure that the correct range is read.
- Find reference for archaic legal range.

References:

- https://news.ycombinator.com/item?id=20036710

Other links

https://github.com/bbc/qtff-parameter-editor